

View-Based Reverse Engineering Approach for Enhancing Model Interoperability and Reusability in Process-Driven SOAs

Huy Tran and Uwe Zdun and Schahram Dustdar

Distributed Systems Group

Information Systems Institute

Vienna University of Technology, Austria

`htran,zdun,dustdar@infosys.tuwien.ac.at`

Abstract. In many companies, process-driven SOAs are introduced by using technical process languages, such as BPEL, to orchestrate services. However, the process models developed using this approach are often too complex and hard to reuse because all process-related concerns are tangled in only one type of model. To make the models more understandable for non-technical stakeholders, many companies additionally introduce high-level process descriptions, e.g., specified in BPMN or EPCs, to offer a non-technical view of the processes. This divergence of process languages, however, often leads to inconsistencies after a few evolution steps. We propose in a novel approach based on the concept of architectural views that not only offers models tailored to the various stakeholders' concerns but also provides an automated integration of models at different abstraction levels. In particular, we propose an extensible reverse-engineering tool-chain to automatically populate various view models with information from existing process descriptions, and generate executable code from these view models via our view-based modeling framework.

1 Introduction

In a process-driven, service-oriented architecture (SOA), business functionality is accomplished by executing business processes consisting of coordinated activities that invoke various services. A typical business process includes a number of activities and a control flow. Each activity corresponds to a communication task (e.g., it invokes other services or processes) or a data processing task. The control flow describes how these activities are orchestrated to achieve a certain goal. A process is typically represented in either an executable language such as BPEL4WS [7], or XPDL [24], etc., or a high-level modeling language such as BPMN [15], EPC [10, 21], or UML Activity Diagram extensions, [14].

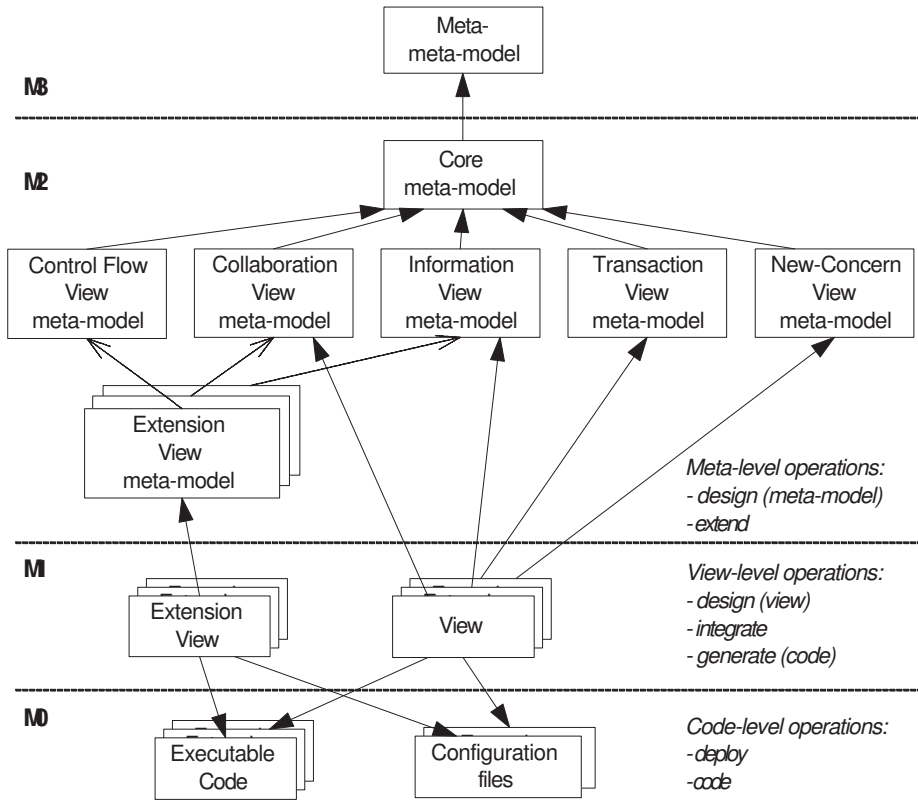
Nowadays, business process developers have to deal with increasing needs for change, for instance, concerning business requirement changes or IT technology changes. Therefore, the process models should

enable a quicker reaction on business changes in the IT by manipulating business process models instead of code. Unfortunately, to the best of our knowledge, most of the existing business processes are developed and maintained by technical experts (aka the IT experts) in low-level, executable languages. It is difficult for the business analysts to get involved in process development and maintenance because for these tasks an understanding of many technical details is required. Hence, technical experts are required for many task in managing, developing, and maintaining the process models. For each change, regarding both business and technology related concerns, the technical experts have to investigate, analyze and modify a number of executable code fragments and/or related process models, which is costly and error-prone. This issue occurs because the process models integrate various tangled concerns. As a consequence, the process models become too complex and the various process concerns are hard to reuse. In addition, there is a lack of adaptation of process models to suit the needs and focus of a particular stakeholder, for instance, a business analyst or technical expert.

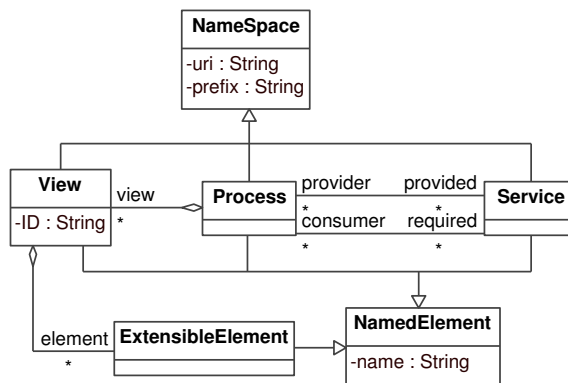
As a solution to these problems, some companies introduce high-level process descriptions, for instance, specified in BPMN or EPCs, to offer a non-technical view of the processes. However, this practice leads to yet another problem, namely, the divergence of process representations. That is, various more and less abstract descriptions of each business process are created. As there is no explicit (i.e., automated) trace link between the high-level process models and the implementation code/executable models for the processes, both of them might quickly become outdated or inconsistent as changes occur. As a consequence, neither the information in the high-level models is reused for defining the technical models, nor vice versa.

To the best of our knowledge, all aforementioned challenges have not been well exploited or targeted in the context of process-driven SOAs yet. We present in this paper a novel view-based reverse engineering approach for addressing these challenges. Our approach harnesses the *concept of architectural view* and the *partial interpreter* pattern to adapt process models to suit the requirements of a particular stakeholder. Using the partial interpreter pattern, we devise a number of interpreters to extract more and less abstract views from process descriptions. This way, different kinds of process modeling languages can be analyzed to build up the relevant representations (or views). The relationships between these process languages are maintained via the integration of the resulting views in our view-based modeling framework (VbMF) [20]. Next, using mechanisms such as extension mechanism, view integration and code generation developed in VbMF, the views can be manipulated to produce more appropriate representations according to stakeholders' requirements, or to re-generate code in executable languages. VbMF not only supports the reuse of information in process models at different abstraction levels and in different process concerns, but also the reuse of information in existing process models, e.g. written in BPEL.

The rest of paper is organized as follows. We give a short introduction of our view-based modeling framework in Section 2. Section 3 describes the view-based reverse engineering approach we propose in this paper. In Section 4, we present the details of using view-based interpreters to analyze existing business processes and extract various architectural views from the processes. Section 5 discusses the related work of our approach. Finally, Section 6 summarizes the main points of our contributions.



(a) View-based model-driven framework



(b) The core meta-model

Figure 1. The VbMF modeling framework and the Core meta-model

2 The View-based Modeling Framework

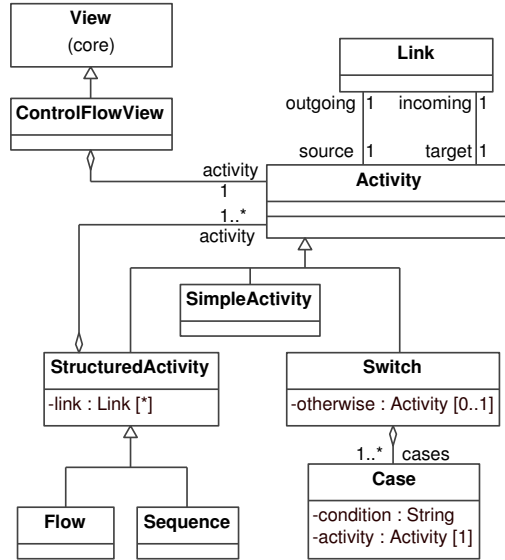
2.1 Overview of the view-based modeling framework

The view-based modeling framework [20] is based on the concept of architectural views. An architectural view is a representation of a system from the perspective of a related set of concerns [8]. Each particular concern is (semi-)formalized by a respective meta-model. A meta-model specifies entities and their relationships that appear in the correspondent view. VbMF defines a number of meta-models that conform to a common meta-meta-model (see Figure 1(a)). This way, VbMF separates process concerns into a number of architectural views. Furthermore, VbMF exploits the model-driven architecture approach [22] to separate the platform-neutral models from the platform-specific models. For this reason, VbMF also separates process models into different levels of abstraction. A meta-model at a lower abstraction level is defined as an extension of the meta-models at higher levels. VbMF's meta-models are either directly or indirectly derived the Core meta-model (shown Figure 1(b)) and therefore their relationships (aka trace links) are explicitly maintained via the model-driven architecture. These relationships enable VbMF to bridge the gaps between meta-models at different abstraction levels and to propagate changes. VbMF is able to generate code in executable language that can be deployed on existing process engines.

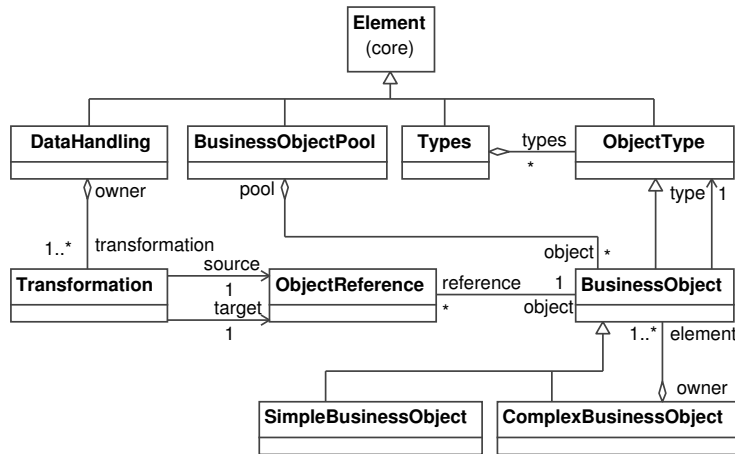
The Core meta-model defines basic, generic elements and appropriate relationship between these elements. Other view meta-models in VbMF are essentially defined in the same way: by extending the model elements provided as integration and extension points in the Core meta-model. Using the meta-models, we develop models that specify appropriate views. Example meta-models that we have derived from the Core meta-model are used to define the following views: Control Flow, Collaboration and Information View (see Figure 2(a), 3(a) and 2(b)), respectively. This way, the more specific extension view meta-models can be specified as extensions of elements in their correspondent root meta-models and elements in the Core meta-model.

For particular technologies, for instance, BPEL/WSDL, the extension mechanisms can be used to enrich existing abstract meta-models with additional details required to represent the specifics of those technologies. To illustrate the extension mechanisms, we present a BPEL-specific collaboration view meta-model in Figure 3(b), which is defined by extending the elements from Figures 1(b) and 3(a). Accordingly, we can use the distinction of the Core meta-model, generic view meta-models, and extension meta-models to represent different abstraction levels, such as business-level concerns and technical concerns.

In our implementation of these concepts, we exploit the model-driven software development (MDSD) paradigm [22] to separate the platform-neutral views from the platform-specific views so that the business experts – in their views – can get rid of technical details. Platform-specific models or executable code, for instance, Java, or BPEL and WSDL descriptions, can be generated from the views by using model-to-code transformations. The separation of view abstraction levels helps in enhancing the adaptability of the process-driven SOA models to business changes. For instance, the business experts analyze and modify the abstract views to meet the requirement of changes. Then, these modifications can be transformed into



(a) Control Flow Meta-model



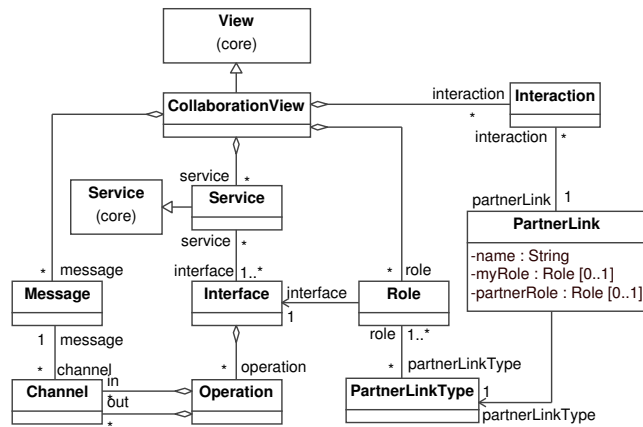
(b) Information View Meta-model

Figure 2. Basic concern meta-models

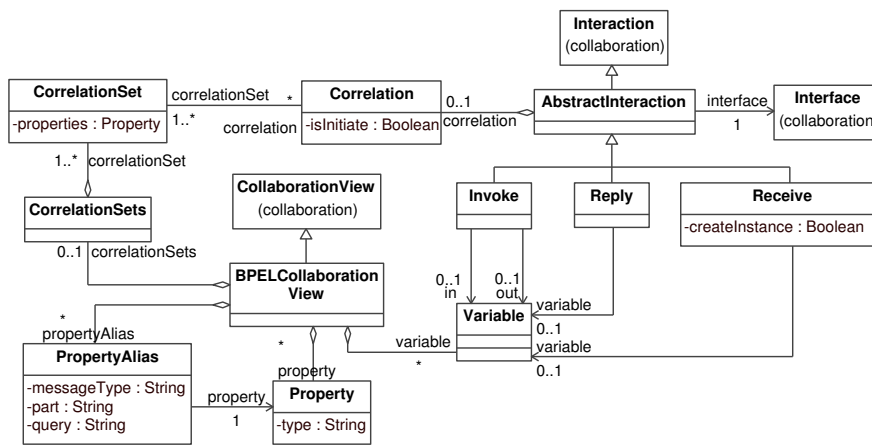
code in executable languages. The technical experts work with platform-specific views to define necessary configurations such that the generated code can be deployed into the corresponding runtime (i.e., process engines and Web service frameworks).

We have realized VbMF in openArchitectureWare framework (oAW) [16], a model-driven software development tool, and all meta-models are defined using the meta-meta-model of the Eclipse Modeling Framework [5]. The model-driven software development tool generates outputs, such as executable code, e.g., in BPEL and WSDL. The tools allow stakeholders of a process to only view a specific perspective, by examining a single view, or to analyze any combination of a number of views (i.e., to produce an integrated view).

To demonstrate our approach, we have exemplified it using the technology combination of BPEL4WS



(a) Collaboration View Meta-model



(b) BPEL extension of the Collaboration View

Figure 3. Basic concern meta-models and a BPEL extension meta-model example

and WSDL, which are likely the most popular process/service modeling descriptions used by numerous companies today. Nevertheless, in general, the same approach can be taken for any other process-driven SOA technologies by defining respective meta-models.

2.2 View-based reverse engineering tool-chain

VbMF mainly consists of a top-down (aka forward engineering) tool-chain (see Figure 4) in which the stakeholders can develop process-driven models in terms of architectural views, can generate process code from these views, or can extend the modeling framework into other process concerns by adding new meta-models or by enhancing existing meta-models.

Companies today have built up a vast amount of legacy process representations, either high-level or low-level, but there is no existing proper integration of these process descriptions, and no appropriate adaptation of process models to the stakeholders' needs and focus. In addition, typically off-the-shelf process modeling tools, such as BPEEL or BPMN tools, are used, and hence it is required to integrate them

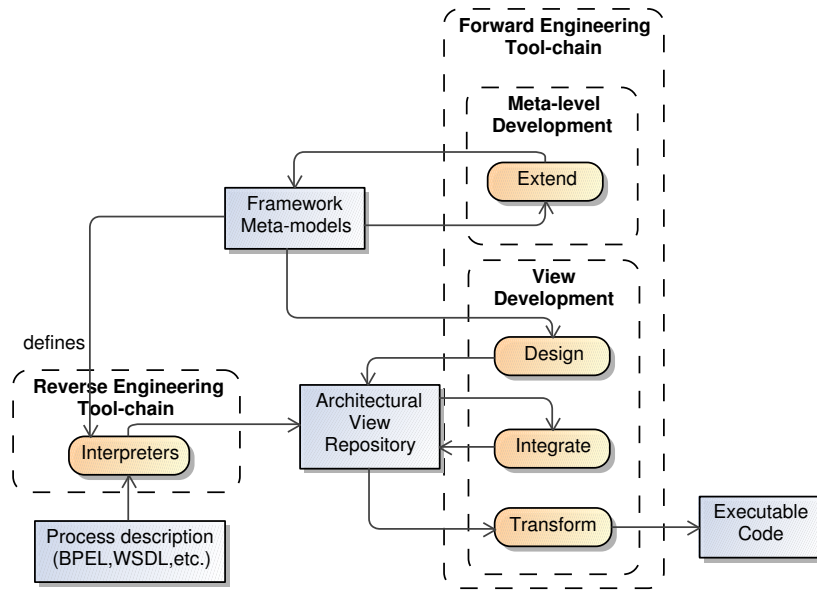


Figure 4. The extended VbMF including the view-based reverse-engineering

into VbMF. For these reasons, we extend VbMF with a bottom-up (aka reverse-engineering) tool-chain used for adapting process models and integrating various modeling representations. The outcome of the bottom-up tool-chain is a number of tailored views that are relevant to stakeholders. These views can be put into a common repository, and then can be re-used in other processes, or can be manipulated to re-generate new executable code which corresponds to any change of the corresponding views (see Figure 4).

3 View-based Reverse Engineering Approach

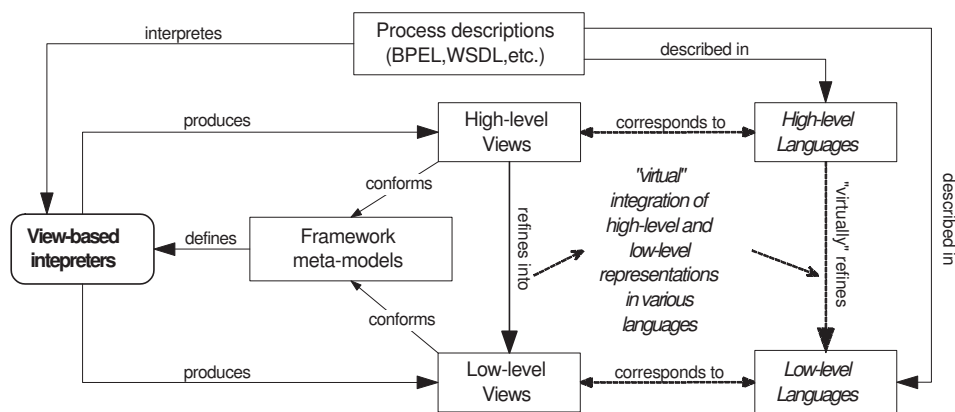


Figure 5. The view-based reverse engineering tool-chain

In the context of process-driven SOAs, many existing systems have built up an enormous repository of existing process code in executable languages, such as BPEL and WSDL. There are two important issues that have not been solved yet. Firstly, such process code integrates many tangled concerns such as message exchanges, data processing, service invocations, fault handling, transactions, and so forth.

Secondly, these languages are rather technology-specific and therefore the abstract representations are not explicitly available at the code level. As a result, the process models become too complex for stakeholders to understand and maintain, to integrate, to cooperate with other processes, or to re-use process models from existing modeling tools, etc.

Our view-based approach introduced in the previous section can potentially resolve these issues. However, for the budgetary reasons, developing the view models, required in our approach, from scratch is a poor and costly option. The alternative is an (automated) re-engineering approach comprising two activities: *reverse-engineering* for building more appropriate and relevant representations of the legacy code and *forward-engineering* for manipulating the process models, and for re-generating certain parts of the process code. During the reverse engineering process, high-level, abstract and low-level, technology-specific views on the process models are recovered from the existing code. This way, the reverse engineering approach helps stakeholders to get involved in process re-development and maintenance at different abstraction levels. An appropriate reverse engineering of business processes should not only help to adapt process models to stakeholder needs but also offer the ability to integrate various process models to enhance the interoperability of process models. The view-based reverse engineering approach we propose in this paper aims at achieving these goals. In the subsequent sections, we present this approach in terms of a tool-chain that fits into the View-based Modeling Framework (VbMF).

3.1 The Reverse Engineering Tool-chain

The reverse engineering tool-chain (see Figure 5) consists of a number of view-based interpreters, such as *control flow interpreter*, *information view interpreter*, and *collaboration view interpreter*, and so forth. Every interpreter is responsible for interpreting and extracting the corresponding view from the process descriptions. In VbMF, a particular view conforms to its meta-model. Therefore, the interpreter of 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*, *Case* according to the *control flow view meta-model* (see Figure 2(a)). In order to extract the control flow view from process descriptions, the interpreter walks through the input descriptions to pick the above-mentioned elements. Other elements are ignored.

3.2 General Approach For View Extraction

The process descriptions comprise the specification of business functionality in a modeling language, for instance, as we exemplify in this paper, BPEL [7]. Moreover, the process functionality also exposes service interfaces, for instance, expressed in WSDL [23]. To extract appropriate views from process descriptions, i.e., BPEL and WSDL specifications, we developed a number of view interpreters such as *control flow interpreter*, *collaboration view interpreter*, *information view interpreter*, as well as a *BPEL-specific extension view interpreter*.

Our general approach to define view interpreters is based on the Partial Interpreter pattern [25]. This

pattern is typically applied when the relevant information to be interpreted from a language is only a (small) sub-set of the source document's language, and thus, the complexity of the whole language should be avoided in the subsequent interpretation. In particular, in the context of this paper, we concentrate on specific views. The approach based on Partial Interpreter enables us to define modular, pluggable interpreters, and the framework to be easily extensible with new views and view extraction interpreters. The solution is to provide a Partial Interpreter for view extraction, which only understands the specific language elements required for one view. There is a generic parser that is responsible for parsing the process descriptions. The parsing events generated by this generic parser are interpreted by the Partial Interpreters, which only interpret the language elements relevant to a particular view. Hence, the following steps are necessary for defining view extraction interpreters:

1. Define a generic interpreter for parsing the content of the process modeling language (and other relevant languages). In the case of BPEL and WSDL, this is a generic XML parser and a parsing event model, which can be interpreted by the Partial Interpreters.
2. For each view: Define a mapping specification between the elements in the process modeling language elements and the view meta-model elements. That is, the mapping specification contains all elements of a particular view meta-model, and describes how they map to a sub-set of the elements in the process modeling language (and other relevant languages).
3. For each view: Define a view-specific interpretation specification that interprets only the relevant elements for a particular view from the process modeling language. That is, the Partial Interpreter specification explains how a view model can be filled with the information from the process modeling language (and other relevant languages).

The Partial Interpreter's mapping specification and view-specific interpretation specification are both defined generically on basis of the meta-models. Hence, they can be reused for many concrete view models. In the subsequent sections, we present the details of the realization of the mapping specifications for basic process concerns, i.e., control flow interpreter, information view interpreter and collaboration view interpreter to illustrate our general approach. Other view interpreters can be implemented following the same approach.

4 Details of the view-based reverse engineering approach: Four empirical analyses

In this section, we empirically analyze the capabilities of the view-based reverse engineering approach, such as the adaptation of process models to stakeholders' needs and the integration of models in different levels of abstraction, by investigating four typical cases in which the view-based reverse engineering approach can get applied. In doing so, we also introduce the details of our approach for applying it to an exemplary process-driven SOA technology: BPEL and WSDL. BPEL/WSDL is used for exemplification because these are likely the most popular process and service descriptions, which are widely adopted in research and

industry today. Nevertheless, our approach is not limited to BPEL and WSDL technologies but is generally applicable for other process-driven SOA technologies by defining respective meta-models and Partial Interpreter specifications.

In particular, in this section, we investigate the following cases: Because existing business processes often integrate various concerns, it is costly to deal with the existing process code/models by translating them manually into view models. Therefore, the reverse engineering tool chain extracts relevant concerns of the process in term of architectural views using view interpreters. The resulting views can be further integrated to adapt to specific needs and focus of a particular stakeholder. Besides, the tool-chain is able to produce high-level views for business analysts and technology-specific views for technical experts from the same existing process code base. The abstract parts extracted from the process code are integrated into VbMF using the higher-level views, whereas the technology-specific parts are integrated using the lower-level views. Additionally, the views obtained from the reverse engineering tool chain, in turn, can be manipulated using view integration mechanisms [20] to produce richer views which fit better to stakeholders' requirements, or produce a thorough view of the whole process. Moreover, any change on the views can be propagated by means of model transformations, or code generation [20].

These empirical analyses have been carried on an industrial case study, namely, customer care, billing and provisioning systems of an Austrian Internet Service Provider (see [6] for more details). In the following, we use the Billing Renewal process as an example. The billing platform includes a wide variety of services provided by various partners such as financial services, domain services, physical hosting services, retail/wholesale services, and so on. These services are exposed in WSDL interfaces and integrated by using BPEL processes.

4.1 Extracting Relevant Views

BPEL Elements	Control Flow View Elements
bpel::process <i>name</i> ="..."	core::Process/ <i>setName()</i>
bpel::invoke, receive, reply, assign <i>name</i> ="..." <i>createInstance</i> ="yes/no"	cfv::SimpleActivity <i>setName()</i> <i>setInitialActivity()</i>
bpel::sequence <i>name</i> ="..."	cfv::Sequence/ <i>setName()</i>
bpel::flow <i>name</i> ="..."	cfv::Flow/ <i>setName()</i>
bpel::switch <i>name</i> ="..."	cfv::Switch/ <i>setName()</i>
bpel::case <i>name</i> ="..." <i>condition</i> ="..."	cfv::Case/ <i>setName()</i> , <i>setCondition()</i>
bpel::otherwise <i>name</i> ="..."	cfv::Otherwise/ <i>setName()</i>

Figure 6. Mapping the BPEL description onto the Control Flow View

The basic analysis, we performed, was to deal with the extraction of the control flow view from BPEL code. The control flow interpreter, which is based on the Control Flow View meta-model, walks through the process description in BPEL and collects the information of atomic and structured activities. Then, it creates the elements in the Control Flow View and assigns their attributes with relevant values as specified in Figure 6. We demonstrate the mapping of Billing Renewal specification in BPEL onto the Control Flow View in Figure 7.

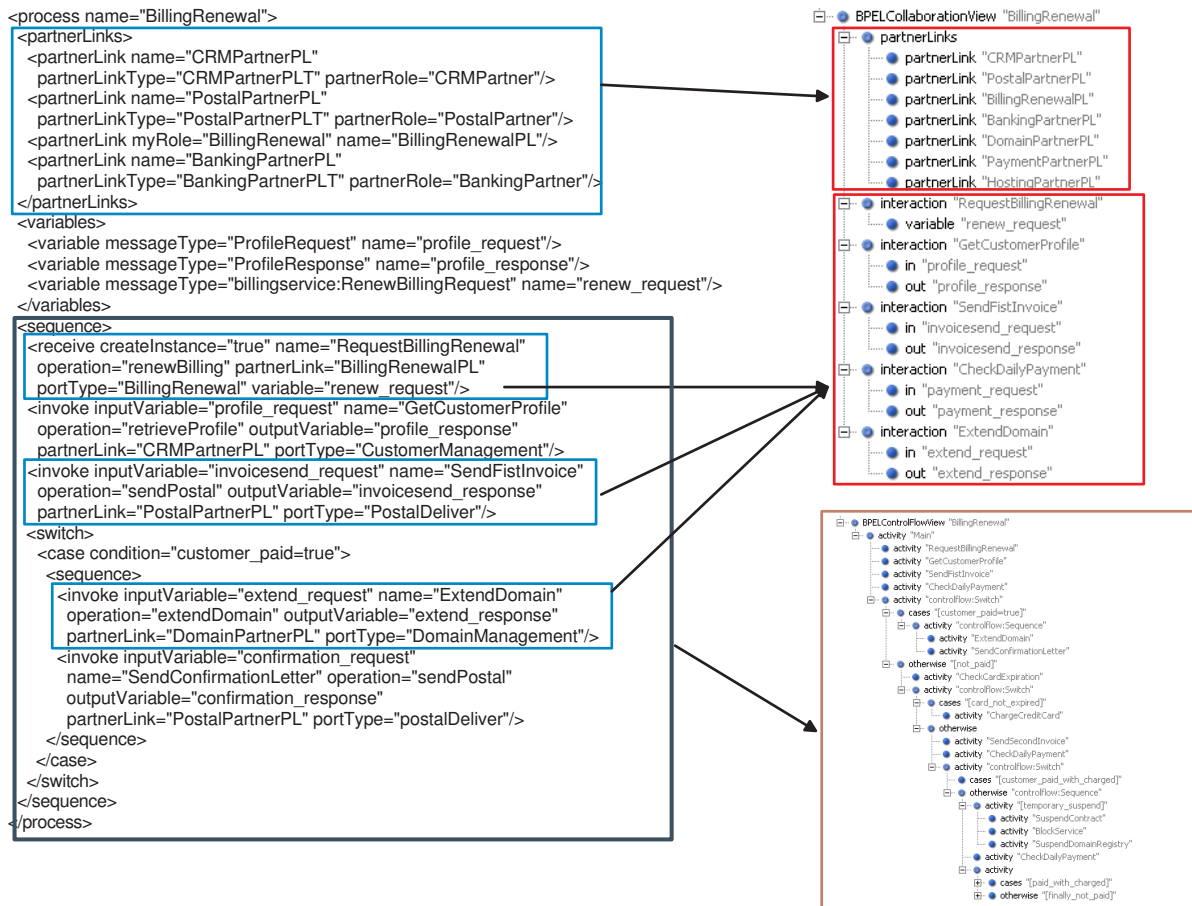


Figure 7. Mapping the Billing Renewal process (left-hand side) onto the VbMF's views including the Collaboration View (top-right) and the Control Flow View (bottom-right)

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()

Figure 8. Mapping the WSDL description onto the Collaboration View and the BPEL extension of the Collaboration View

4.2 Extracting Views at Different Abstraction Levels

To illustrate the ability of adapting views at different levels of abstraction, we devise two interpreters to extract the Collaboration View and the BPEL-specific extension of the Collaboration view. These interpreters are realized using the same approach as used for the control flow interpreter. However, these views comprise not only elements from the BPEL descriptions but also elements of the process interfaces specified in WSDL files. So, first of all, the interpreters collect information from WSDL descriptions and



Figure 9. Mapping the Billing Renewal process (left-hand side) onto the Collaboration View (right-hand side)

create relevant elements on the views using the specifications in Figure 8. Then, the interpreters walk through the BPEL specifications to extract relevant elements in a similar manner as used for in the control flow interpreter. For instance, the *Invoke*, *Receive*, *Reply* activities that are responsible for service invocations will appear on the Collaboration View with the same name as used in the Control Flow View. However, these activities have attributes that are specific for collaboration as depicted in Figure 10. Figure 7 and Figure 9 illustrate the extraction of the Collaboration View from BPEL descriptions of the Billing Renewal process.

The Collaboration View is a high-level representation compared to the BPEL extension of the Collaboration View, which is at a lower level of abstraction. Therefore, the BPEL extension view consists of additional elements and some of these elements have extra properties compared to those of the Collaboration View. This way, other process-driven modeling languages, either high-level or low-level, can be handled and integrated by using the view-based reverse engineering tool-chain and VbMF.

4.3 Enhancing the Adaptability of the Process Models

The adaptability of process models to the requirements of a certain stakeholder can be enhanced using two methods developed in VbMF: extension mechanisms and view integration. View extension mechanisms [20] allow us to enrich existing meta-models with additional elements and/or extra attributes for the existing elements of the original meta-models. This way, the abstract views can be gradually refined into less abstract views by increasing their granularity with added technology-specific features until the resulting views are relevant to a particular stakeholder's needs. Next, we define respective interpreters for these views and use the interpreters to extract the corresponding views from the existing process code. An example of view extension is the BPEL-specific extension of the Collaboration View shown in the previous

BPEL element	Collaboration view element	BPEL Collaboration 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()

Figure 10. Mapping the BPEL description onto the Collaboration View and the BPEL extension of the Collaboration View

analysis.

View integration [20] is another method to produce new richer views by merging existing views. For instance, in [20], we have developed a simple named-matching algorithm and presented an example of integrating the control flow view and the collaboration view. The matching algorithm searches the input views for integration points, which are, in this case, the conformable elements with the same name [20]. Afterward, the two views are merged together at these integration points. The resulting view inherits the control flow that defines the execution order of activities. In addition, the activities in the resulting view that are responsible for invoking services inherit a number of attributes from the corresponding activities defined in the collaboration view. These attributes are necessary to perform service or process invocations.

4.4 Change Propagation

In the previous analyses, the view-based reverse engineering tool chain is used to build views of process models, relevant for particular stakeholders, from existing process code. However, to make this usable in practice, changes on the views must lead to corresponding changes on process code. The propagation of

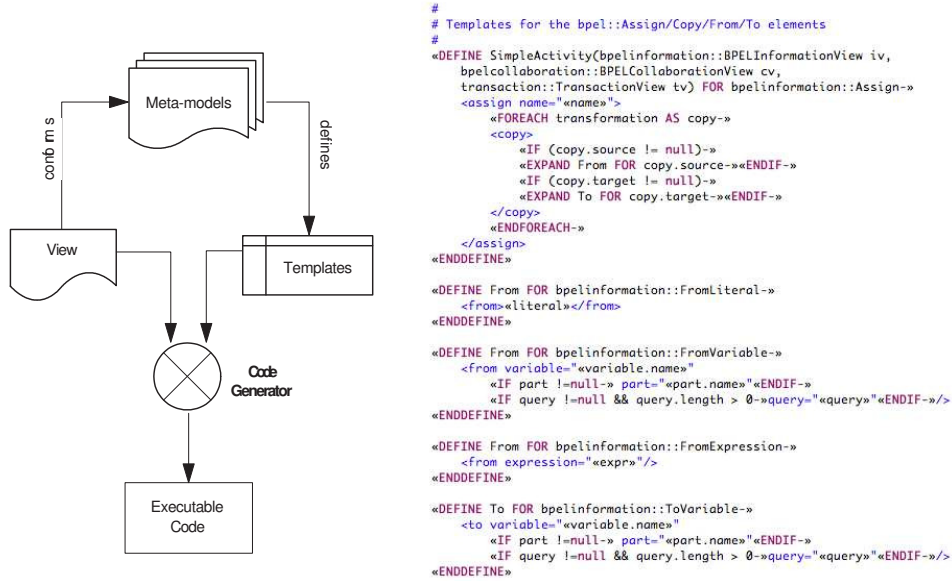


Figure 11. Code generation as a mean of change propagation

change is performed in VbMF using model-to-code transformations (aka code generations). The code generation [20] is realized using the template+meta-model technique [22]. In Figure 11, we present an excerpt of the template for generating BPEL code of the Assign activities. This way, any change in the BPEL extension of the information view or any other view is automatically reflected in the process code.

5 Related Work

Our work presented in this paper focuses on a *reverse engineering approach* [3] based on the concept of architectural views. The whole VbMF tool-chain provides support for *reengineering* [17?] as well. That is, in addition to the reverse engineering parts of the tool chain, means for re-structuring, modification, or forward engineering are provided in order to yield new system structures, or new functionality to meet certain requirement.

In [20], the VbMF approach has been analyzed and compared to other related view-based work on modeling business processes. In the context of reverse engineering, view-based approaches are an emerging area of interest. For instance, the approaches reported in [2, 4, 19] focus on inter-organizational processes (in term of cross-organizational workflows) and use views to separate the abstract process representations (aka public processes) from the internal processes (aka private processes). Bobrik et al. [1] present an approach to process visualization using personalized views and a number of operations to customize the views. Zou et al. [27] propose an approach for extracting business logic, also in term of workflows, from existing e-commerce applications.

All these approaches aim at providing perspectives on business processes at a high level of abstraction and maintaining the relationships among different abstraction levels in order to quickly re-act to changes in business requirements. However, these approaches have in common that only the control flow of process

activities (aka the workflows) is considered. Other process concerns, as for instance service/process interaction, data processing, etc., have only been partially exploited, or even not targeted. In addition, these approaches do not support enhancing process views or propagating changes as provided in our approach, for instance, through view integration, view extension and code generation.

Kazman et al. [9] describe the Dali workbench, an approach for understanding and analysis the system architecture. The extraction process begins with extracting views from source code using some kinds of lexical analyzers, parsers or profilers. Next, the relationships among views are established by view fusion to improve the quality and the correctness of views. However, because of the complexity of typical process models, this approach is hardly applicable to capture the whole process description in a unique view.

In the context of process-driven modeling, there are a number of standard languages in which some provide high-level descriptions, for instance, BPMN [15], EPC [10, 21] and Abstract BPEL in WS-BPEL 2.0 [13]. EPC and BPMN provide high-level diagrams that consist of graphical notations for visualizing representations of processes. These diagrams are relevant to the business analysts. However, there is no explicit link between these languages and the executable languages.

This has led to a number of recent research approaches. For instance, Mendling et al. [12] discuss the transformation of BPEL to EPCs. Ziemann et al. [26] present an approach to model BPEL processes using EPC-based models. Recker et al. [18] translate between BPMN and BPEL. Mendling et al. [11] report on efforts in 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, which describes the execution order of process activities. They offer no support for extension of process models or integrating other concerns of process models, such as service interactions, data processing, transaction handling, etc. Hence, during the transformation from process code to abstract representations, necessary information required to re-generate executable code gets lost.

WS-BPEL 2.0, the newly revised standard, provides the concept of an Abstract BPEL process, which is represented by the same structures as an Executable BPEL process. Developers can explicitly hide some syntactic constructs in an Abstract BPEL process using predefined opaque tokens as explicit placeholders for the omitted details. An abstract process is often associated with a profile which specifies the semantics of the opaque tokens. Hence, one could use an approach akin to our approach where the high-level view is the abstract process profile, and low-level representations are respective profiles. Then our reverse engineering tool-chain could be used to extract the relevant views.

All the above-mentioned approaches and standards have difficulties in handling the complexity of process models: Because the business process integrates numerous concerns, the complexity of process model increases as the number of process elements, such as message exchanges, service invocations, data processing tasks, etc. grows. Hence, these approaches are less efficient than our approach in dealing with pretty huge existing process repositories, developed in other languages or dialects, or integrating arbitrary process modeling tools.

6 Conclusion

The view-based reverse engineering approach, presented in this paper, can help the various stakeholders of a process-driven SOA to overcome two important issues. Firstly, it exploits the concept of architectural view to deal with the complexity of existing process repositories and to adapt the process representations to the stakeholders' needs and focus. Secondly, it provides the ability of integrating diverse process models and offers explicit relationships for understanding and maintaining process models and for propagating changes. Hence, process models at different abstraction levels and different process concerns can be reused to populate the other. This has been achieved by developing a novel concept for a reverse engineering tool chain, based on partial interpreters and view models, and by seamlessly integrating this reverse engineering tool chain into our view-based modeling framework, which also supports means for forward engineering, such as view integration, view extension and code generation. The reverse engineering tool chain enables the reuse of existing process code, e.g. written in BPEL/WSDL, in the view-based modeling framework.

Our work in this paper also introduces some open issues that need be addressed in further research. For instance, other concerns such as event and fault handling, transaction, and so forth, should be examined and integrated into the reverse engineering tool-chain. The tool-chain should be enhanced to cover a broader spectrum of input descriptions in other process-driven languages and technologies.

References

- [1] R. Bobrik, M. Reichert, and T. Bauer. View-Based Process Visualization. In *BPM*, volume 4714/2007, pages 88–95. Springer, 2007.
- [2] I. Chebbi, S. Dustdar, and S. Tata. The view-based approach to dynamic inter-organizational workflow cooperation. *Data Knowl. Eng.*, 56(2):139–173, 2006.
- [3] E. J. Chikofsky and J. H. I. Cross. Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Software*, 7(1):13–17, 1990.
- [4] D. K. W. Chiu, S. C. Cheung, S. Till, K. Karlapalem, Q. Li, and E. Kafeza. Workflow View Driven Cross-Organizational Interoperability in a Web Service Environment. *Inf. Tech. and Management*, 5(3-4):221–250, 2004.
- [5] Eclipse. Eclipse Modeling Framework. <http://www.eclipse.org/emf/>, 2006.
- [6] M. Evenson and B. Schreder. SemBiz Deliverable: D4.1 Use Case Definition and Functional Requirements Analysis. <http://sembiz.org/attach/D4.1.pdf>, August 2007.
- [7] IBM, BEA Systems, Microsoft, SAP AG, and Siebel Systems. Business Process Execution Language for Web Services. <ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf>, May 2003.
- [8] IEEE. Recommended Practice for Architectural Description of Software Intensive Systems. Technical Report IEEE-std-1471-2000, IEEE, 2000.
- [9] R. Kazman and S. J. Carriere. View Extraction and View Fusion in Architectural Understanding. In *ICSR '98: Proc. of the 5th Int. Conference on Software Reuse*, page 290, Washington, DC, USA, 1998. IEEE Computer Society.
- [10] E. Kindler. On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. In *Business Process Management*, pages 82–97, 2004.
- [11] 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.

- [12] 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.
- [13] OASIS. Business Process Execution Language (WSBPEL) 2.0. <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf>, May 2007.
- [14] OMG. Unified Modelling Language™(UML) 2.0. <http://www.omg.org/spec/UML/2.0/>, July 2005.
- [15] OMG. Business Process Modeling Notation. [http://www.bpmn.org/Documents/OMG Final Adopted BPMN 1-0 Spec 06-02-01.pdf](http://www.bpmn.org/Documents/OMG%20Final%20Adopted%20BPMN%201.0%20Spec%2006-02-01.pdf), Feb. 2006.
- [16] openArchitectureWare.org. <http://www.openarchitectureware.org>, Aug. 2002.
- [17] P. Antonini, G. Canfora, and A. Cimitile. Reengineering Legacy Systems to Meet Quality Requirements: An Experience Report. In *ICSM '94: Proceedings of the International Conference on Software Maintenance*, pages 146–153, Washington, DC, USA, 1994. IEEE Computer Society.
- [18] 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.
- [19] K. A. Schulz and M. E. Orłowska. Facilitating cross-organisational workflows with a workflow view approach. *Data Knowl. Eng.*, 51(1):109–147, 2004.
- [20] H. Tran, U. Zdun, and S. Dustdar. View-based and Model-driven Approach for Reducing the Development Complexity in Process-Driven SOA. In W. Abramowicz and L. A. Maciaszek, editors, *International Conference on Business Process and Services Computing (BPSC)*, volume 116 of *LNI*, pages 105–124. GI, 2007.
- [21] W. van der Aalst. On the Verification of Interorganizational Workflows. *Computing Science Reports 97/16*, Eindhoven University of Technology, 1997.
- [22] M. Völter and T. Stahl. *Model-Driven Software Development: Technology, Engineering, Management*. Wiley, 2006.
- [23] W3C. Web Services Description Language 1.1, Mar. 2001.
- [24] WfMC. XML Process Definition Language (XPDL). <http://www.wfmc.org/standards/XPDL.htm>, Apr. 2005.
- [25] U. Zdun. Patterns of Tracing Software Structures and Dependencies. In *Proc. of 8th European Conference on Pattern Languages of Programs (EuroPLoP 2003)*, pages 581–616, Irsee, Germany, jun 2003.
- [26] 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.
- [27] 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.