TECHNISCHE UNIVERSITÄT WIEN

DISSERTATION

View-Based and Model-Driven Approach for Process-Driven, Service-Oriented Architectures

ausgefhrt zum Zwecke der Erlangung des akademischen Grades eines
Doktors der technischen Wissenschaften

unter der Leitung von

Univ.Prof. Mag.rer.soc.oec. Dr.rer.soc.oec. Schahram DUSTDAR
Priv.-Doz. Dr.rer.nat. Uwe ZDUN
E184-1
Institut fr Informationssysteme

eingereicht an der

Technischen Universitt Wien
Fakultt fr Informatik

von

Hoang-Huy TRAN-NGUYEN, Dipl.-Ing.
Matrikelnummer: 0527396
Argentinier Str. 8/184-1
A-1040 Wien, sterreich.

Wien, am Okt 20 2009.

Abstract

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 an approach to business process development by using processes, running in a process engine, to invoke existing processes or services from their process activities (aka process tasks). We call this kind of architecture *process-driven, service-oriented architecture* (SOA).

As the number of elements involved in a business process architecture, for instance, processes, process activities and services, grows, the complexity of process development and maintenance also extremely increases along with the number of the elements' relationships, interactions, and data exchanges – and becomes hardly manageable. This occurs because of two major issues that have not been solved yet in existing approaches for process-driven SOA. On the one hand, the process descriptions comprise various tangled concerns, such as the control flow, data dependencies, service invocations, security, compliance, etc. This entanglement seriously reduces many aspects of software quality such as the *understandability*, *adaptability*, and *maintainability*. On the other hand, the differences of language syntaxes and semantics, the difference of granularity at different abstraction levels, and the lack of explicit links between process design and implementation languages hinder the *interoperability*, *reusability*, *understandability*, and *traceability* of software components or systems being built upon or relying on such languages.

This dissertation presents a novel approach for addressing the aforementioned challenges. Our approach exploits a combination of the concept of architectural views – a realization of the *separation of concerns* principle – and the model-driven development paradigm – a realization of the *separation of abstraction levels* – to achieve the following major contributions: *first*, it captures different perspectives of a business process model in separated, (semi-)formalized view models in order to adapt to various stakeholders' interests; *second*, it separates different abstraction levels in a business process architecture; *third*, it provides a seamless, extensible integration solutions for improving interoperability and reusability of process descriptions; *finally*, it reduces the complexity of dependency management and enhances traceability in process development.

As a proof-of-concept, the aforementioned concepts has been realized in the View-based Modeling Framework, which is an extensible development framework for process-driven SOAs. This approach has been evaluated on a number of research- and industry-based case studies. In addition, a number of qualitative comparisons have been conducted to assess our approach and compare it with existing related approaches for process-driven SOA development.

Kurzfassung

Service-oriented Computing ist ein aufkommendes Paradigma, das eine wichtige Vernderung von traditioneller, stark gekoppelter und schwer zu adaptierender Software-Entwicklung hin zu plattformneutraler und lose gekoppelter Software-Entwicklung mit sich bringt. Services sind interoperabel und plattformunabhngig und untersttzen daher einen Ansatz zur Entwicklung von Geschftsprozessen, welche in einer Prozess-Engine ablaufen um existierende Prozesse und Services aus den Prozessaktivitten heraus aufzurufen. Wir nennen diese Art der Architektur prozessgetriebene, service-orientierte Architectur (SOA).

Wenn die Anzahl der Elemente in der Geschftsprozessarchitektur, wie Prozesse, Prozessaktivitten und Services, ansteigt, nimmt auch die Komplexitt der Entwicklung und Wartung von Prozessen zusammen mit der Anzahl der Beziehungen, Interaktionen und dem Austausch von Daten dieser Elemente zu – und wird schnell schwer handzuhaben. Das geschieht, weil die Anstze fr prozessgetriebene SOA bislang zwei Hauptprobleme noch nicht gelst haben. Zum einen umfassen die Prozessbeschreibungen diverse, untereinander verwobene Belange, wie Kontrollfluss, Datenabhngigkeiten, Service-Aufrufe, Sicherheit, Compliance, etc. Diese Verwobenheit hat einen negativen Einfluss auf die Software-Qualitt, wie Verstehbarkeit, Adaptierbarkeit und Wartbarkeit. Zum anderen verhindern die Unterschiede der Sprachsyntaxen und -semantiken, die Unterschiede in der Granularitt auf verschiedenen Abstraktionsebenen und das Fehlen expliziter Bindeglieder zwischen Prozessentwurfssprachen und Prozessimplementierungssprachen die Interoperabilitt, Wiederverwendbarkeit, Verstehbarkeit und Nachvollziehbarkeit von Software-Komponenten oder Systemen, die mit diesen Sprachen erstellt werden oder von ihnen abhngig sind.

Die vorliegende Dissertation prsentiert einen neuen Ansatz, um diese Herausforderungen zu meistern. Dieser Ansatz nutzt eine Kombination von Architektursichten – als eine Realisierung des Separation-of-Concern-Prinzips – und von modellgetriebener Entwicklung aus, um die folgenden Hauptbeitrge zu leisten. Erstens werden verschiedene Perspektiven eines Geschftsprozessmodells in separaten, (semi-)formalen Sichtenmodellen festgehalten, um sich den Interessen verschiedener Beteiligter am Prozess anzupassen. Zweitens werden verschiedene Abstraktionsebenen in einer Geschftsprozessarchitektur separiert. Drittens wird eine bergangslose und erweiterbare Integrationslsung zur Verbesserung von Interoperabilitt und Wiederverwendbarkeit prsentiert. Zu guter Letzt wird die Komplexitt des Abhngigkeitsmanagement reduziert und die Nachvollziehbarkeit in der Prozessentwicklung erhht.

Als Realisierung der Konzepte wird das View-based Modeling Framework, das ein erweiterbares Rahmenwerk fr process-driven SOAs darstellt, vorgestellt. Der Ansatz wurde in einer Reihe von Forschungs- und Industriefallstudien evaluiert. Zustzlich wurden verschiedene qualitative Vergleiche angestellt, um den Ansatz zu bewerten und ihn mit anderen Arbeiten zu vergleichen.

Acknowledgements

This work would never have been made possible without the significant support of my mentor, Professor Schahram Dustdar. He gave me a great chance to work in his group – a hot spot of inspiration, enthusiasm, and innovation. He has safely and skillfully guided me through the entirety of the work presented in this dissertation. He has strongly and professionally influenced my way of thinking and working.

I am in great debt to Uwe Zdun for your kind advice, guidance, enthusiasm, and tireless support. Saying thank you is definitely not enough. It is a big pleasure for me to work with you to foster various insightful ideas, especially on the notion of architectural views and the model-driven paradigm. I also would like to thank my colleagues at the Distributed Systems Group, past and present, for many fruitful discussions, comments, and supports, especially thanks Ta'id Holmes for his important contributions on the modeling of human interactions. Also thanks to my colleagues at the Faculty of Computer Science and Engineering, HCMC University of Technology, Vietnam, for your support, especially thanks to Dr. Nguyen Thanh Son and Dr. Thoai Nam for your kind recommendation, advice and encouragement.

I am grateful to anonymous reviewers for numerous critical comments and insights that extremely helpful for building up this work as well as my professional career.

I am truly grateful to my beloved wife, Minh Xuân, for your deep love, endless patience, and graceful encouragement. Thank you, my dear, for loving me, taking care of my busy life, and being with me in every of most difficult moments.

This work is one of grateful presents that I would like to dedicate to my parents, parents in-law, brothers and sisters. I do always engrave your great love, sympathy, patience, support, and encouragement in my heart.

Last but, of course, not least, thanks to many friends of mine for helping a lot in setting up and enlightening my life in Vienna.

This work was partially supported by the *Technologiestipendien Südostasien Doktorat* by *Austrian Federal Ministry for Science and Research (Bundesministerium fr Wissenschaft und Forschung – BMWF)* and the *European Union FP7 STREP project COMPAS, grant no. 215175 (*`http://www.compas-ict.eu`*).*

Dedicated to my wife and family.
I love you so much.

Contents

List of Figures

List of Tables

List of Acronyms

BPEL	Business Process Execution Language
BPDM	Business Process Definition Metamodel
BPML	Business Process Modeling Language
BPMN	Business Process Modeling Notation
CORBA	Common Object Requesting Broker Architecture
DSL	Domain-Specific Language
EMF	Eclipse Modeling Framework
EPC	Event-Driven Process Chain
FDL	WebSphere© FlowMark© Definition Language
IDEF3	Integrated DEFinition for Process Description Capture Method
Java EE	Java Platform Enterprise Edition
jPDL	jBPM Process Definition Language
MDA	Model-Driven Architecture
MDD	Model-Driven Development
MDE	Model-Driven Engineering
MDSD	Model-Driven Software Development
MOF	Meta-Object Facility
OCL	Object Constraint Language
PIM	Platform-Independent Model
PDM	Platform Definition Model
POJO	Plain Old Java Object
PSM	Platform-Specific Model
RBAC	Role-Based Access Control
REST	REpresentational State Transfer
RM-ODP	Reference Model of Open Distributed Processing
RPC	Remote Procedure Call
SOA	Service-Oriented Architecture
SOC	Service-Oriented Computing
UML	Unified Modeling Language
URI	Uniform Resource Identifier
VbMF	View-based Modeling Framework
VbTrace	View-based Traceability Framework
WCF	Windows Communication Foundation
WS-BPEL	Web Services Business Process Execution Language
WS-CDL	Web Services Choreography Description Language
WSDL	Web Services Description Language
XPDL	XML Process Definition Language
YAWL	Yet Another Workflow Language

Chapter 1

Introduction

“*Everything is vague to a degree you do not realize till you have tried to make it precise* (“The Philosophy of Logical Atomism”).”

— *Bertrand Russell (1872-1970)*

1.1 Problem statement

Service-oriented architecture (SOA) is a popular architectural style for developing distributed systems and software. SOAs have made an important shift from traditional tightly coupled, hard-to-adapt software development to more platform neutral, loosely coupled software development. In SOAs, systems and software functionalities are exposed in terms of services. Each service has a standard interface and is made accessible over a network. Services communicate with each other by exchanging messages[5,41,126].

The interoperable and platform independent nature of services supports a novel approach, namely, *process-driven, service-oriented architecture*[59]. In process-driven SOAs, systems and software are described in terms of processes using high-level, notational modeling languages and implemented in executable languages. A typical process comprises a number of activities, the control flow, and the process data. Each activity corresponds 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 accomplish particular business goals.

Leveraging the underlying service-oriented architectures, process-driven SOAs provide an efficient development paradigm that supports decoupling the business logics from the specific platform technologies and implementation languages. In addition, process-driven SOAs aim at supporting business agility, i.e., to enable a quicker reaction on business changes in the IT by manipulating high level- process descriptions instead of code. The increasing attention of both research and industry in process-driven SOAs has led to a number of encouraging results, for instance, the fostering of several standards, methodologies, and techniques, such as IDEF3[96], ARIS/EPC[34], WS-CDL[173], BPMN[121], BPML[16], BPEL[67,109], XPDL[181], and UML Activity Diagram extensions[116] for modeling and developing processes.

An important problem process-driven developers face today is the increasing size and complexity of process descriptions. This occurs because the process descriptions integrate various tangled concerns: control flows, data processing and dependencies, service invocations, fault and transaction handling, event handling, and human interactions, etc. As such, the complexity of developing and maintaining the processes increases along with the number of invocations and data exchanges and quickly become hardly manageable as the number of services or processes involved in a process grows. This complexity resides in different phases of process-driven development ranging from design to implementation, deployment, and maintenance.

In addition, this problem also appears at different abstraction levels[59] due to not only the differences of syntax and semantics but also the differences of granularity. Processes are usually specified using highly abstract and notational elements, such as the elements in IDEF3, BPMN, EPC, or UML Activity Diagrams. As business experts are rather familiar with domain knowledge and concepts than technology-specific ones, these abstract, notational elements are suitable to capture desired business functionality, but they are not executable. Therefore, IT experts necessarily need to be involved in development of the processes by transforming these abstract design elements into executable descriptions. For example, IT experts can translate the high-level process model into BPEL, specify the processes interfaces and involved services in terms of WSDL[170] and XML schema[177], and/or implement some business logics using particular technologies, such as Web Services[171], RESTful Web Services[47,128], and Java EE[151]. Additional deployment configurations (e.g., process descriptors) might need to be defined in order to successfully deploy and execute the implemented processes. Processes expressed in execution languages are often more concrete, technology-specific, and of finer granularity than the design counterparts. Additionally, there are a number of implementation artifacts that are not present in process designs, such as process deployment configurations, service bindings, service implementation endpoints, XML schema definitions, etc.

To the best of our knowledge, the complexity of process descriptions and the discordance between different levels of abstraction, as described in the previous paragraph, have not been thoroughly addressed in the literature. As a consequence, these problems impair various quality properties in process-driven SOA development, maintenance, and evolution, such as the understandability, adaptability, interoperability, reusability, and traceability.

Understandability and adaptability Business process developers have to deal with constant 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. Most of the existing business processes are developed and maintained by IT experts in low-level, executable languages. Process designs, if they exist, are often done in highly abstract and notational languages. Unfortunately, there are no explicit links between process design and implementation languages. As a consequence, 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, IT experts are required for many tasks in managing, developing, and maintaining the process models. For each change, regarding both business and technology related concerns, the IT experts have to investigate, analyze, and modify a number of executable code fragments and/or related process models that is cumbersome and error-prone. In addition, there is a lack of the separation of process concerns as well as the adaptation of process representations to the needs, knowledge, and experience of a particular stakeholder. Thus, in order to thoroughly understand or analyze a certain concept of either a process design or an implementation, a certain stakeholders has to go across numerous dependencies between various concerns, some of which are even not suitable for the developer's expertise and skills.

Interoperability and reusability Business experts often design processes in high-level abstraction languages, such as BPMN, EPC, or UML Activity Diagram, and IT experts implement them using executable languages, such as BPEL/WSDL. An important issue that hinders the interoperability of existing process representations is the huge divergence of these languages. The difference of language syntax and semantics as well as the difference of granularity hinder the smooth integration of the various fragments in process descriptions described in different languages, and therefore, reduce the interoperability of those languages. In addition to the aforementioned divergence of languages, the entanglement of different process concerns also suppresses the reusability of process models. In order to reuse a certain excerpt of a process model, the stakeholder has to go across various concerns. Existing process languages and tools have not properly supported cross referencing between process languages, for instance, between UML Activity Diagram and EPC and BPMN and BPEL, or between two BPEL descriptions and so on. As a consequence, the reusability of process descriptions, either in high-level or low-level languages, are merely achieved by using the "copy-and-paste" approach, which is very tedious and error-prone.

Traceability Understanding trace dependencies between process design and implementation is vital for change impact analysis, change propagation, documentation, and many other activities[148]. Unfortunately, artifacts created during the process development life cycle likely end up being disconnected from each other. This impairs the traceability. First of all, there are no explicit links between process design and implementation languages not only due to the differences of syntax and semantics but also the differences of granularity. The second factor is the complexity caused by tangled process concerns that multiplies the difficulty of analyzing and understanding the trace dependencies. Finally, there is a lack of adequate tool support for establishing and maintaining the trace dependencies between process designs and implementations.

1.2 Scientific contributions

One of the successful approaches to manage complexity is *separation of concerns* [35,53,152]. Process-driven SOAs use a specific realization of this principle, *modularization*[53]: 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 stakeholders' 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 stakeholder. In the field of software architecture, *architectural views* (or view for short) have been proposed as a solution to this problem. A *view* is a representation of a system from the perspective of a related set of *concerns*[69]. The notion of views 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 notion of 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. We then formalize the notion of view using view models. That is, the structure and semantics of each view are specified by a corresponding view model. In particular, our approach offers separated view models, such as the collaboration, information, control flow, event, transaction, and human view models, each of which represents a specific part of the business process. The view models can be viewed and manipulated separately to get a better understanding of a particular concern, or they can be integrated to produce a richer view or a more thorough view of the processes and services. This way, we use view models – the formalizations of process views – to separate and manage the various tangled concerns of business processes in a flexible and extensible manner.

As the notion of views aims to reduce the complexity of business processes, the challenging gap between different abstraction levels, mentioned in the previous section, is still unsolved. We support stakeholders in bridging this gap by using a model-driven stack that is a specific realization of the model-driven development (MDD) paradigm[57,150] . This model-driven stack separates view models into abstract and technology-specific layers. The abstract view models aim at providing problem-oriented concepts and structures by which business experts can better understand, analyze, and manipulate. Platform-specific and technical details are excluded from these abstract views and only provided in the technology-specific views that are relevant to the IT experts. Then, we develop a Core model as the basis for deriving other view models along with various mechanisms, such as view extension/refinement, view integration, code generation, and reverse engineering. This way, we support stakeholders in (semi)-automatically creating and maintaining the dependency relationships between view models at the same or different abstraction levels.

Numerous existing process-oriented systems have been built upon executable process languages such as BPEL. Process's functionality are exposed in service description languages such as WSDL. Unfortunately, there is no explicit link between the process design languages (e.g., EPC, BPMN, and UML Activity Diagram) and the implementation counterparts (e.g, BPEL, jPDL, and BPML) such that business objectives comprised in the process designs are unexpectedly disconnected from the implementations. Hence, the business experts, who understand the business objectives best, hardly analyze and modify the "as-is" business processes because they are confronted with several tangled technical details in the process implementations. On the other hand, stakeholders hardly check which parts of the process implementations are accordant with certain parts of the process design and vice versa, hardly estimate the ripple-effect of a certain part of the process designs or implementations, and so on. Existing approaches to business process development have not considered and addressed these issues thoroughly. Our solutions for these problems are twofold. We firstly devise a view-based reverse engineering approach in order to extract view models from existing process descriptions. The resulting view models are either abstract or technology-specific, and they can be tailored or adapted to stakeholders' needs and knowledge. This way, our reverse engineering approach helps stakeholders to get involved in process re-development and maintenance at different abstraction levels. Secondly, we propose a view-based traceability approach as an orthogonal dimension of our model-driven stack in order to support stakeholders in (semi-)automatically creating and maintaining the dependency relationships between various process development artifacts. Our view-based traceability approach not only captures the relationships among elements of VbMF, but also helps reconciling the dependency gap between process designs (e.g., BPMN) and implementations (e.g., BPEL) by using VbMF as an intermediate layer.

In summary, this research contributes a novel approach that addresses two major challenging issues in process-driven SOAs: (1) managing the complexity of the process development and (2) bridging the gap between different levels of abstraction from process design to implementation. In particular, our approach contributes various valuable aspects to process development, including:

- ***Reducing the complexity of process development***: Exploiting the notion of views, our approach offers the stakeholders a number of tailored perspectives that are much more relevant and appropriate to their needs and knowledge. A certain stakeholder can work on a specific view, a combination of related views, or a thorough view of the business process according to his distinct knowledge and skills. As such, stakeholders may focus solely on their concerns of interest, and therefore, better contribute their specific expertise to process development. In other words, our approach aids the stakeholders mastering the *horizontal dimension*, i.e., the dimension of different process concerns, of process-driven SOAs in a more flexible manner.
- ***Enhancing adaptability and automation*** : We devise a model-driven stack in which views are separated into different abstraction levels ranging from abstract, high-level view models to

technology-specific views. One the one hand, our approach offers stakeholders the ability of working with suitable levels of abstraction, especially, with concerns of interest, as described in previous paragraph, according to their specific knowledge and skills. One the other hand, this approach helps the stakeholder mastering the *vertical dimension*, i.e., the dimension of abstraction levels, by developing modeling mechanisms, such as view extensions and integration, to bridge the gap between these abstraction levels. This bridge is expanded further as we develop model-to-code transformation mechanisms for automatically generating executable code and runtime configurations from the technology-specific view models.

- ***Enhancing interoperability and reusability*** : Interoperability and reusability suffer from the heterogeneous nature of the participants of a software system. SOA has partially reconciled 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 harnesses the *notion of views* and the *Partial Interpreter* pattern[184] in order to adapt process models to suit knowledge and skills of various stakeholders. Using the Partial Interpreter pattern, we devise a number of view-based interpreters to extract more or less abstract view models from process descriptions. This way, different kinds of process modeling languages can be analyzed to build up the relevant representations (or view models). Next, using mechanisms such as the extension mechanisms, view integration, and code generation mentioned in the previous paragraph, these views can be manipulated to produce more appropriate representations according to stakeholders' requirements, or to re-generate code in executable languages. This way, our approach 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 from existing process models, e.g. written in BPEL.

- ***Enhancing dependency management and traceability*** : An adequate dependency management is crucial in process development because it strongly supports tracing development artifacts, change impact analysis, managing process evolution, etc. We devise a view-based, model-driven traceability approach that adds a vertical dimension to the aforementioned model-driven stack. In our traceability approach, a traceability meta-model is devised for representing numerous kinds of traceability meta-data acquired during the process development life cycle. As such, the traceability approach acts as an intermediate bridge that offers stakeholders the ability of leveraging the traceability meta-data for tracing development artifacts, analyzing the impact of a certain change in process models, or managing the evolution of the process development.

We also prototypically develop a proof-of-concept framework for process-driven SOA development.

The framework consists of a forward engineering and a reverse engineering toolchain . In the forward engineering toolchain, we offer various tailored and adapted views that are more relevant to the knowledge and needs of particular stakeholders, e.g. business analysts or IT experts. The code generation process is driven by model transformations from view models or integrated models into executable code. On the other side, stakeholders can use the reverse engineering toolchain for integration of legacy business process descriptions . These descriptions are represented in terms of appropriate view models that can be manipulated or re-used to develop other processes than those developed in the forward engineering toolchain.

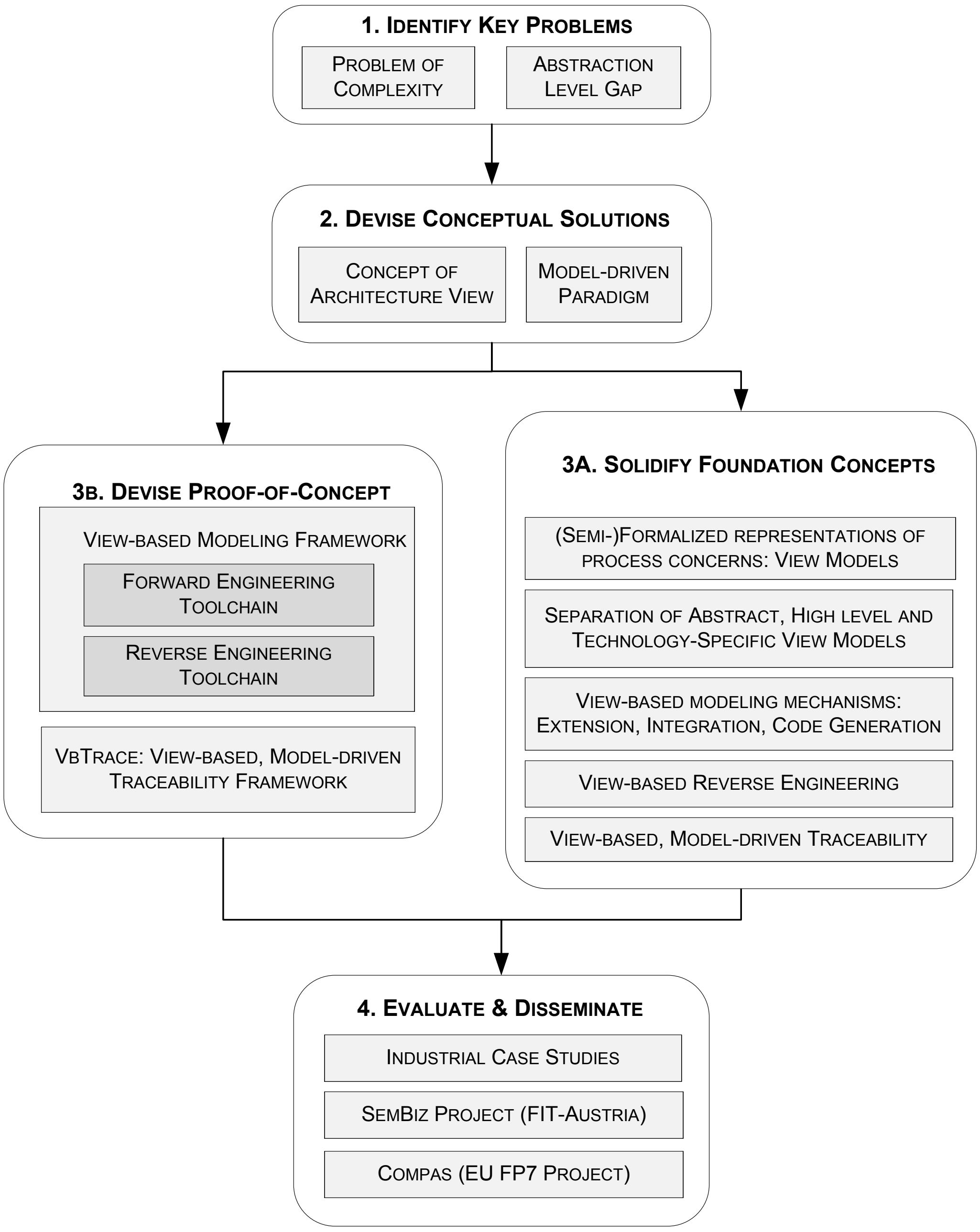

1. Identify Key Problems
Problem of Complexity
Abstraction Level Gap
2. Devise Conceptual Solutions
Concept of Architecture View
Model-driven Paradigm
3B. Devise Proof-of-Concept
View-based Modeling Framework
Forward Engineering Toolchain
Reverse Engineering Toolchain
VbTrace: View-based, Model-driven Traceability Framework
3A. Solidify Foundation Concepts
(Semi-)Formalized representations of process concerns: View Models
Separation of Abstract, High level and Technology-Specific View Models
View-based modeling mechanisms: Extension, Integration, Code Generation
View-based Reverse Engineering
View-based, Model-driven Traceability
4. Evaluate & Disseminate
Industrial Case Studies
SemBiz Project (FIT-Austria)
Compas (EU FP7 Project)

Figure 1.1: Research methodology

1.3 Research methodology

To achieve the aforementioned contributions, we have conducted the research tasks illustrated in Figure 1.1.

Milestone 1: First of all, we studied the state-of-the-art of languages, standards, technologies as well as scientific approaches in the field of process-driven SOAs and identified two major problems, explained in Section 1.1, that have not been addressed in the literature yet.

Milestone 2: We came up with our solutions for these problems that resulted in a foundational publication[155]. This work introduces the conceptual vision of our approach based on the notion of views and the model-driven development paradigm.

Milestone 3A: Later, the above-mentioned concepts are solidified via a number of subsequent tasks of which the most important ones are the (semi-)formalizations of process views in form of view models and the separation of abstraction levels via the model-driven stack. These tasks are complemented by the various mechanisms such as extension, view integration, code generation to make our approach flexible and extensible to many other concerns[62,97,154,155]. In addition, we designed a view-based reverse engineering approach that is able to interpret existing process descriptions and extract view models that are relevant to the stakeholders needs and knowledge[156,157]. Last but not least, we proposed a view-based, model-driven traceability approach that supports stakeholders in establishing and managing the dependencies of development artifacts during the process life cycle.

Milestone 3B: In parallel, we have implemented a view-based modeling framework (VbMF), including the forward engineering and reverse engineering toolchains, and a view-based traceability framework (VbTrace) as proof-of-concept prototype of our approach[158].

Milestone 4: The aforementioned proof-of-concept prototypes are also used for evaluating our approach via two industrial case studies as well as the comparison with the related work[155–158,187]. Moreover, our view-based, model-driven approach also contributed an important modeling and integration foundation to two scientific projects: SemBiz[49] and COMPAS[42].

The outcomes of these scientific tasks include a number of publications at international conferences and workshops, which are:

1. **Huy Tran**, Uwe Zdun, and Schahram Dustdar. *View-based and Model-driven Approach for Reducing the Development Complexity in Process-Driven SOA*. In First International Working Conference on Business Process and Services Computing (BPSC), pp. 105–124, LNI, 2007.

2. **Huy Tran**, Uwe Zdun, and Schahram Dustdar. *View-based Integration of Process-driven SOA Models At Various Abstraction Levels*. In First International Workshop on Model-Based Software and Data Integration (MBSDI), pp. 55–66, Springer, 2008.
3. **Huy Tran**, Uwe Zdun, and Schahram Dustdar. *View-Based Reverse Engineering Approach for Enhancing Model Interoperability and Reusability in Process-Driven SOAs*. In 10th International Conference Software Reuse, LNCS 5030, pp. 233–244, Springer, 2008.
4. Ta'id Holmes, **Huy Tran**, Uwe Zdun, and Schahram Dustdar. *Modeling Human Aspects of Business Processes - A View-Based, Model-Driven Approach*. In 4th European Conference on Model Driven Architecture Foundations and Applications (ECMDA-FA), LNCS 5095, pp. 246–261, Springer, 2008.
5. **Huy Tran**, Ta'id Holmes, Uwe Zdun, and Schahram Dustdar. *Chapter 2: Modeling Process-Driven SOAs - a View-Based Approach*. Eds. J. Cardoso and W. van der Aalst, Handbook of Research on Business Process Modeling, IGI Global, 2009.
6. **Huy Tran**, Uwe Zdun, and Schahram Dustdar. *VbTrace: Using View-based and Model-driven Development to Support Traceability in Process-driven SOAs*, Journal of Software and Systems Modeling, DOI: 10.1007/s10270-009-0137-0, Nov 2009.
7. Uwe Zdun, **Huy Tran**, Ta'id Holmes, Ernst Oberortner, Emmanuel Mulo, and Schahram Dustdar. *Compliance in Service-oriented Architectures*, Information Systems Journal, Wiley, 2009 (submitted).

1.4 Dissertation structure

The rest of this dissertation is organized as following.

Chapter 2 introduces the background literature, including the basic concepts as well as the state of the art in service-oriented computing (SOC) and process-driven, service-oriented architectures (SOAs), model-driven development (MDD), and architectural views.

The major contributions are covered by three following chapters:

Chapter 3 presents the view-based, model-driven approach that exploits the notion of views and leveraging the MDD paradigm to address existing problems in process-driven SOA development. This chapter contributes a forward engineering toolchain for process development.

Chapter 4 is about the view-based reverse engineering approach for interpreting and extracting view models at high or low levels of abstraction from existing process descriptions. This chapter shapes a reverse engineering toolchain for process-driven SOA development.

Chapter 5 contributes an additional dimension to the realization of the View-based Modeling Framework presented in Chapter 3 that supports stakeholders in (semi-)automatically establishing and maintaining the trace dependencies between various development artifacts.

The evaluation and summarization of the dissertation come in the last chapters, including:

Chapter 6 presents the illustration of our approach via a scenario-driven approach that examines a number of industrial case studies. We also conduct a qualitative analysis of our approach with respect to important quality properties such as the complexity, reusability, and separation of concerns.

Chapter 7 summarizes our major contributions, discuss open problems, and broadens our visions with some future works.

Chapter 2

State of the Art

2.1 Introduction

Our main contribution in this dissertation aims to address existing problems in the field of process-driven, service-oriented architectures (SOA) using a view-based, model-driven approach. In this approach, we consolidate the *separation of concerns* principle – realized in terms of the notion of architectural views – and the *separation of abstraction levels* – realized in terms of the model-driven paradigm – to support stakeholders in dealing with the complexity of process-driven SOA development in a flexible, adaptable, and (semi-)automatic manner with respect to particular needs, knowledge, and experience of each stakeholder. Therefore, the concepts and contributions presented in our approach are relevant to different scientific disciplines, including process-driven SOA development, model-driven development, software reuse, and traceability. In this chapter, we briefly introduce basic concepts and the state of the art in these scientific disciplines.

2.2 Service-oriented architectures

Service-oriented computing (SOC) is an emerging cross-disciplinary paradigm for distributed computing that uses services to support the development of interoperable, evolvable, and massively distributed application[127]. Services are autonomous, platform-independent entities that can be described, published, discovered, and loosely coupled by using standard protocols[37,127]. Service-oriented Architecture (SOA) is the main architectural style for SOC.

There are a number of different perspectives into SOC/SOA. For instance, a business expert envisions a set of services that a business wants to expose to their customers and partners, or other portions of the organization[15]. A system architect envisages an architectural style consisting of service providers, consumers, and service descriptions as well as a set of architectural principles and patterns that address characteristics such as *modularity*, *encapsulation*, *separation of concerns reusability*, *composability*, and *interoprability*[15]. A developer considers SOA *a programming model comprising standards, tools, and technologies such as Web Services*[15].

The popularity of SOC/SOA is confirmed by industry observers and market research organizations: Research and Markets reports SOA markets at $450 million in 2005 growing to $1.6 billion in 2006, $2 billion in 2007, and only slightly growing to $3.3 billion in 2008 due to global economy crisis.

The SOA markers are anticipated to grow at an average rate of 8% initially reaching 95% per year by 2015 for $16.75 billion[136–139]. CA Wily recently released the results of a survey that covered 615 companies in the process of SOA-based efforts[21]. In this survey, 73% say their organizations have either deployed an SOA application whilst 27% are planning an SOA deployment. In addition, 92% say their SOA initiatives met or exceeded business unit objectives, while only 8% say they did not[21]. These numbers all point to exponential growth and enormous popularity of SOA and SOC in a global perspective.

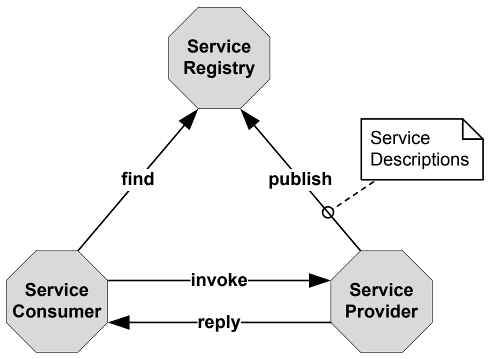


Figure 2.1: A typical service-oriented architecture (adapted from[171])

Services are often made accessible via network systems and can be invoked by means of public, standard interfaces. The communication between services can involve either simple data exchanges or complex coordinations of activities of two or more services. A service interface is a platform- and protocol-independent specification that describes what functionality is provided by the service and how to access this functionality[5,41,126]. This way, SOA has a potential to deliver several benefits, such as supporting interoperability to reconcile the heterogeneity nature of software and systems, integrating legacy systems and applications, supporting business agility by decoupling service interfaces from implementation details of business functionality, and so on[5,41,126].

Two essential roles in a typical SOA are service provider and service consumer (or so-called service client), see Figure 2.1. A service provider offers functionalities of software and systems to its clients via public interfaces whilst a service consumer utilizes these functionalities by sending appropriate messages to the corresponding service endpoints specified in the service interfaces. Service descriptions, including service interfaces, providers' information, etc., might be stored in a service registry where the clients can query and lookup for a particular service, functionality, or provider[5,41,126]. There is a wide range of technologies that can be used to implement SOAs, such as Common Object Request Broker Architecture (CORBA)[114], Java EE[131], Representational State Transfer (REST)[47], Remote procedure call (RPC)[70], Web Services[171], and Windows Communication Foundation (WCF)[104], to name but a few. Web Services and REST are currently the most prominent SOA-based technology[128,171].

Definition 2.1 (W3C, Web Service)**.** *A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-*

processable format (specifically WSDL). Other systems interact with the Web Service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards[172].

REST is an architectural style for designing distributed systems and applications[47]. The REST architectural style is based on the four principles:

- *Identification*: Resources are identified by URIs that provided a global addressing space for resource and service discovery.
- *Manipulation*: Resources are manipulated through a uniform interface that is a fixed set of create, read, update, and delete operations: PUT, GET, POST, and DELETE.
- *Self-descriptive messages*: Messages include enough meta-data to describe how to process the message.
- *Stateless communication*: The communication is constrained by no context being stored.

The biggest advantage of Web Services and REST over its competitors is the use of Internet as the communication medium[127]. RESTful Web Services are services implemented using HTTP and the principles of REST. As a consequence, such RESTful services can be considered as a collection of resources[128]. Web Services utilizes several technologies and (de facto) standards[5,126,180], such as Web Services Description Language (WSDL)[170,175] for defining service interfaces, Simple Object Access Protocol (SOAP)[169,174] for delivering XML messages between services over existing network infrastructures, Universal Description Discovery and Integration (UDDI)[108] for publishing and discovering services, WS-Policy[176] for specifying policy requirements and constraints, and so on.

2.3 Process-driven, service-oriented architectures

A promising vision of SOC/SOA is to better support the integration of business functionality in terms of loosely coupled services that span across organizations and computing platforms to either accomplish a certain business goal or quickly react to the changes of business environment and technology[127].

A process-driven SOA[59] extends SOAs with a process composition layer, and therefore, benefits from those SOA's advantages. Particularly, process-driven SOAs delivers a development paradigm for systematically and efficiently accomplishing business functionality by using processes running in a process engine to invoke existing business services from process activities (see Figure 2.2). In process-driven SOAs, the notion of process is central. A process consists of many activities each of which may perform a service invocation, human interaction, or data processing task. The process control flow specifies the execution order of these activities to achieve a specific business

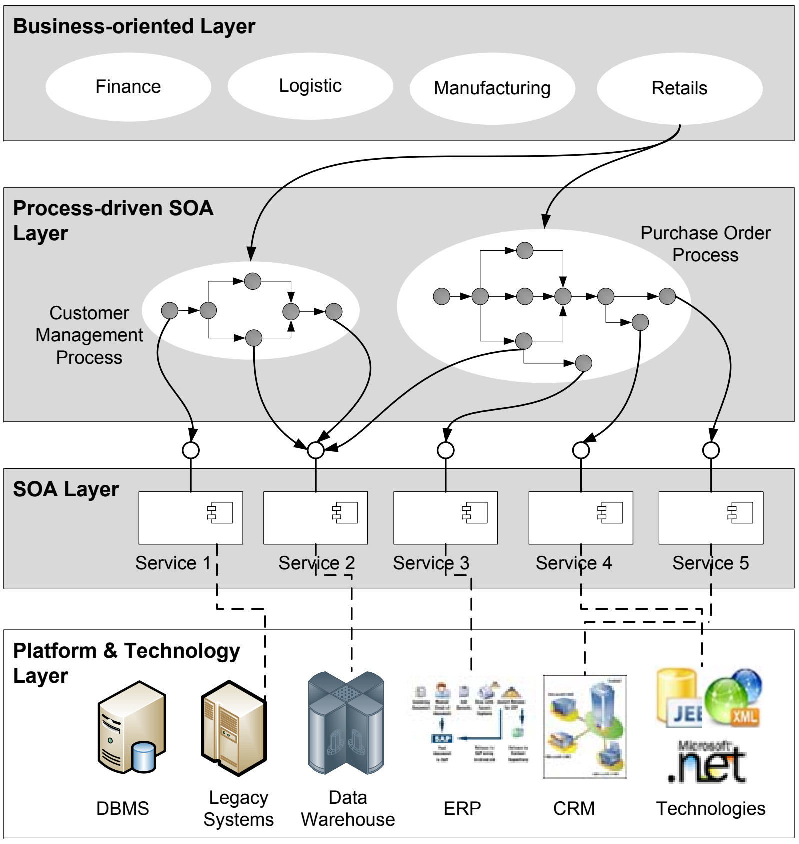


Figure 2.2: Layered view of SOA and process-driven SOA

goal. Processes are often deployed in a process engine for executing and coordinating functionality provided by the other services or processes. Moreover, a process itself can also be exposed to its users as a service with standard interfaces, and therefore, can be invoked by the users, services, or other processes.

Process-driven SOAs aim at enhancing the productivity and flexibility via process management. In this approach, the high-level business processes are aligned with the IT infrastructure. Organizations can quickly adapt to changes in business requirements or environments by (re)-engineering the high-level business processes. The processes are then implemented by connecting their tasks with the functionality provided by the underlying software and systems. In comparison to manipulating business logics embedded in code, changing high-level processes is more efficient because these processes are more comprehensible and relevant to knowledge and skills of the business experts.

Process-driven SOA has roots in the business process management and workflow management field[160,165]. Various theoretical models have been used as the underlying foundation for describing business processes, such as Petri-net and its variants[56,79,140,165], π-calculus models[91,106,130,191], and state machines[19,36,44], to name but a few .

Several contemporary languages for process modeling and development have emerged based on these formal backgrounds, such as Business Process Definition Metamodel (BPDM)[120], Business

Process Modeling Notation (BPMN)[121], Business Process Execution Language (BPEL)[67,109], Business Process Modeling Language (BPML)[16], Event-Driven Process Chains (EPC)[34,80,142], WebSphere© FlowMark© Definition Language (FDL)[66], Integration Definition for Function Modeling (IDEF3)[96], jBPM Process Definition Language (jPDL)[75], UML Activity Diagram Extensions[112,116], XML Process Definition Language (XPDL)[181], and Yet Another Workflow Language (YAWL)[164]. Among of those languages, BPMN and BPEL are widely-accepted as (de factor) standards for business process design and implementation, respectively. We briefly summarize these languages in Table 2.1.

Name	Maintainer	Notation	Exchange format	Language type
BPDM[120]	OMG	Yes	Yes	Modeling/Execution
BPML[16]	BMPI	No	No	Execution
BPMN[121]	OMG	Yes	No	Modeling
BPEL[67,109]	OASIS	No	Yes	Execution
EPC[34,80,142]	-	Yes	No	Modeling
FDL[66]	IBM	Yes	Yes	Modeling/Execution
IDEF3[96]	KBS Inc.	Partial	No	Modeling
jPDL[75]	jBoss	Yes	Yes	Modeling/Execution
UML/AD[112,116]	OMG	Yes	Yes	Modeling
XPDL[181]	WfMC	No	Yes	Execution
YAWL[164]	Van der Aalst et al.	Yes	No	Modeling

Table 2.1: The summary of existing process modeling and development languages

2.3.1 Process-driven development life cycle

IEEE 1471 Std[69] defines a life cycle as *a framework containing the processes, activities, and tasks involved in the development, operation, and maintenance of a software product, which spans the life of the system from the definition of its requirements to the termination of its use*. Zur Mühlen[192] suggests a typical process life cycle including design, implementation, enactment, and evaluation phases. In Figure 2.3, we separate these phases into design time and run-time, respectively, and annotate with relevant process modeling/development languages and involving stakeholders .

In reality, there are many stakeholders, who might involve in the process development life cycle with different interests and concerning, such as clients, users, managers, business analysts, project managers, software architects, developers, evaluators, integrators, infrastructure administrators, and so on. From now on, we use two abstract roles to represent the stakeholders who are relevant to the process development life cycle, especially at the design time, including:

- *Business experts* are the stakeholders who understand business- and domain-oriented concepts and knowledge best. Business experts can analyze and formulate the process, from the business

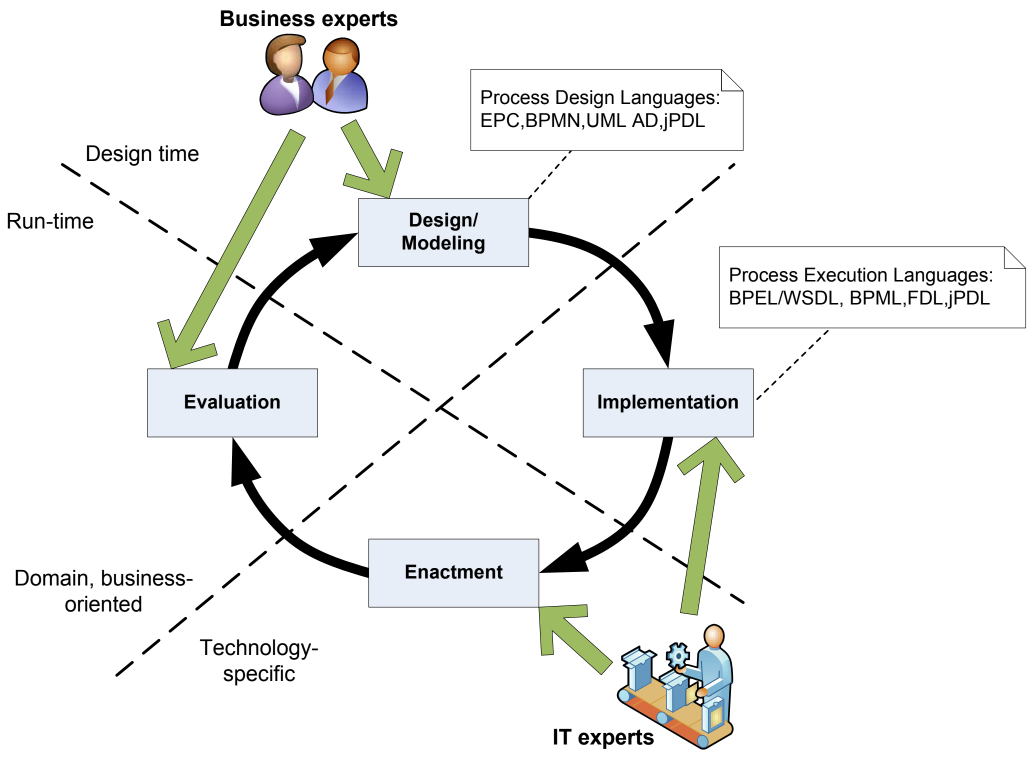

Business experts
Process Design Languages: EPC,BPMN,UML AD,jPDL
Design time
Run-time
Design/ Modeling
Process Execution Languages: BPEL/WSDL, BPML,FDL,jPDL
Evaluation
Implementation
Enactment
Domain, business-oriented
Technology-specific
IT experts

Figure 2.3: A typical process-driven development life cycle

point of view, to satisfy a certain business goal, but they pose limited, almost no, technical experience and skill.

- *IT experts*, in contrast to the business experts, pose expertise in technical landscape but not being familiar with business- and domain-oriented concepts. IT experts will involve in process implementation, deployment, and execution.

During the first phase, business experts create process models using high-level, notational modeling languages, such as EPC, BPMN, or UML Activity Diagrams. The business experts, with the help of high-level languages and tools, define necessary tasks and the order of these tasks in the business process in order to accomplish a certain business goals.

Process models designed by the business experts are usually conceptual and high-level so that they cannot be executed in process engines. Therefore, the process models are interpreted and implemented by IT experts in terms of executable descriptions using technology-specific languages, such as BPEL and WSDL, in the implementation phase. One the one hand, the IT experts translate high-level, business-oriented concepts, for instance, business tasks, into concrete, technical descriptions, e.g., service invocations or human interactions. On the other hand, these technical descriptions have to be adequately aligned with the existing IT infrastructures.

In the third phase, the implemented process can be deployed into a process engine for execution and monitoring. The engine executes process tasks according to the execution order described by the control flow. In the course of process execution, run-time events and execution logs can be collected for using in the evaluation phase. In this final phase, the ”as-is” processes can be analyzed, revised, and/or optimized, and become the input for subsequent iterations of the life cycle.

Our work in this dissertation mainly focuses on the process-driven development at the design time. Nonetheless, the resulting View-based Modeling Framework, based on our approach, can also support stakeholders at run-time by generating deployment and monitoring configurations. The View-based Modeling Framework presented in Chapter 3 supports forward engineering for business processes and is complemented with the *reverse engineering* toolchain presented in Chapter 4.

2.3.2 Forward and reverse engineering

Forward engineering and reverse engineering are two important processes in software development and maintenance. Forward engineering is *the traditional process of moving from high-level abstraction and logical, implementation-independent designs to the physical implementation of a system*[30]. Forward engineering denotes the progress from analyzing requirements through creating designs to implementing and deploying software and systems .

Reverse engineering tackles the opposite direction, i.e., the process of raising the abstraction level in order to help stakeholders understanding legacy code, especially when software or system's documentation is obsoleted or lost. The concept of *reverse engineering* has its origin in hardware technology. In this context, reverse engineering is the process of retrieving the specification of complex hardware systems through analyzing of system's structure, functions, and operations[135]. In the field of software engineering, *reverse engineering is the process of analyzing a system to identify the system's components and their relationships, and to create representations of the system in another form or at a higher level of abstraction*[14,30]. In the course of reverse engineering, the outcome of the implementation phase, i.e., either source code or binary code is translated back to relevant forms in the previous phases, for instance, system designs in the analysis phase.

During the reverse engineering process, the software system under consideration is unmodified. The evolution of existing software systems is often performed by a *re-engineering* effort. A re-engineering process includes the reverse engineering phase for comprehending and analyzing the “status quo” followed by some modifications or restructuring. Finally, the forward engineering phase is performed to derive system's implementations[30].

In Figure 2.4, we present a typical process development scenario wherein business experts design business processes in terms of BPMN diagrams while IT experts interpret the designs and implement the processes in BPEL and deploy them into a Web Services platform. Due to the popularity and high adoption of these languages in both academia and industry, we use these two languages as representatives for exemplifying, illustrating, and evaluating our approach in the sense that BPMN is the process design/modeling language and BPEL is the process implementation language. In

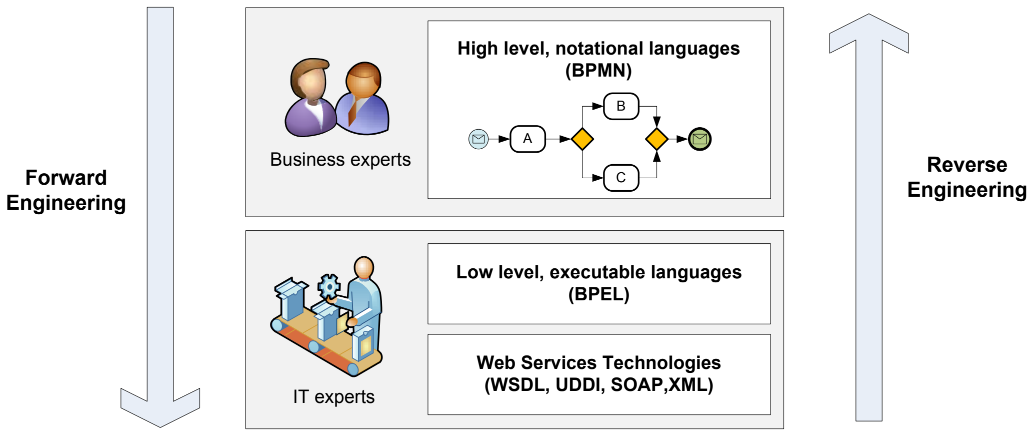

Forward Engineering
Business experts
High level, notational languages (BPMN)
A
B
C
IT experts
Low level, executable languages (BPEL)
Web Services Technologies (WSDL, UDDI, SOAP,XML)
Reverse Engineering

Figure 2.4: A typical process-driven SOA development scenario

the subsequent sections, we introduce the basic concepts, structures, and semantics of these two languages.

2.3.3 Process design languages

The Business Process Modeling Notation (BPMN) is a standard notational language introduced in May, 2004, by the Business Process Management Initiative (BPMI). Since 2005, BPMN has been mainly developed and maintained by the Object Management Group (OMG)[121]. The main goal of BPMN is to enable stakeholders, especially business experts, to specify, document, and communicate business processes in the form of process diagrams that are very similar to flowcharts. A process diagram is constructed from various notations and elements categorized into *Flow Objects*, *Connecting Objects*, *Swimlanes*, and *Artifacts*[121] .

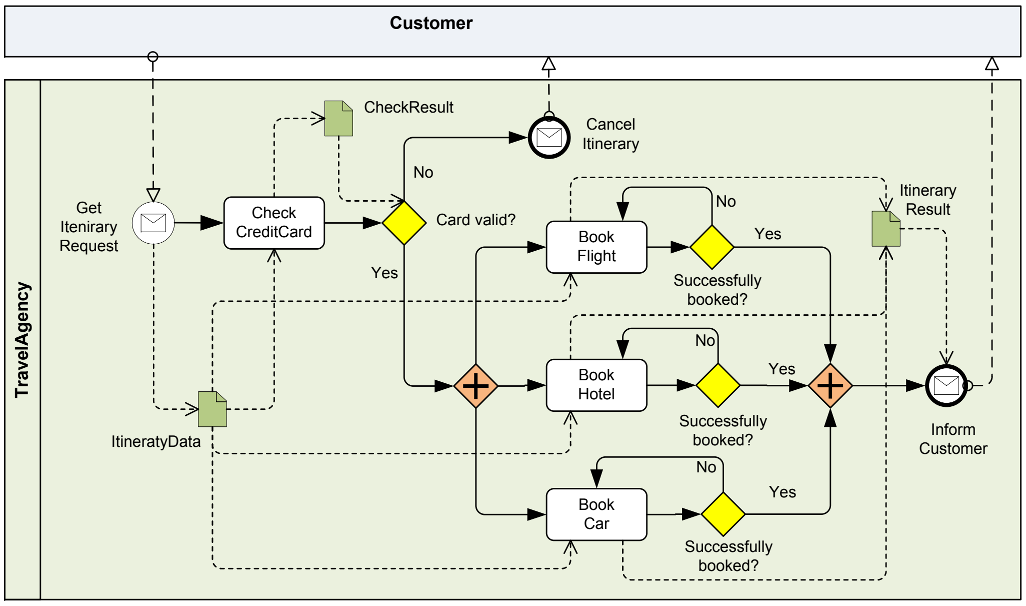


Figure 2.5: A sample BPMN diagram - a Travel Booking process

A typical BPMN process diagram consists of various nodes each of which is of the type *Flow Object*, including:

- *Event*: something that can happen during the process execution.
- *Activity*: a process task that must be performed.
- *Gateway*: a decision node that determines branching and merging of execution paths.

Connecting Objects are used to link these nodes, for instance, a *Sequence Flow* from node A to node B indicates that A must be done before B, a *Message Flow* from two process participants specifies that there is a message travel from one participant to the another, and an *Association* can associate a certain *Data Object*, that is a specific kind of *Artifact*, to a *Flow Object*. The activities of a process diagram then can be visually organized into various logical groups by using two essential kinds of *Swimlanes*: *Pool* and *Lane*. Figure 2.5 shows a Travel Booking Process[64], which is designed using BPMN notations. Further details can be found in the OMG BPMN specification[121].

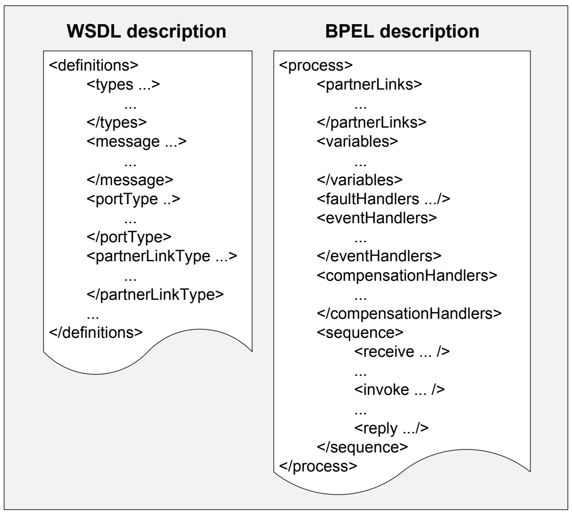
The Travel Booking process starts when a customer enters the data for his itinerary. Then, credit card information is checked for correctness and validity. If the credit card data is invalid, the process ends with a cancelation. Otherwise, three different tasks are performed for booking the flight, hotel, and car. These booking task are repeated until the corresponding reservations are successfully finished. When all the reservations are completed, either a successful notification along with different reservation codes or an error notification will be returned to the customer.

2.3.4 Process implementation languages

A process can be implemented in an executable language, such as the Business Process Execution Language (BPEL)[67,109]. BPEL processes are typically exposed to its consumers (e.g., other services or processes) as services with standard interfaces in Web Services Description Language (WSDL)[170].

BPEL is initially proposed as an executable language for specifying the orchestration of Web Services. The most notable and stable version of BPEL, BPEL 1.1, which has been widely adopted as a de facto standard for process development, is developed by a number of leading companies including IBM, Microsoft, SAP, BEA Systems, and Siebel Systems[67]. BPEL 1.1 is a convergence of two other languages: Microsoft's XLANG, which is based on the π-calculus, and IBM's Web Services Flow Language (WSFL), which is based on Petri-nets. Later on, BPEL has been submitted to OASIS for standardization in 2003. The newest version of this language is WS-BPEL 2.0 proposed by OASIS in April, 2007[109].

Figure 2.6 depicts a skeleton of a typical process implementation that includes two types of descriptions:



```
<definitions>
      <types ...>
            ...
      </types>
      <message ...>
            ...
      </message>
      <portType ..>
            ...
      </portType>
      <partnerLinkType ...>
            ...
      </partnerLinkType>
      ...
</definitions>
```

```
<process>
      <partnerLinks>
            ...
      </partnerLinks>
      <variables>
            ...
      </variables>
      <faultHandlers .../>
      <eventHandlers>
            ...
      </eventHandlers>
      <compensationHandlers>
            ...
      </compensationHandlers>
      <sequence>
            <receive ... />
            ...
            <invoke ... />
            ...
            <reply .../>
      </sequence>
</process>
```

Figure 2.6: A typical BPEL/WSDL implementation

WSDL A WSDL description describes the Web Service interface, including the *portType*(s) that declare the functionality exposed to its external partners, the *message*(s) that define the format of each *portType*'s input and output, and the *partnerLinkType*(s) that embody information about the process's partners.

BPEL A BPEL description, which is encoded in XML form, consists of the definition of the process, including process activities, the control structure, internal variables, and the various handlers for events, faults, and compensation.

In BPEL, an activity might carry an communication task or a data processing. The communication task is one of three types: receiving a message from a partner (`<receive>`), invoking other services or processes (`<invoke>`), or replying to the partner who sent a message before (`<reply>`). The data processing task in BPEL is merely assigning a variable or property with a certain value using an (`<assign>`) activity. Variables and properties of a BPEL process are declared using `<variable>`, `<property>` and `<propertyAlias>`, respectively.

These above process activities are orchestrated by different BPEL control structures, such as a `<sequence>` encapsulates many activities each of which runs sequentially after another, a `<flow>` allows the concurrent execution of enclosing activities, a `<switch>` changes the execution path to one of many choices of which the conditional value is evaluated to true, and a `<while>` performs an execution loop as long as the condition is still true.

```
<process name="TravelBooking">
  <!-- Links to process partners -->
  <partnerLinks>
    <partnerLink name="Client" partnerLinkType="Client" myRole="TravelBookingAgency"/>
    <partnerLink name="CreditBureau" partnerLinkType="CreditBureau" partnerRole="
         CreditBureau"/>
  <partnerLink name="CarAgency" partnerLinkType="CarAgency" partnerRole="CarAgency" />
  <partnerLink name="FlightAgency" partnerLinkType="FlightAgency" partnerRole="
      FlightAgency"/>
  <partnerLink name="HotelAgency" partnerLinkType="HotelAgency" partnerRole="HotelAgency"
      />
</partnerLinks>
<!-- Process variables-->
<variables>
  <variable name="itenriaryInput" messageType="ItineraryRequest"/>
  <variable name="itenriaryOutput" messageType="ItineraryResponse"/>
  <variable name="carBookingInput" messageType="CarBookingRequest"/>
  <variable name="carBookingOutput" messageType="CarBookingResponse"/>
  <variable name="cardCheckingInput" messageType="VerifyRequest"/>
  <variable name="cardCheckingOutput" messageType="Result"/>
  <variable name="flightBookingInput" messageType="FlightBookingRequest"/>
  <variable name="flightBookingOutput" messageType="FlightBookingResponse"/>
  <variable name="hotelBookingInput" messageType="HotelBookingRequest"/>
  <variable name="hotelBookingOutput" messageType="HotelBookingResponse"/>
  <variable name="isCarBooked" type="boolean"/>
  <variable name="isFlightBooked" type="boolean"/>
  <variable name="isHotelBooked" type="boolean"/>
</variables>
<!-- Main control flow -->
<sequence name="TravelBooking">
  <receive name="GetItineraryRequest" operation="book" partnerLink="Client" portType="
      TravelBooking" variable="itenriaryInput" createInstance="yes"/>
  <assign name="InitializeStatusVariable">
    ...
  </assign>
  <assign name="PrepareCardChecking">
    ...
  </assign>
  <invoke name="CheckCreditCard" operation="verify" inputVariable="cardCheckingInput"
      outputVariable="cardCheckingOutput" partnerLink="CreditBureau" portType="
      CreditBureau"/>
  <if name="ValidCreditCard">
    <condition>$cardCheckingOutput.response/status</condition>
    <sequence name="ItineraryProcessing">
      <flow>
        <sequence>
          <assign name="PrepareBookHotel">
            ...
          </assign>
          <while>
            <condition>not($isHotelBooked)</condition>
            <sequence>
              <invoke inputVariable="hotelBookingInput" name="BookHotel" operation="book"
                   outputVariable="hotelBookingOutput" partnerLink="HotelAgency" portType
                  ="HotelBooking"/>
              <assign name="GetHotelBookingStatus">
                <copy>
                  <from>$hotelBookingOutput.response/status</from>
                  <to variable="isHotelBooked"/>
                </copy>
              </assign>
            </sequence>
          </while>
        </sequence>
        <sequence>
          <assign name="PrepareBookFlight">
            ...
          </assign>
          <while>
            <condition>not($isFlightBooked)</condition>
            <sequence>
              <invoke name="BookFlight" operation="book" inputVariable="
                  flightBookingInput" outputVariable="flightBookingOutput" partnerLink="
                  FlightAgency" portType="FlightBooking"/>
               <assign name="GetFlightBookingStatus">
                <copy>
                  <from>$flightBookingOutput.response/status</from>
                  <to variable="isFlightBooked"/>
                </copy>
               </assign>
            </sequence>
          </while>
        </sequence>
        <sequence>
          <assign name="PrepareBookCar">
            ...
          </assign>
          <while>
            <condition>not($isCarBooked)</condition>
            <sequence>
              <invoke name="BookCar" operation="book" inputVariable="carBookingInput"
                  outputVariable="carBookingOutput" partnerLink="CarAgency" portType="
                  CarBooking"/>
              <assign name="GetCarBookingStatus">
                <copy>
                  <from>$carBookingOutput.response/status</from>
                  <to variable="isCarBooked"/>
                </copy>
               </assign>
            </sequence>
          </while>
        </sequence>
      </flow>
      <assign name="PrepareConfirmation">
        ...
      </assign>
      <reply name="InformCustomer" operation="book" partnerLink="Client" portType="
          TravelBooking" variable="iteniraryOutput"/>
    </sequence>
    <else>
      <sequence>
        <assign name="PrepareCancellation">
```

```
          ...
         </assign>
         <reply name="InformCancellation" operation="book" partnerLink="Client" portType="
             TravelBooking" variable="itenirary0utput"/>
       </sequence>
     </else>
   </if>
  </sequence>
</process>
```

Listing 2.1: The Travel Booking process in WS-BPEL 2.0

In addition, BPEL provides many other activities and structures for handling events, such as `<eventHandlers>`, `<pick>`, `<onMessage>`, and `<onAlarm>`, handling compensation, such as, `<compensationHandlers>` and `<compensate>`, processing faults and exceptions, such as, `<throw>`, `<faultHandlers>`, `<catch>`, and `<catchAll>`. For further details of BPEL process descriptions, please reference the BPEL 1.1[67] and WS-BPEL 2.0[109] as well as W3C WSDL specification[170]. Listing 2.1 presents the BPEL description of the Travel Booking process shown in Figure 2.5. The Travel Booking process exposes its functionality to the partners in terms of Web Services. The description of the exposed Web Services is presented in Listing 2.2. For the sake of readability, some information such as namespace declarations, namespace prefixes, namespace URIs and the details of data transformations of the `<assign>` elements have been omitted.

```
<definitions name="TravelBooking">
  <types>
    <schema>
       <import schemaLocation="travelbooking.xsd" namespace="http://travelbooking.at"/>
    </schema>
  </types>
  <!-- Exchange messages types -->
  <message name="ItineraryRequest">
    <part name="request" element="tns:ItineraryRequest" />
  </message>
  <message name="ItineraryResponse">
    <part name="response" element="tns:ItineraryResult" />
  </message>
  <!-- Main portType of the process -->
  <portType name="TravelBooking">
    <operation name="book">
      <input message="ItineraryRequest" />
      <output message="ItineraryResponse" />
    </operation>
  </portType>
  <!-- Types of links to the partners -->
  <partnerLinkType name="Client">
    <role name="TravelBookingAgency" portType="TravelBooking" />
  </partnerLinkType>
  <partnerLinkType name="CreditBureau">
    <role name="CreditBureau" portType="CreditBureau" />
  </partnerLinkType>
  <partnerLinkType name="CarAgency">
    <role name="CarAgency" portType="CarBooking" />
  </partnerLinkType>
  <partnerLinkType name="FlightAgency">
    <role name="FlightAgency" portType="FlightBooking" />
  </partnerLinkType>
  <partnerLinkType name="HotelAgency">
    <role name="HotelAgency" portType="HotelBooking" />
  </partnerLinkType>
</definitions>
```

Listing 2.2: WSDL Web Services description of the Travel Booking process

2.3.5 Process enactment

The process implementation, i.e., a set of relevant BPEL and WSDL descriptions, can be deployed on a BPEL execution engine (or so-called BPEL process engine) which instantiates the process and executes process activities according to the orchestration described by the control flow. There are a number of BPEL process engines, including open source products such as ActiveBPEL[1], Apache ODE[9], Intalio Server (dual license)[73], Orchestra[124], etc., and propriety products such as Oracle BPEL Process Manager (formerly known as Collaxa BPEL Server)[123], and IBM WebSphere Process Server[65]. Each process engine often requires additional configurations to deploy and execute business processes properly. Those configurations are merely developed by IT experts because platform- and technology-specific knowledge and skills are needed. Due to the specific process engine realization of each vendor, deployment configurations are different from each other.

2.4 Model-driven development

Model-driven development (MDD)[*] is an emerging software development methodology aiming at enhancing development speed and software quality[57,150]. In MDD, models are first-class artifacts that can be used not only for documentation and communication solely, but also for many other purposes, such as reasoning about business or solution domain, analyzing the architecture of the solution, generating code, and so on, in the software life cycle[20,57,150] .

Mello et al. proposed a definition of model[99]:

Definition 2.2. *A model is a coherent set of formal elements describing something (e.g., a system, bank, phone or a train) built for some purpose that is amenable to a particular form of analysis, such as communication of ideas between people and machines, completeness checking, race condition*

[*]Model-driven development is also called model-driven engineering (MDE) or model-driven software development (MDSD) in the literature.

analysis, test case generation, viability in terms of indicators such as cost and estimation, standards, and transformation into an implementation.

Kurtev et al.[84] presented a formal framework for MDD approaches, in which, the definitions of models and the system and their relationships are given as follows:

Definition 2.3 (System)**.** *System is a delimited part of the world considered as a set of elements in interaction.*

Definition 2.4 (Model)**.** *Model is a representation of a given system, satisfying the substitutability principle.*

Definition 2.5 (Principle of substitutability)**.** *A model M is said to be a representation of a system S for a given set of questions Q if, for each question of this set Q, the model M will provide exactly the same answer that the system S would have provided in answering the same question.*

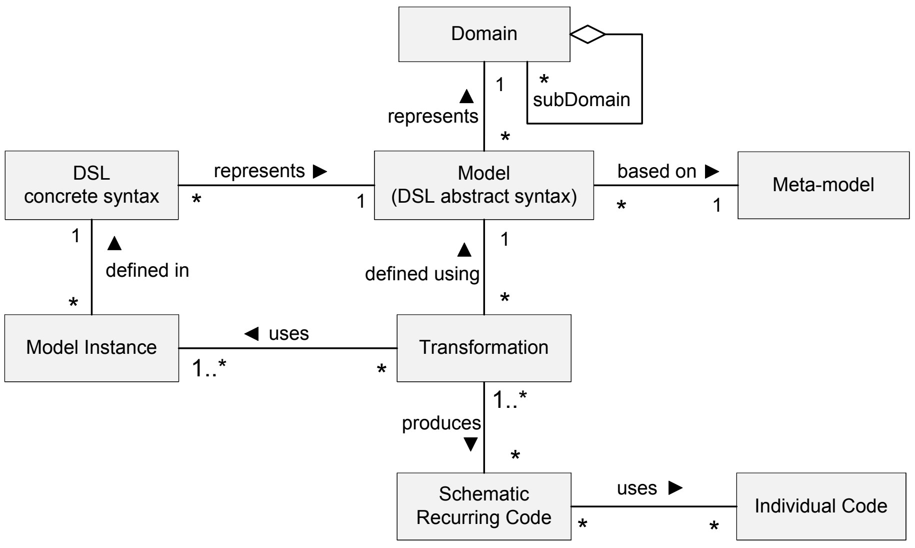


Figure 2.7: Fundamental concepts of Model-Driven Development

The OMG's MDA specification[113] is one specific MDD approach which is different from the MDD approach in general. The MDA primarily focuses on interoperability, platform independence, and merely based on, as well as often limited to, OMG specifications such as MOF[117], UML[116], OCL[118], etc. MDD is not bound to specific standards or technologies and advocates the idea of using customized, tailored domain-specific languages (DSL) to capture precise representations of structure, function or behavior of systems or software in a particular domain[150]. Figure 2.7 presents the key concepts of the MDD paradigm[150,186].

A *domain* under consideration may be divided into smaller sub-domains. Domain-specific languages (DSLs) are usually used for modeling domain concepts and knowledge in MDD. DSLs are small, sometimes declarative languages that can offer powerful expressiveness through appropriate notations and abstractions of a particular problem domain[166,179,183]. The most important characteristics of DSLs, with respect to general-purpose languages, are the compactness and expressiveness in a certain domain such that domain experts themselves can understand, analyze, validate, modify, and even develop DSLs[166,179,183].

A DSL has one or many concrete syntaxes, which are either textual or graphical. A DSL's concrete syntax can be used to define formal models. This concrete syntax is based on a language model (abstract syntax)[185] which specifies the structure and static semantics of the DSL's concrete syntax. DSL's abstract syntax is embodied in a meta-model. Thus, DSLs are sometime mentioned as *modeling language*. The model, i.e., DSL's abstract syntax has to conform to a meta-model that specifies structure and the semantics of that model[150].

Model transformation plays a very important role in which another model can be created from a source model according to some predefined mapping rules[51,57,150]. For instance, a platform-independent model (PIM) is mapped into a platform-specific model (PSM), or code is generated from a PSM[51,150]. The mappings between models, i.e., PIM to PIM or PIM to PSM, are *model-to-model* transformations, while the generations of code from PSMs are *model-to-code* transformations (or so-called, *code generation*)[51,150]. As such, model transformations establish relationships between models at the same or different levels of abstraction as well as between models and generated code. Therefore, they become very important factors in MDD for enhancing development automation and bridging abstraction levels. The results of code generation are usually schematic recurring code that form the basic skeleton of the systems or software under developing. The rests must be filled by non-generated code (or so-called, individual code or handwritten code) which are manually implemented[150].

Stahl and Völter state a number of advantages that MDD brings to software development, such as increasing productivity through automation, enhancing software quality and reusability by generating code from proven patterns and architectures, and improving manageability of complexity through appropriate abstractions[150]. Therefore, we use MDD paradigm to realize the *separation of abstraction levels* in order to organize the process representations according to specific stakeholders interests, for instance, high-level representations used by business and domain experts whilst technology-specific representations employed by IT experts.

2.5 Architectural views

Nowadays, software and systems are too complex such that they cannot be fully described by one single perspective but multiple point of views of different stakeholders are required[32,53]. In the field of software architecture, *architectural views* have been proposed as a solution to this problem. An architectural view (or view for short) is the central concept of ISO/IEC 42010, the precedence of IEEE 1471:2000, *Recommended Practice for Architectural Description of Software-Intensive Systems*[69], for documenting and analyzing software architectures. Other important concepts that make up the ISO/IEC Standard 42101 are *stakeholder*, *architectural concern*, and *viewpoint*[69] (see Figure 2.8, which is based on the conceptual model of the IEEE architectural description[69]).

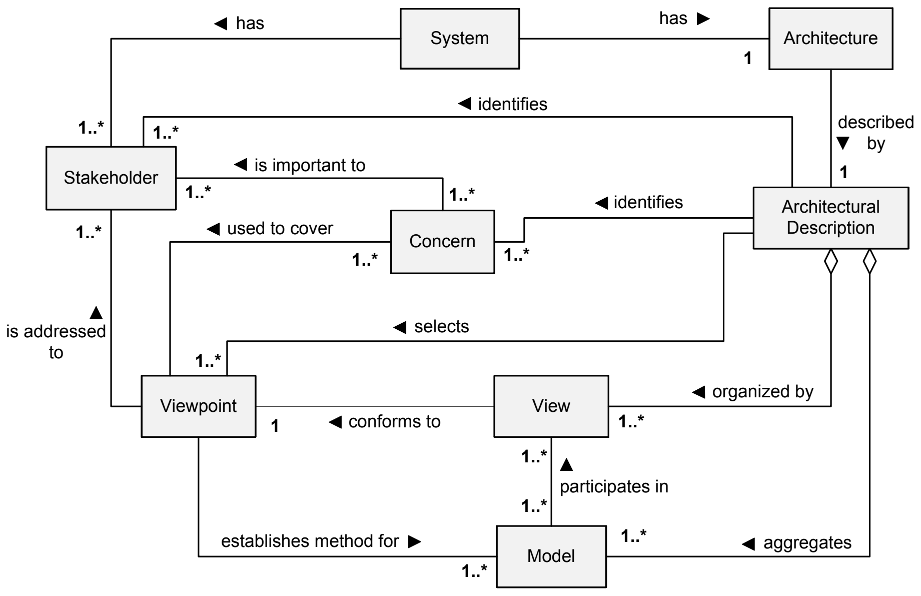


Figure 2.8: Architectural view and relevant concepts (IEEE 1471:2000)

ISO/IEC Std 42010[69] focuses on architectural descriptions which describe the fundamental organization of a system's components as well as their relationships to each other and to the environment. An architectural description is recommended to separate into a number of architectural views. The definition of architectural view is given as follows:

Definition 2.6 (Architectural view)**.** *An architectural view is a representation of a whole system from the perspective of a related set of concerns*[69].

Each view addresses one or more architectural concerns that the system stakeholders are interested in. A view must conform to a viewpoint that specifies conventions for creating and analyzing the view. In other words, a viewpoint defines the abstract syntax of the modeling language used to develop architectural views.

Several approaches proposed different interpretation of architectural views used for documenting software and systems. Kruchten presented the *"4+1" architectural view model* that offers four different viewpoints[82]. The *Logical View* is used for capturing system's functionality from an end-user's point of view, the *Process View* describes dynamic aspects of the system, *Development View* represents the system's organization from developers' perspectives, and *Physical View* illustrates the system from a system engineer's point-of-view, for instance, the mapping(s) of the software onto the hardware, how applications are installed and how they execute in a network of computers[82]. The fifth view, namely, use cases or scenarios view, selectively describe or consolidate the other views. Kruchten's "*4+1 views*" approach is the foundation of the Rational Unified Process (RUP)[83].

ISO/IEC Std 10746[74] proposes the *Reference Model of Open Distributed Processing (RM-ODP)* as a reference model to coordinate different open distributed processing (ODP) standards for distributed systems. The reference model defines a framework comprising five viewpoints for representing a basis of the system's specification. The *enterprise viewpoint* focuses on the purpose, scope and policies for the system. The *information viewpoint* focuses on the semantics of the information and the information processing. The *computational viewpoint* enables distribution through functional decomposition on the system into objects which interact at interfaces. The *engineering viewpoint* focuses on the mechanisms and functions required to support distributed interactions between objects in the system. The *technology viewpoint* focuses on the choice of technology of the system. Each viewpoint is based on a particular *viewpoint language* that defines concepts and rules for describing systems from the corresponding viewpoint.

Hofmeister et al.[61] present a method for software architecture design, namely, Siemens' 4 Views (S4V) that offers four architectural views to separate different engineering concerns in order to reduce the complexity of architecture design. The *conceptual view* captures the system's functionality in terms of decomposable, interconnected components and connectors. The *module view* reflects the organization of the system's architecture through two orthogonal structures: decomposition and layers. The *execution architecture view* represents the system's structure in terms of its runtime elements. Finally, the *code architecture view* is used to describe the organization of the software artifacts.

Recently, Clements et al. examine several architectural styles as well as provide guidance for choosing appropriate architectural views for analyzing, reconstructing, design communication, deriving code, and so on[32]. The authors emphasize the importance of using multiple views in the architectural description to communicate to the various stakeholders.

The notion of views offers a separation of concerns that has the potential to resolve the complexity challenges in process-driven SOAs. Therefore, we propose in this dissertation a *view-based approach* to modeling of process-driven SOAs. Perspectives on process representations and service interactions are used as central concepts in our approach. We then formalize the notion of view using view models as well as organize views into different abstraction levels using the MDD paradigm. In particular, our approach offers separated process views, such as the collaboration, information, control flow, event, transaction, and human view models, each of which represents a specific part of the business process. These views can be viewed and manipulated separately to get a better understanding of a particular concern, or they can be integrated to produce a richer view or a thorough view of the processes and services. We dedicate the whole Chapter 3 to present and discuss all aspects of our view-based, model-driven approach.

Chapter 3

View-based, Model-driven Approach for Process-driven SOAs

The orthogonal features, when combined, can explode into complexity ("The Philosophy of Ruby").

— *Yukihiro Matsumoto*

3.1 Fundamental concepts

A typical business process embodies various tangled concerns, such as the control flow, data processing, service and process invocations, fault handling, event handling, human interactions, transactions, to name but a few. The entanglement of those concerns increases the complexity of process development and maintenance as the number of involved services and processes grow.

In order to deal with this complexity, we use the notion of *architectural views* (or *views* for short) to describe the various process concerns. In particular, a view is a representation of one particular concern of a process. We devise different view models for formalizing the concept of architectural view. A *view model* specifies the abstract syntax, i.e., the view's structure, and the static semantics, i.e., the meaning of the view's structure which a view derived from that view model must conform to. The view models, in turn, are defined on top of the *meta-model*. Figure 3.1 shows the fundamental concepts of our view-based, model-driven approach along with their relationships to each other.

In Figure 3.1, we present the formalizations of basic process concerns such as the control flow, service invocations, and data processing, in terms of the *FlowView*, *CollaborationView*, and *InformationView* model, respectively. These view models, which are presented in detail in the subsequent sections, are built up around a *Core* model. The Core model is intentionally developed for conceptually representing the essence of a business process. That is, the Core model covers three distinct concepts: the process, the relationships between process and the environment, i.e., the services, and the internal representation of the process, i.e., the process views. Process concerns described by the view models merely relate to these concepts in the sense that each concern involves either the process's interior or exterior, or both. In other words, the other view models derive and extend the foundational concepts

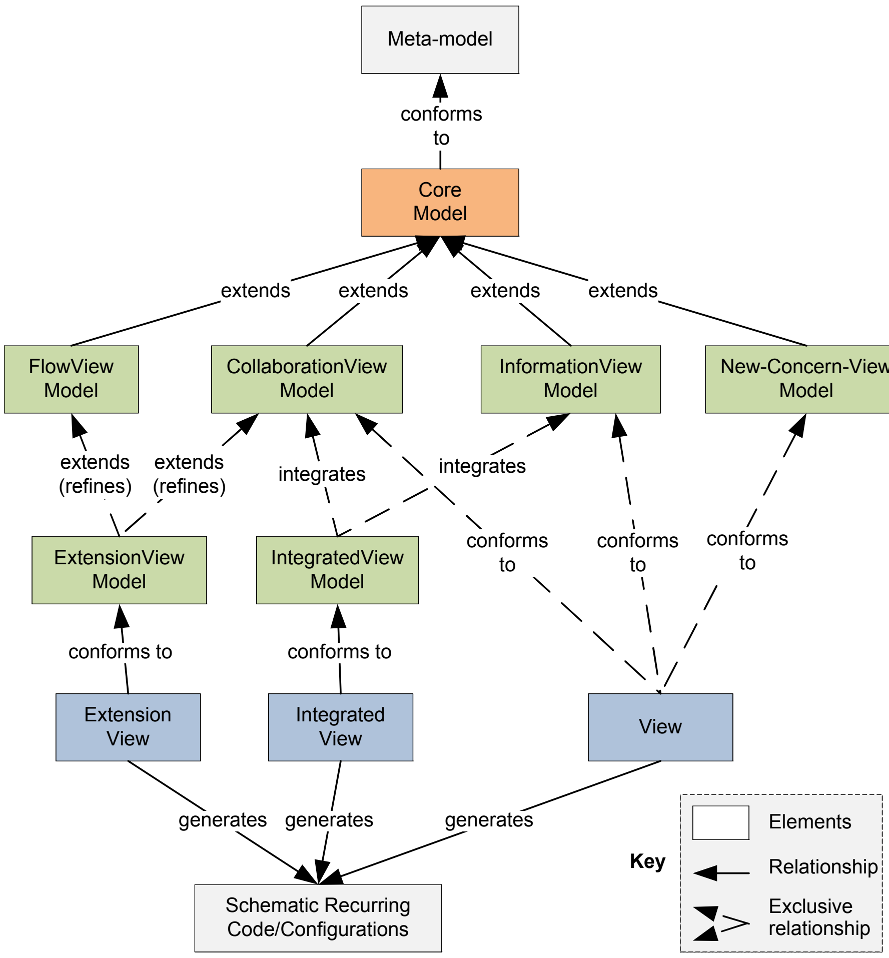


Figure 3.1: Fundamental concepts of the view-based, model-driven approach

provided in the Core model as shown in Figure 3.1. As a result, the Core model plays an important role in our approach because it provides the basis for extending and integrating view models, and establishing and maintaining the dependencies between view models.

Nonetheless, our view-based approach is not limited to these concerns, but can be extended to cover various other concerns, for instance, human interactions, transactions, event handling have been realized as extensions[62,97]. A new concern can be integrated into our approach by using a corresponding *New-Concern-View* model that extends the basic concepts of the Core model and defines additional concepts of that concern. By adding new view models for additional process concerns, we can extend the view-based approach along the *horizontal dimension*, i.e., the dimension of process concerns, to deal with the complexity caused by the various tangled process concerns.

There are many stakeholders involved in process development at different levels of abstraction. For instance, business experts require high-level abstractions that offer domain or business concepts concerning their distinct knowledge, skills, and needs, while IT experts merely work with low-level, technology-specific descriptions. The MDD paradigm provides a potential solution to this problem by separating the platform-independent and platform-specific models. A platform-independent

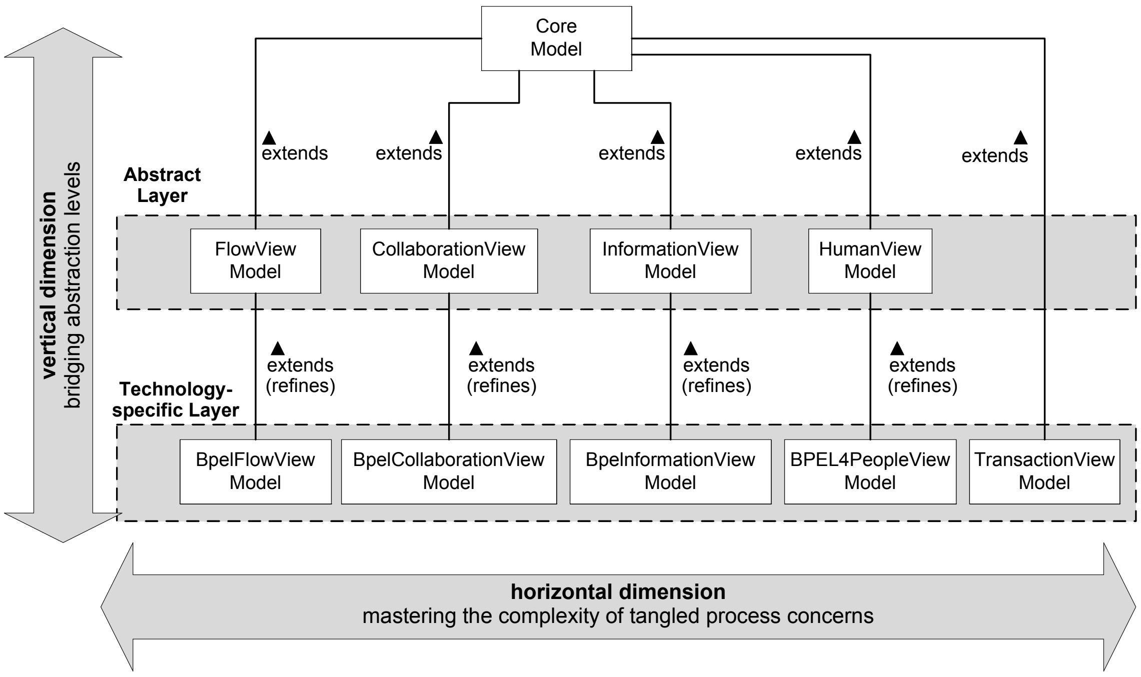


Figure 3.2: Layered architecture of the view-based, model-driven approach

model is a model of a software system that does not depend on the specific technologies or platforms used to implement it while a platform-specific model links to particular technologies or platforms[51,113]. Leveraging this advantage of the MDD paradigm, we devise a model-driven stack that has two basic layers: abstract and technology-specific. The abstract layer includes the views without the technical details such that the business experts can understand and manipulate. Then, the IT experts can refine or map these abstract concepts into platform- and technology-specific views. The technology-specific layer contains the views that embody concrete information of technologies or platforms. On the one hand, a technology-specific view model can be directly derived from the Core model, such as the *TransactionView* model shown in Figure 3.2. One the other hand, a technology-specific view model can also be an extension of an abstract one, for instance, the *BpelCollaborationView* model extends the *CollaborationView* model, the *BPEL4PeopleView* model extends the *HumanView* model[62], etc., by using the model refinement mechanism (see Figure 3.2). By refining an abstract layer down to a technology-specific layer, our view-based approach helps bridging the abstraction levels along the *vertical dimension*, i.e., the dimension of abstraction, which is orthogonal to the *horizontal dimension* described in the previous paragraph (see Figure 3.2).

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

Based on the aforementioned view model specifications, stakeholders can create different types of views for describing specific business processes. These process views can be instances of the concerns' view models, extension view models, or integrated view models (see Figure 3.1). They can be manipulated by the stakeholders to achieve a certain business goal, or adapt to new requirements in business environment or changes in technology and platform. Finally, we provide *model-to-code transformations* (or so-called *code generations*) that take these views as inputs and generate process implementations and deployment configurations. The resulting code and configurations, which may be augmented with hand-written code, can be deployed in process engines for execution.

In the subsequent sections, we present in detail the concepts and mechanisms described above as well as the realization of these concepts in our prototype, namely, the View-based Modeling Framework (VbMF).

3.2 View-based Modeling Framework

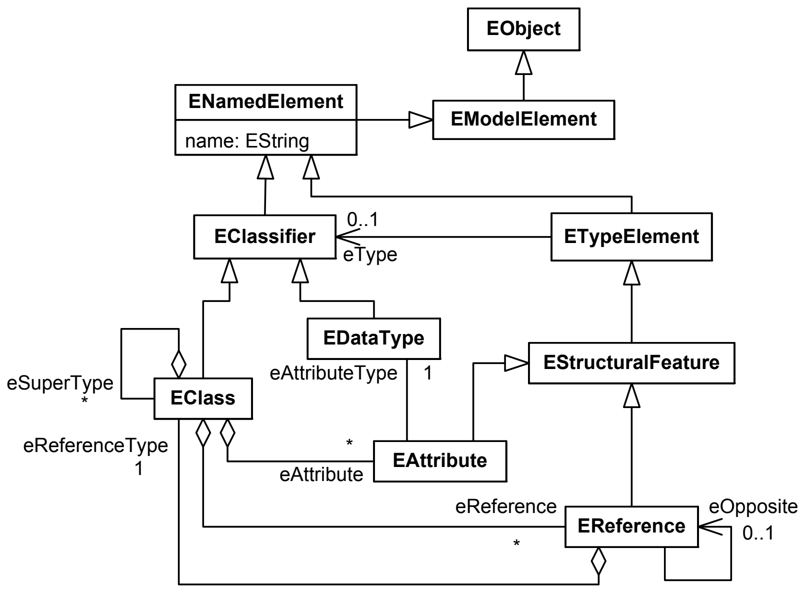


Figure 3.3: The Ecore meta-model – an MOF-compliant meta-model

The meta-model, which is the basis of our view models, can be simple or more elaborate like Meta-Object Facility (MOF)[117]. For better integration and interoperability with existing modeling and development tools which are using MOF-compliant models and utilizing the XMI standard[119] for persisting models, we choose the Eclipse Modeling Framework (EMF)[38] for realizing our view-based, model-driven approach. Figure 3.3 shows an excerpt of the EMF Ecore meta-model which is used as the basis for deriving our view models.

Based on Ecore, we firstly define the Core model as the foundation for the View-based Modeling Framework. After that, different view models are derived from the Core model to represent the various process concerns. In Figure 3.4, we present the overall architecture of VbMF in which modeling artifacts such as view models and views are created, manipulated, and maintained. We categorize distinct functional components of VbMF as shown in Table 3.1.

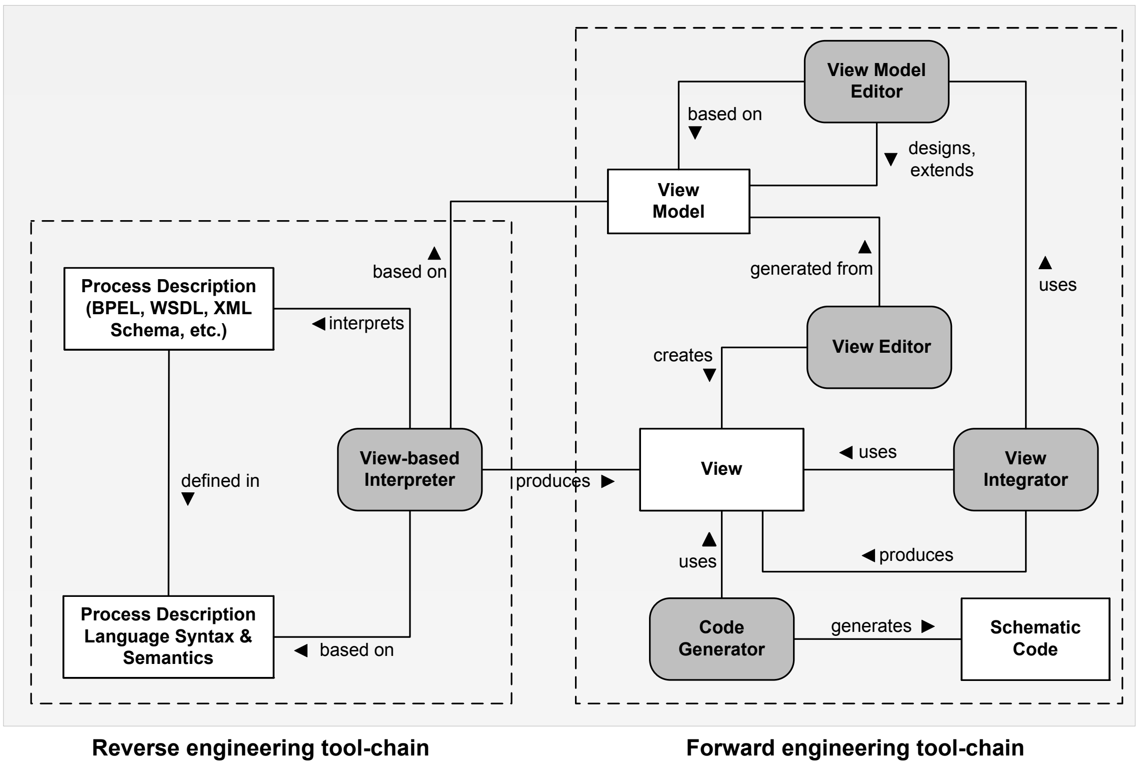


Figure 3.4: View-based modeling framework architecture

These components of VbMF shape the forward and reverse engineering toolchain shown in Figure 3.4. In the VbMF forward engineering toolchain, abstract views are designed first. Next, these abstract views are refined down to their lower level counterparts, which are technology-specific views. The code generator then uses the technology-specific views to produce schematic process code and/or the necessary configurations for deployment and execution. In the course of the reverse engineering toolchain, the view-based interpreters take as input legacy process descriptions and produce abstract or technology-specific views that can be re-used later in the forward engineering toolchain.

3.3 Formalization of basic process concerns

In this section, we present in detail the (semi-)formalized representations of basic process concerns in terms of view models, including the control flow, process interactions, and data handling concerns.

3.3.1 Core model

Aiming at the openness and the extensibility, we devised the Core model as a foundation for creating the other view models (see Figure 3.5). The Core model comprises a number of conceptual classes: *View*, *Process*, and *Service*. Each of these classes can be extended further. The Core model conforms to the Ecore meta-model shown in Figure 3.3. In particular, classes shown as rectangle shapes, for

Component	Description
View model editors	View models editors are based on the view models. Using these editors, a new view model can be developed from scratch by deriving the Core model. Moreover, an existing view model can also be extended with some additional features to form a new view model. We use the basic Ecore editor generator[38] to develop our view model editors.
View editors	View editors can be (semi)-automatically generated from VbMF view models based on the editor generations provided in EMF[38]. The view editors support stakeholders in creating new views or editing existing ones. For the sake of simplicity and illustration purposes, we choose the simplest visualization of the view editors, which is a tree-view editor.
View integrators	View integrators aid the stakeholders in combining view models to produce a richer view or a more thorough view of a certain process.
Code generators	Code generators generate schematic recurrent code and configurations from one or many views. Before generating outputs, the code generators will validate the conformity of the input views against the corresponding view models.
View-based interpreters	View-based interpreters are the key contributions of our view-based reverse engineering approach which can be used to extract appropriate views from existing process descriptions such as BPMN, WSDL, and BPMN (cf. Chapter 4)

Table 3.1: View-based Modeling Framework components

instance, *NamedElement*, *Namespace*, *Element*, *View*, *Service*, and *Process*, are instances of *EClass*. The attributes of a class, for instance, *name* and *nsURI* of the *NamedElement* class, are instances of *EAttribute*, and the relationships between two classes are instances of *EReference*.

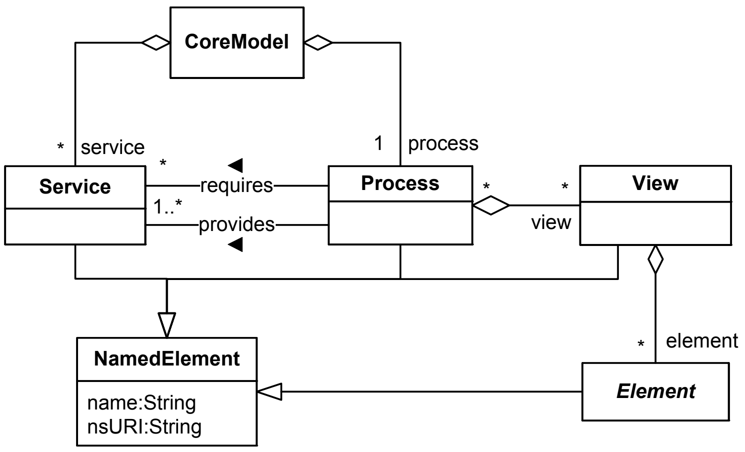


Figure 3.5: The Core model

At the heart of the Core model is the *View* class that represents the concept of architectural view. Each specific view (i.e., a specialization of the *View* class) represents one perspective of a *Process*. A *Service* stands for functionality that the corresponding *Process* provides or requires. The meaning of *Process* is self-explained. A *View* also stands for a container of several *Elements* representing the objects that describe process interior. Different instances of each of those classes can be distinguished through the features of the common supertype *NamedElement*, defining a name property (*name*) and a namespace URI (*nsURI*). Figure 3.6 depicts a Core model of the Travel Booking process described in Figure 2.5 and Listing 2.1 and 2.2.

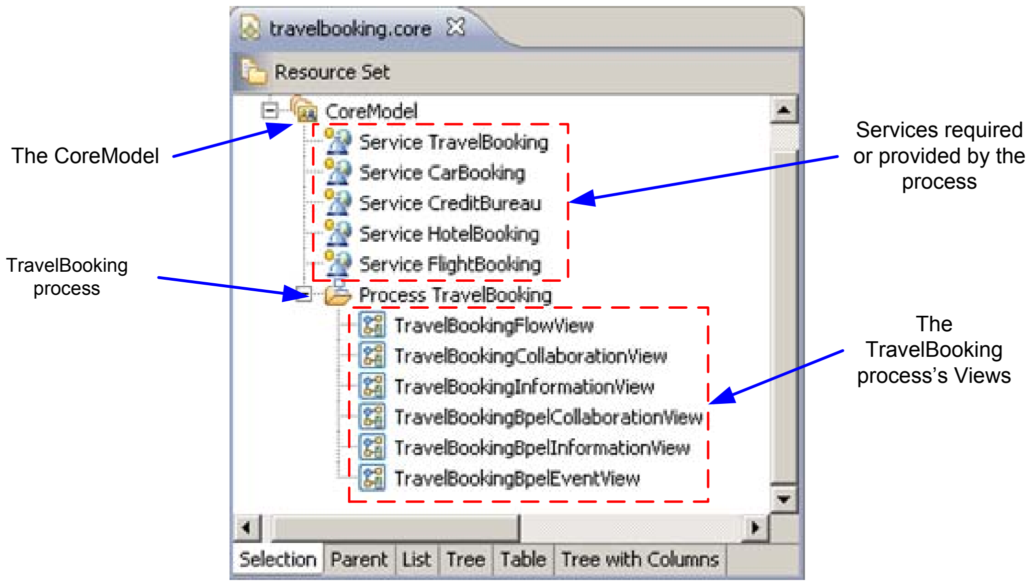


Figure 3.6: The Travel Booking process's Core model

The view models that represent concerns of a business process are mostly defined by extending the concepts of the Core model. As such, the view models are independent of each other, and the Core model becomes the place where the relationships among the view models are maintained. Hence, the relationships between concepts of the Core model are necessary not only for extension and dependency management, but also for defining the integration points used to merge view models as mentioned in the description of the integration mechanisms below.

3.3.2 FlowView model

As we mentioned above, the notion of process is central in process-driven SOAs. Therefore, the flow of control is one of the most important concerns of a process which is formalized by a FlowView model. A FlowView model embodies 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 in order to achieve a certain goal.

There are several approaches to modeling process control flows such as state-charts, block structures[67,109], UML Activity Diagram extensions[116], Petri-nets[56,140,162,165], and so on. Despite of this diversity in control flow modeling, it is widely accepted that existing modeling languages have the following patterns in common: *Sequence*, *Parallel Split*, *Synchronization*, *Exclusive Choice*, *Simple Merge*, and *Structured Loop*[161,162]. Furthermore, zur Mühlen and Recker report a statistical result

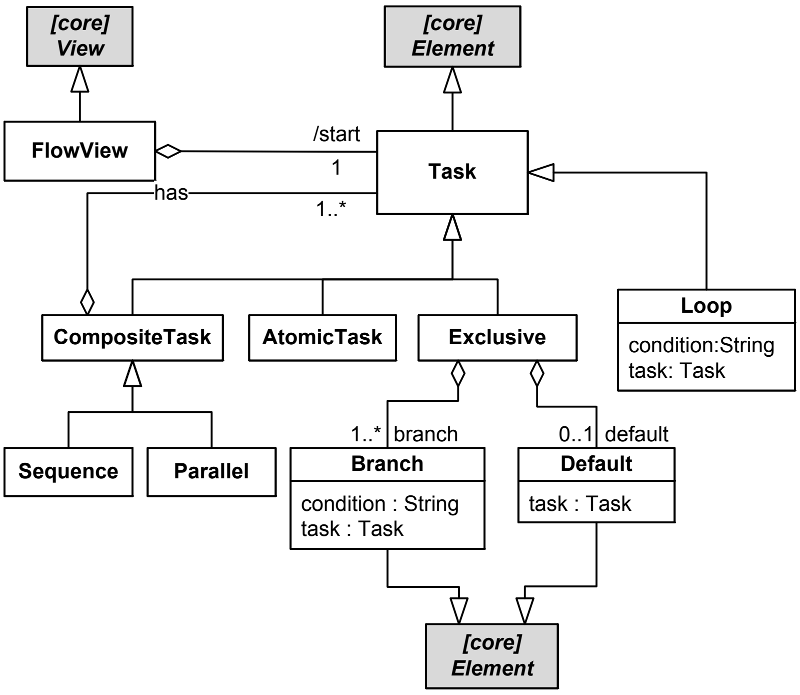


Figure 3.7: The FlowView model

in[193] how modeling elements are commonly used in practice. The statistics indicates only a few of control structures are regularly used by process modelers in reality[193], which is accordant with the above observation of the common patterns. Therefore, we adopt these patterns as the building blocks of the FlowView model. Other, and more advanced, patterns can be added later by using extension mechanisms to augment the FlowView model. We define the semantics of the control structures in the FlowView model with respect to these patterns in Figure 3.2.

The primary class of the FlowView is the *Task* (see Figure 3.7), which is the base class for other classes such as *Sequence*, *Parallel*, and *Exclusive*. Another important entity in the FlowView is the *AtomicTask*, a specialization of *Task*, that represents a single concrete action, such as a service invocation, a data processing task, etc. The actual description of each *AtomicTask* is only given in another specific view rather than in the FlowView. For instance, a service invocation is described in a CollaborationView, while a data processing action is specified in an InformationView. In other words, an *AtomicTask* is a placeholder or a reference to a specific activity. It is an interaction or a data processing task, which is defined respectively in the corresponding concern's view. Therefore, every *AtomicTask* becomes an integration point that can be used to merge a FlowView with another view, for instance, an InformationView or a CollaborationView. The *CompositeTask* is an abstract representation of a group of related activities.

Sequence, *Parallel*, *Exclusive*, and *Loop* classes are straightforward realizations of the patterns described in Table 3.2. Both *Sequence* and *Parallel* consist of unlimited number of process tasks which are executed sequentially or concurrently. A *Sequence* or a *Parallel* completes when all of the enclosed tasks have finished. An *Exclusive* structure consists of one or many *Branches*, each of which embodies

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 described in[162,163].
Parallel	All activities of a *Parallel* structure are executed in parallel. The subsequent activity of the *Parallel* structure is only enabled after the completion of all activities in the *Parallel* structure. The semantics of the *Parallel* structure is equivalent to a control block starting with the *Parallel Split* pattern and ending by the *Synchronization* pattern presented in[162,163].
Exclusive	Only one of many alternative paths (*Branches* or *Default*) of control inside an *Exclusive* structure is enabled according to a condition value of each branch. After the active branch finished, the process continues with the subsequent activity of the *Exclusive* structure. The semantics of the *Exclusive* structure is equivalent to a control block starting with the *Exclusive Choice* pattern and ending by the *Simple Merge* pattern proposed in[162,163].
Loop	The *Loop* executes the containing task zero or more times while a pre-defined condition still evaluates to true. The condition is evaluated before the first iteration of the *Loop* and is re-evaluated prior to each of subsequent iterations. Once the condition evaluates to false, the thread of control passes to the task immediately following the *Loop*[162,163].

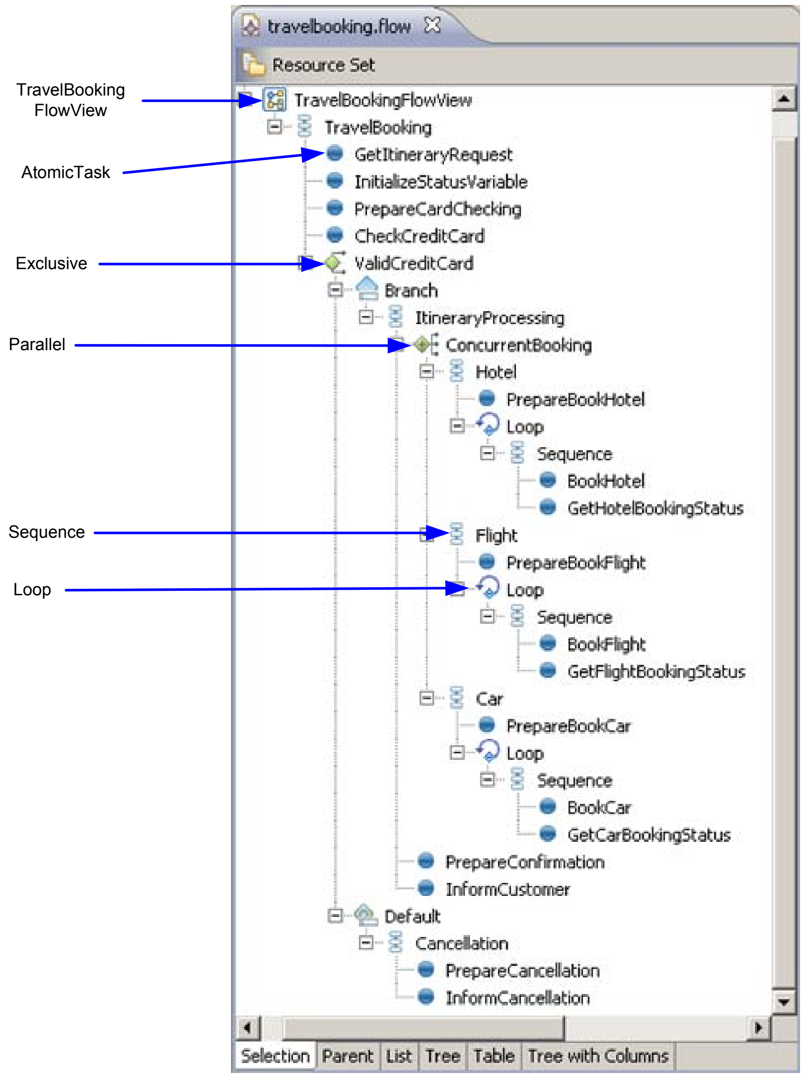
Table 3.2: The semantics of basic control structures of the FlowView model

a condition that decides the path of execution at run-time, and references to a particular *Task* being executed if the corresponding condition is satisfied. A *Default* may be included in an *Exclusive* structure in which the exceptional conditions that are not covered by any *Branch*'s condition are handled. A *Loop*, which is a specialization of *Task*, can be used for describing an iteration of a certain process task under some conditions. In Figure 3.8, we illustrate a FlowView that represents the control flow of the Travel Booking process.

3.3.3 CollaborationView 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, may use the process. All business functions required or provided by the process are typically exposed in terms of standard interfaces (e.g., WSDL *portTypes*). So far, we have modeled these concepts in the Core model by the relationships between the two classes *Process* and *Service*. The CollaborationView model extends these concepts to represent the interactions between the process and its partners (see Figure 3.9).

In the CollaborationView model, the *Service* of the Core model is extended by a tailored and specific *Service* class that exposes a number of *Interfaces*. Each *Interface* provides some *Operations*, each of which stands for a particular functionality provided by the associated *Service*. The *provided* attribute of a *Service* is either *true* if that *Service* is provided by the process or *false* otherwise.



```
TravelBookingFlowView
  TravelBooking
    GetItineraryRequest
    InitializeStatusVariable
    PrepareCardChecking
    CheckCreditCard
    ValidCreditCard
      Branch
        ItineraryProcessing
          ConcurrentBooking
            Hotel
              PrepareBookHotel
              Loop
                Sequence
                  BookHotel
                  GetHotelBookingStatus
            Flight
              PrepareBookFlight
              Loop
                Sequence
                  BookFlight
                  GetFlightBookingStatus
            Car
              PrepareBookCar
              Loop
                Sequence
                  BookCar
                  GetCarBookingStatus
          PrepareConfirmation
          InformCustomer
      Default
        Cancellation
          PrepareCancellation
          InformCancellation
```

Figure 3.8: The Travel Booking process's FlowView

Each party involved in the process is described by a *Partner* class. The ability and the responsibility of a process's *Partner* are modeled by the associated *Role*. Functionality provided by a *Partner* is modeled by the *Interface* associated with a certain *Role* of the *Partner*. A collaboration between the process and a Partner is represented by an *Interaction* linking to that *Partner*. The type of communication in each *Interaction* is either input, output, or both. In addition, an *Interaction* has a *createInstance* attribute that indicates whether the Interaction triggers the execution of the process or not. This feature is needed to inform the process engine when to instantiate and start executing the process. It serves as the entry point for the process just like the *main()* method in programming languages such as C/C++ and Java. Last but not least, an *Interaction* task is performed whenever the process invokes a service or another process, or the process is invoked itself via its provided interface. Therefore, each *Interaction* is associated with a specific service's *Interface* and *Operation* (see Figure 3.9).

Partner, *Role*, *Interface*, *Operation*, and *Interaction* are subtypes of the *Element* from the Core model.

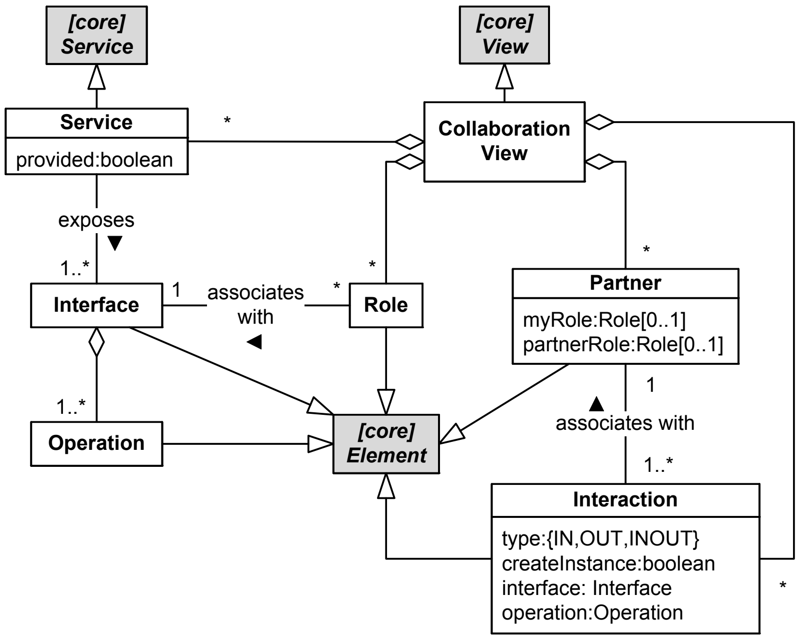


Figure 3.9: The CollaborationView model

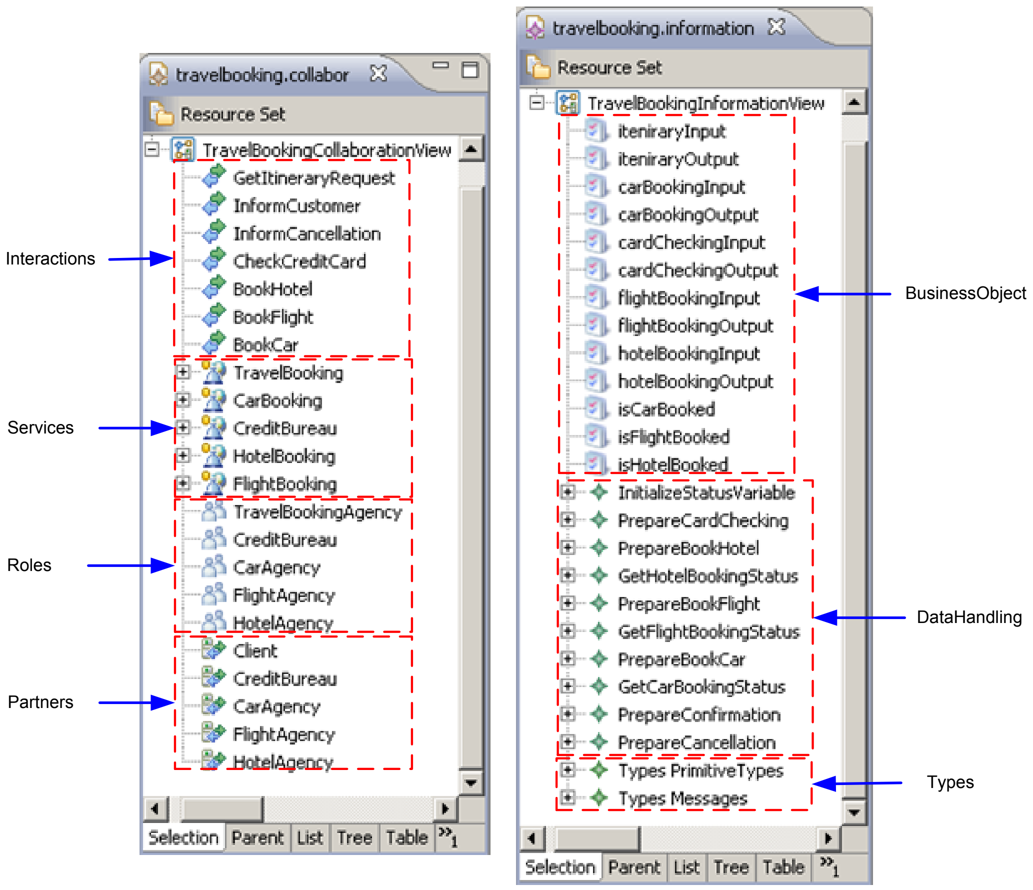
It means that those are classes of the CollaborationView, which is one specific subtype of the process *Views*. Furthermore, those classes inherit the *name* attribute that can be used in the named-based matching algorithm described in Section 3.5 to integrate view models.

In summary, the relationships between a process and the external environment, i.e., other processes or services, are modeled by a number of interactions. Each of these interactions requires a specific role of a certain partner of the process. A partner's role is reified through the functionality provided in terms of services each of which may expose one or many operations via its well-defined interfaces. We presents a CollaborationView of the Travel Booking process in Figure 3.10a.

3.3.4 InformationView model

The third process concern we consider in the context of this chapter is data handling which is formalized by the InformationView model (see Figure 3.11). This view model involves the representation of data object flows inside the process and message objects traveling back and forth between the process and its partners.

In the InformationView model, the *BusinessObject* class, which may have a generic type, namely, *Type*, is the abstraction of any piece of information, for instance, a purchase order received from the customer, or a request sent to a banking service to verify the customer's credit card, and so forth. Each InformationView comprises of a number of *BusinessObjects*. *Messages* exchanged between the process and its partners or data flowing inside the process might go through some *Transformations* wherein input data is converted or extracted to yield new pieces of output data. The transformations are performed inside a *DataHandling* class. The source or the target of a certain transformation is



a) TravelBooking CollaborationView **b) TravelBooking InformationView**

Figure 3.10: A CollaborationView and an InformationView of the Travel Booking process

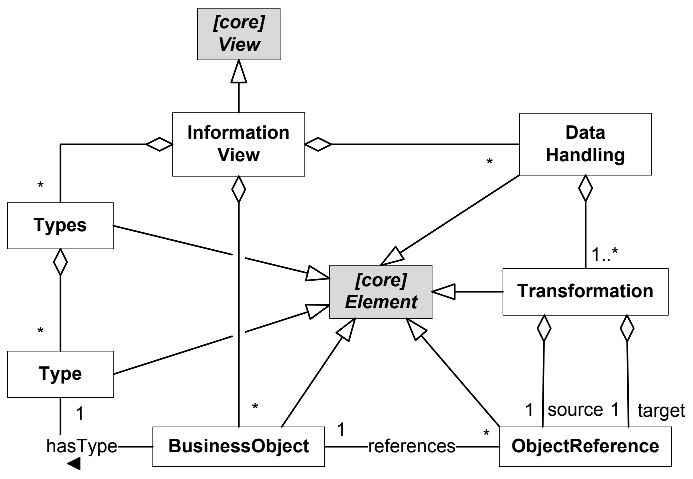


Figure 3.11: The InformationView model

represented by an *ObjectReference* that holds a reference to a particular *BusinessObject*.

To summarize, the InformationView model conceptualizes the data that are either processed inside the process or exchanged between the process and its partners. The Information view is the subtype of the *View* class. Its elements, described in the previous paragraphs, specialize the *Element* class of the Core model. Figure 3.10b presents an InformationView of the Travel Booking process.

3.4 Formalizations of additional process concerns

To illustrate the extensibility of our approach along the horizontal dimension in order to cover newly added process concerns, we present in this section some extra view models developed to formalize long-running transaction, event handling, and human interaction concerns of processes.

3.4.1 HumanView model

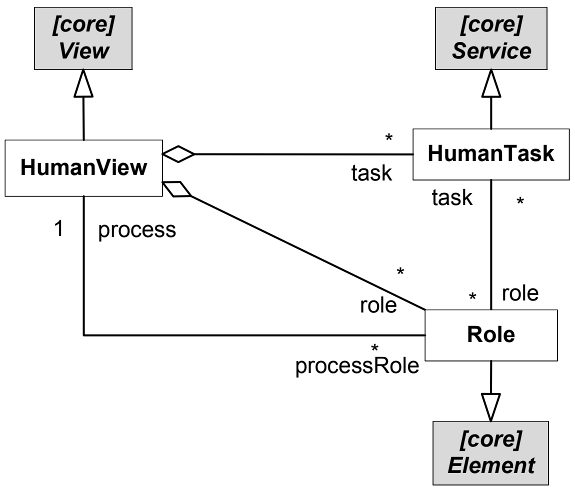


Figure 3.12: The HumanView model

A process task's functionality is often accomplished by invoking external services. However, business processes might also involve human interactions, for instance, a customer submits a purchase order that triggers an order processing process, a manager accepts or declines a loan approval in a loan approval process, and so on. We call such process elements *HumanTask*s. *HumanTask*s, thus, are special process tasks that are performed by a person. *HumanTask*s may need certain input values as well as a task description and may produce some output data.

Besides the *HumanTask* as a special process element, the HumanView model shown in Figure 3.12 defines human roles and their relationships to the respective process and tasks. *Roles* are abstracting concrete users that may play certain roles. The HumanView thus establishes a role-based abstraction. This role-based abstraction can be used for role-based access control (RBAC)[46]. RBAC, in general, is administered through roles and role hierarchies that mirror an enterprise's job positions and organizational structure. Users are assigned membership into roles consistent with a user's duties, competency, and responsibility.

Examples for different types of *Roles* are task owner, process supervisor, and escalation recipient. By binding, for instance, the role of a process supervisor to a process, RBAC can define that those users that are associated with this role may monitor the process execution. Similarly, the owner of a *HumanTask* may complete the task by sending results back to the process.

We can specify an activity as defined within a FlowView to be a *HumanTask* in the HumanView that is, for instance, bound to an owner, i.e., the person who performs the task. Likewise, process stakeholders can be specified for the process by associating them with the HumanView.

3.4.2 TransactionView model

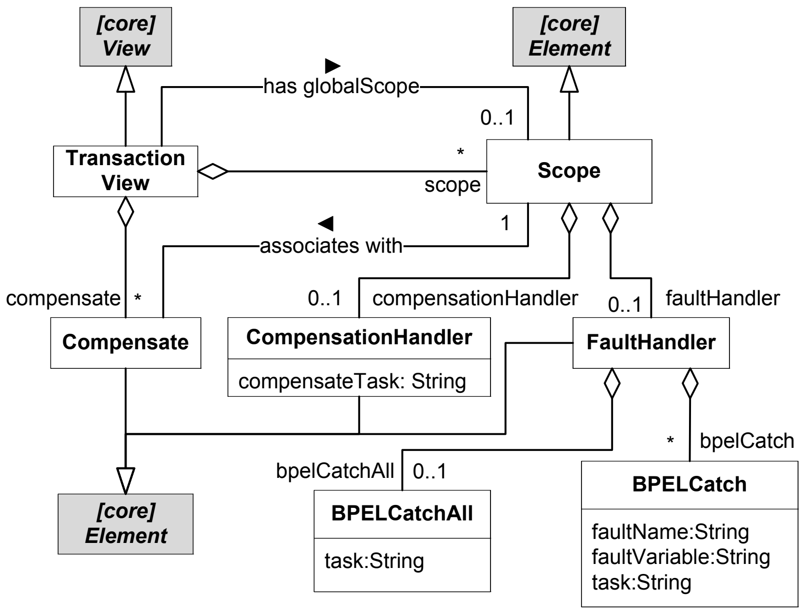


Figure 3.13: The TransactionView model

The TransactionView model depicted in Figure 3.13 represents long-running transactions of processes. This view embodies concepts which are specific to the BPEL standards[67,109]. A *Scope* specifies the behavior context of one or group of process activities. In such a context, developers may define a *CompensationHandler*, which invokes necessary activities to compensate the functionality done by the activities associated with the corresponding scope, and a *FaultHandler* for dealing with occurring faults or exceptions. In addition, the TransactionView also consists of some *Compensates* that explicitly invokes a certain *CompensationHandler* of the linked *Scope*. For more details of the transaction handling concern, refer the BPEL standards[67,109].

Scope, *Compensate*, *CompensationHanlder*, and *FaultHanlder* are subtypes of the *Element* of the Core model because these elements, independently developed though, need to be integrated into the FlowView model of a process to augment the transactional semantics for that process.

3.4.3 EventView model

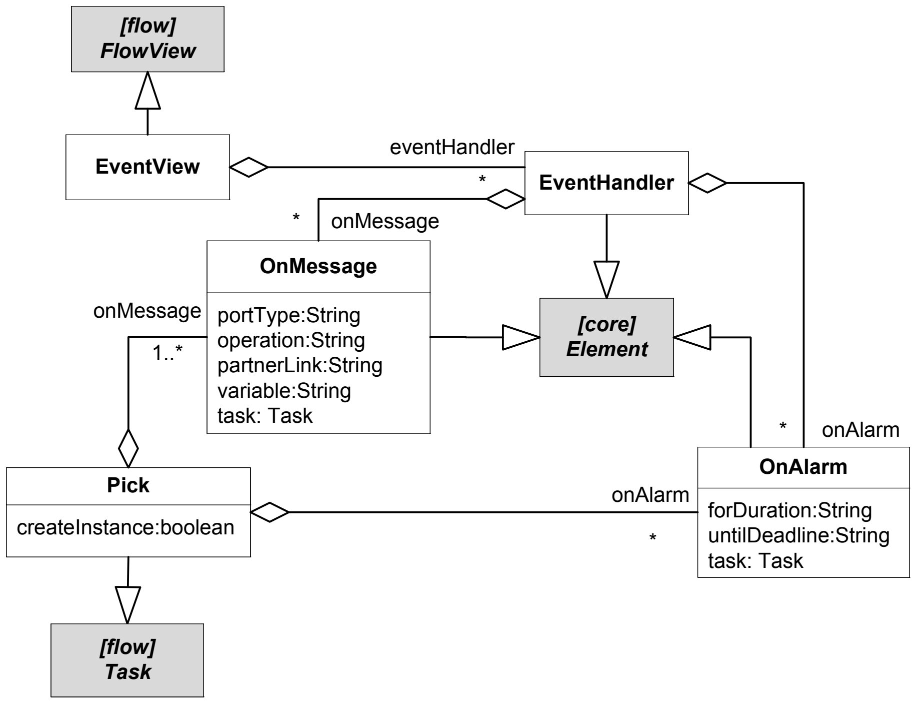


Figure 3.14: The EventView model

The EventView model (see Figure 3.14) describes the process ability to manage and handle events. On the one hand, a process can be triggered by message-based events, for instance, callbacks from the process partners. This type of events is represented by the *OnMessage* class. On the other hand, the process or its activities is also activated by timing events, for instance, the purchase order process will wait for a manual payment from the customer for one week. We model this type of events by the *OnAlarm* class that allows the modelers to specify timing period via one of its attributes: *forDuration* or *untilDealine*. When corresponding event arrive, *OnMessage* and *OnAlarm* will activate the enclosing process task.

There are two kinds of containers of *OnMessage* and *OnAlarm*: *Pick* and *EventHandler*. The difference between *Pick* and *EventHandler* is the logical scope of effect, the explicitness, and the synchrony. The modelers will use a *Pick*, which is a specialization of *Task* of the FlowView, in a certain part of the process control flow to specify that the process must wait for particular events to occur. The *Pick* is done when all the process tasks in the activated event are completed. The *EventHandler* is used to handle the events occurring in a certain scope – a logic group of process tasks. If the events do not arrive, the control flow of the scope, which the *EventHandler* is attached to, is not affected or interfered by the *EventHandler*. In other words, an *EventHandler* stands loose from the rest of the process tasks in the scope such that when there are process tasks in the scope active, it is still possible to receive and handle an event specified in the *EventHandler*.

3.5 View manipulation mechanisms

3.5.1 Extension mechanism

During the process development life cycle, various stakeholders take part in the process with different needs and responsibility. For instance, the business experts - who are familiar with business concepts and methods - sketch blueprint designs of the business process functionality using abstract and high-level and/or notational languages such as flow-charts, EPC[80], BPMN[121], and UML Activity Diagram extensions[116]. Based on these designs, the IT experts implement the business processes using executable languages, such as BPEL[67,109]. As a consequence, these stakeholders work at different levels of abstraction of the process models.

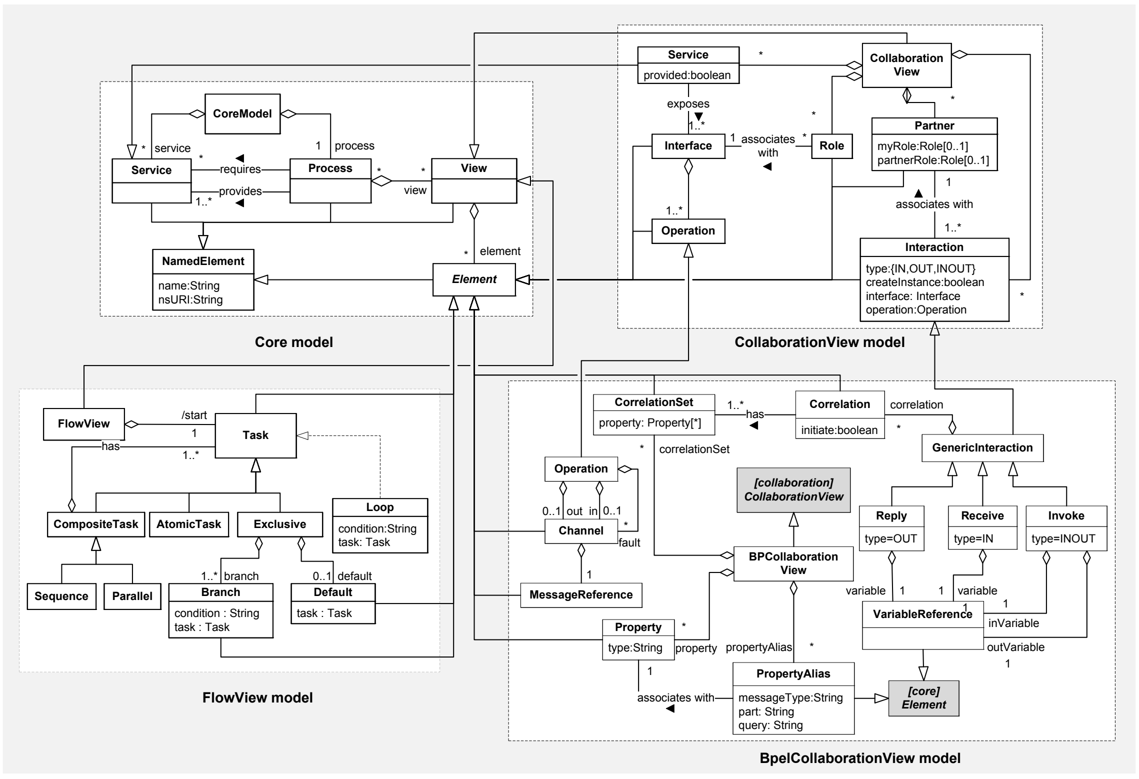
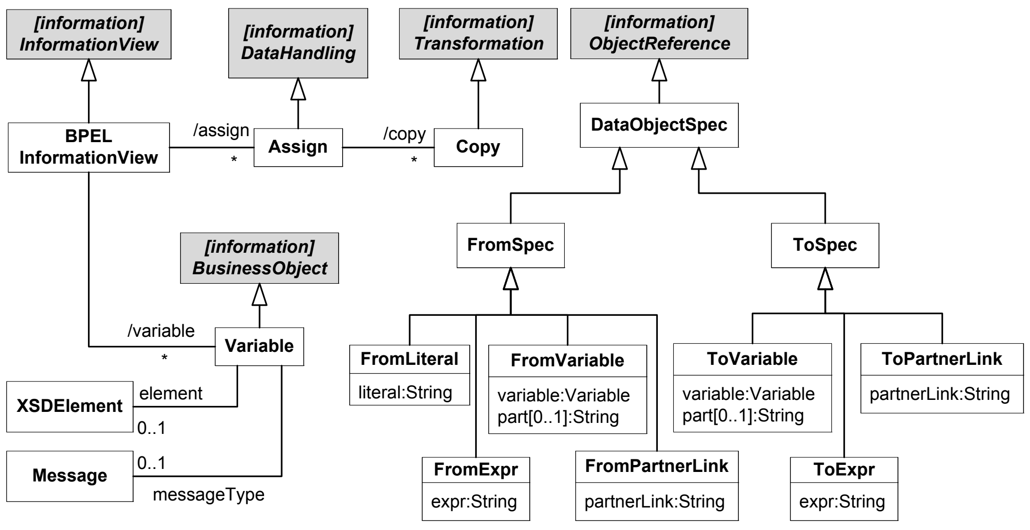
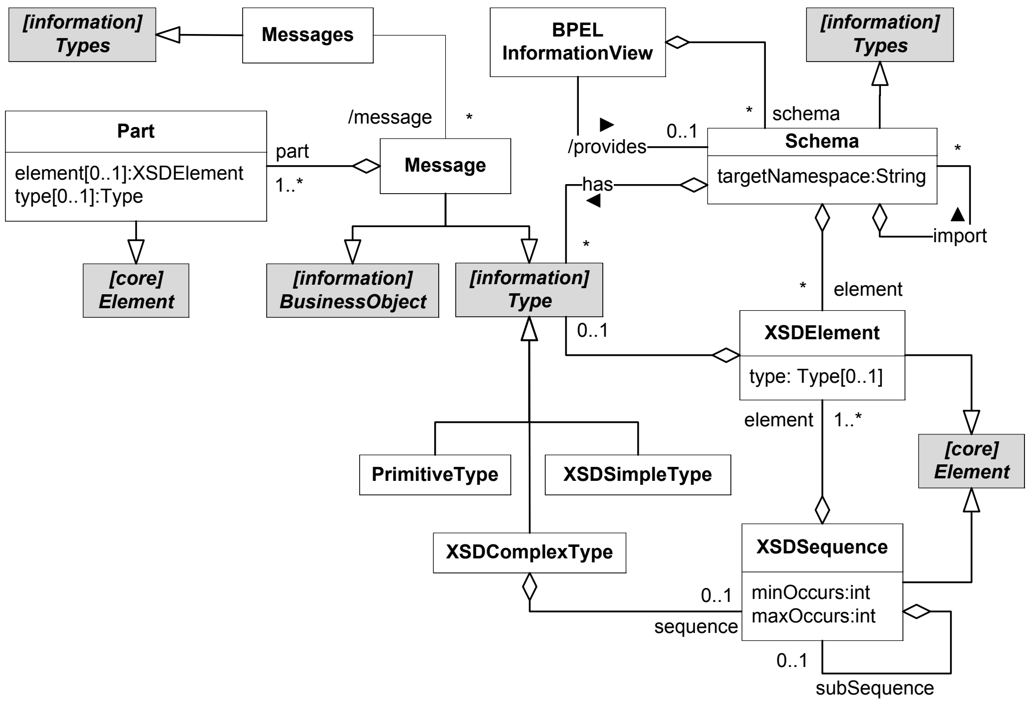


Figure 3.15: Illustration of VbMF extensibility based on the Core model

The FlowView, CollaborationView, and the InformationView model are the cornerstones to create abstract views. These abstract views aim at representing the high-level, domain-related concepts, and therefore, they are in the first place useful the business experts. Moreover, these views are also useful for technical experts and for enhancing the communications between different the business experts and IT experts. Nonetheless, the IT experts often need more information, especially platform- and technology-specific descriptions. According to the specific requirements on the granularity of the



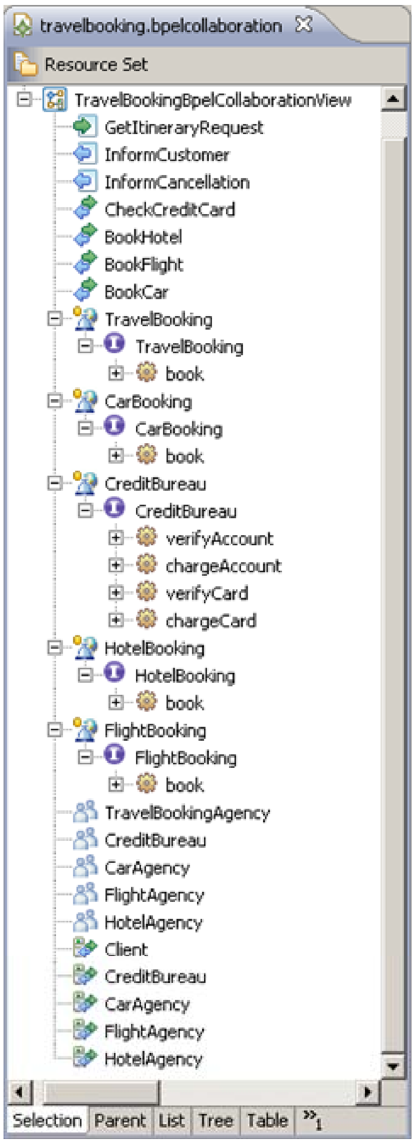
(a) BpelInformationView model – Part 1



(b) BpelInformationView model – Part 2

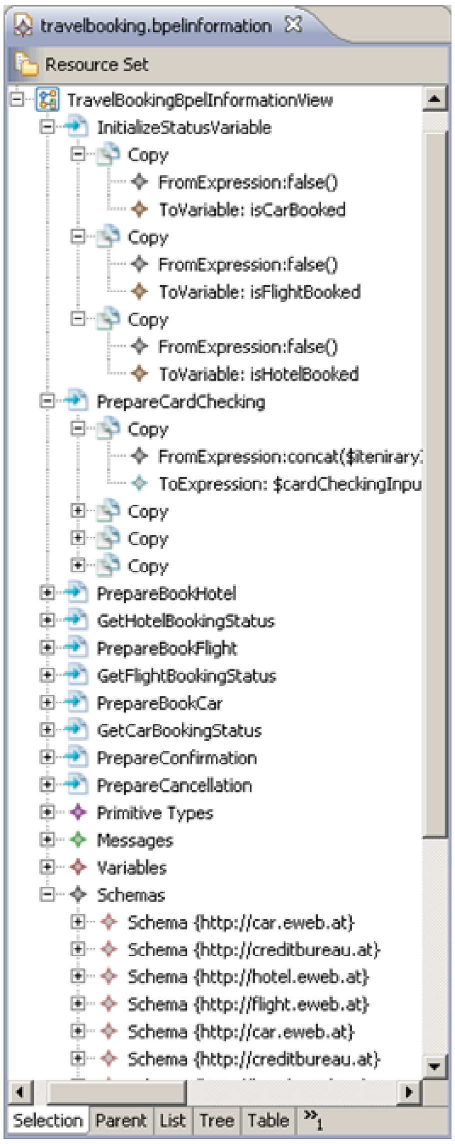
Figure 3.16: BpelInformationView model - an extension of the InformationView model

views, we can refine these views toward more concrete, technology-specific views using *extension mechanisms*.



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

a) TravelBooking BpelCollaborationView



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

b) TravelBooking BpelInformationView

Figure 3.17: A BpelCollaborationView and BpelInformationView of the Travel Booking process

A view refinement is performed by, firstly, choosing adequate extension points, and consequently, applying extension methods to create the resulting view. An extension point of a certain view is a view's element which is enhanced in another view by adding additional features (e.g., new element attributes, or new relationships with other elements) to form a new element in the corresponding view. Extension methods are modeling relationships such as generalization, extend, etc., that we can use to establish and maintain the relationships between an existing view and its extension. For instance, the FlowView, CollaborationView, and InformationView model are extensions of the Core model using the generalization relationship. In Figure 3.15, we demonstrate the extensibility of the Core model by the FlowView and the CollaborationView model as well as the extensibility of the CollaborationView model by an enhanced view model, the BpelCollaborationView model. A similar BPEL-specific extension has also been developed for the InformationView model. For the sake of readability, the extension of the InformationView model presented in Figure 3.16 comprises two parts that represent data processing (see Figure 3.16a) and data structure (see Figure 3.16b), respectively. Figure 3.10(a) depicts a BpelCollaborationView – an extension of the CollaborationView – and Figure 3.10(b) shows a BpelInformationView – an extension of the InformationView – of the Travel Booking process. These extension views inherit and refine the elements of their abstract counterparts.

In the same way, more specific view models for other technologies can be derived. In addition, other process concerns such as long-running transactions, event handling, human interactions, as described above, have been extended by using new view models derived from the basic view model using the same approach as used above.

3.5.2 Integration mechanism

In our approach, the FlowView – as the most important concern in process-driven SOA – is often used as the central view. Views can be integrated via integration points to produce a richer view or a more thorough view of the business process. We devise a name-based matching algorithm for realizing the view integration mechanism (see Algorithm 3.1). This algorithm is simple, but effectively used at the view level (or model level) because from a modeler's point of view in reality, it makes sense, and is reasonable, to assign the same name to the modeling entities that pose the same functionality and semantics. Nonetheless, other view integration approaches such as those using class hierarchical structures or ontology-based structures are applicable in our approach with reasonable effort as well.

Before discussing in detail the name-based view integration, we propose the definition of conformity of model elements and integration points. Let m be an element of a certain view model, the symbol $\hat{m}$ denotes the hierarchical tree of inheritance of m, i.e., all elements which are ancestors of m, and $m.x$ denotes the value of the attribute x of the element m.

Definition 3.1 (Conformity). *Let M_1, M_2 be two view models and $m_1 \in M1$ and $m_2 \in M2$. Two elements m_1 and m_2 are **conformable** if and only if m_1 and m_2 have at least one common parent type in their tree of inheritance or m_1 is of type m_2, or vice versa.*

Using $m_1 \uparrow m_2$ to denote m_1 and m_2 are **conformable**, Definition 3.1 is given as:

$$\boldsymbol{m_1 \uparrow m_2} \iff (\hat{m}_1 \cap \hat{m}_2 \neq \varnothing) \vee (m_1 \in \hat{m}_2) \vee (m_2 \in \hat{m}_1)$$

Definition 3.2 (Integration point). *Let M_1, M_2 be two view models and two views V_1, V_2 be instances of M_1 and M_2, respectively. A couple of elements e_1 and e_2, where $e_1 \in V_1$ and $e_2 \in V_2$, e_1 is an instance* *of m_1, and e_2 is an instance of m_2, is an **integration point** between V_1 and V_2 if and only if m_1 and m_2 are **conformable** and e_1 and e_2 have the same value of the attribute "**name**".*

Using $I(e_1, e_2)$ to denote the integration point between two views V_1 and V_2 at the elements e_1 and e_2, and $x > y$ to denote x is an instance of y, Definition 3.2 can be written as:

$$I(e_1, e_2 | e_1 \in V_1, e_2 \in V_2, e_1 > m_1, e_2 > m_2, V_1 > M_1, V_2 > M_2) \iff (m_1 \uparrow m_2) \wedge (e_1.name = e_2.name)$$

The main idea of the name-based matching for view integration is to find all integration points $I(e_1, e_2)$ between two views V_1 and V_2 and merge these two views at those integration points. The merging at a certain integration point $I(e_1, e_2)$ is done by creating a new element which aggregates the attributes and references of both e_1 and e_2. This idea is realized by the code from line *7th* to line *20th* in Algorithm 3.1.

The complexity of the name-based matching algorithm is approximately $O(k + l + k \times l)$, where $k = |V_1|$ and $l = |V_2|$. This complexity can be significantly reduced by generating and maintaining a configuration file containing the integration points of every pair of views with tool support. This is reasonable in reality because the integration points can be directly derived from the relationships between the FlowView and other views. For instance, a developer uses the modeling framework to realize an *AtomicTask* that performs a service invocation. The service invocation is defined by an *Invoke* activity in the BpelCollaborationView with the same name. The tooling then can automatically derive an integration point between the *AtomicTask* and the *Invoke* activity. Later on, the view integration algorithm only loads the configuration files and performs view merging rather than iterates through all elements of two input views. This way, the complexity of the view integration algorithm is reduced to approximately $O(P)$, where P is the number of integration points between V_1 and V_2. We note that $P \leq k \times l$. In reality, the number of elements which are used for view integration are often much less the total number of elements of the containing view, and therefore, $P \ll k \times l$). Nonetheless, this approach requires additional support by the modeling framework for deriving and maintaining the integration points, which is beyond the scope of our work.

3.6 Code generation

There are two basic types of model transformations: model-to-model and model-to-code. A *model-to-model* (M2M) transformation maps a model to another model. *Model-to-code* (M2C), so-called *code generation*, often produces schematic recurring, and maybe executable, code, that makes up the software products from the models[150]. In both types of transformation, the transformation rules are often defined, firstly, based on the source model. In addition, the transformation rules in M2M require the specification of the target model while the transformation rules in M2C may need specific platform-definition models[51,150].

Algorithm 3.1: View integration by name-based matching

```
   Input: View $V_1$ and view $V_2$
   Output: Integrated view $V_{12}$
1  begin
2      $V_{12}$.initialize()
3      $E_1 \leftarrow V_1$.getAllElements()
4      $E_2 \leftarrow V_2$.getAllElements()
5      $V_{12}$.addElements($E_1$)
6      $V_{12}$.addElements($E_2$)
       /* Look for conformable elements and integrate their attributes and
          references into a newly created element */
7      foreach $e_1 \in E_1$ do
8          $found \leftarrow$ false
9          while not found do
10             $e_2 \leftarrow E_2.next()$
11             if $(e_1.name = e_2.name) \wedge (e_1.superType \uparrow e_2.superType)$ then
12                 $found \leftarrow$ true
13                 create $e_{new}$
14                 $e_{new}$.mergeAtributes($e_1$.getAttributes(), $e_2$.getAttributes())
15                 $e_{new}$.mergeReferences($e_1$.getReferences(), $e_2$.getReferences())
16                 $V_{12}$.addElements($e_{new}$)
17                 $V_{12}$.removeElements($e_1$,$e_1$)
18             end
19         end
20     end
21     return $V_{12}$
22 end
```

In our view-based approach, the model transformations are merely model-to-code that take as input one or many views and generate schematic code in executable process languages, such as BPEL[67,109] (see Figure 3.18). In the literature, there are different code generation techniques such as template-based transformation, inline generation, or code weaving have been proposed[150]. In our proof-of-concept implementation, we use the template-based technique which has been realized using the XPand language of the openArchitectureWare framework[122] to implement the code generations. Nevertheless, other above-mentioned techniques can be leveraged in our approach with reasonable modifications as well.

Figure 3.18 shows our view-based code generator which is defined based on the view models.

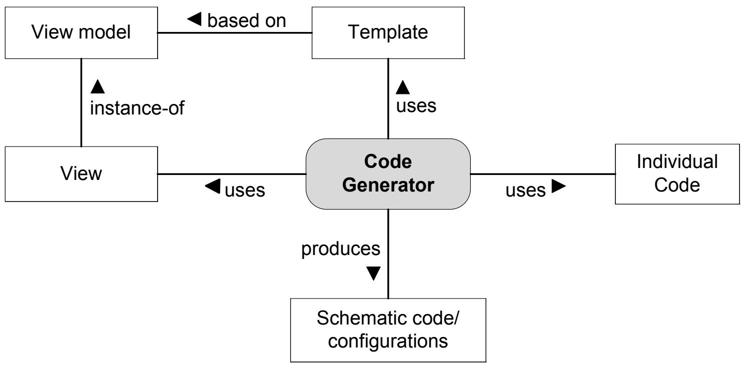
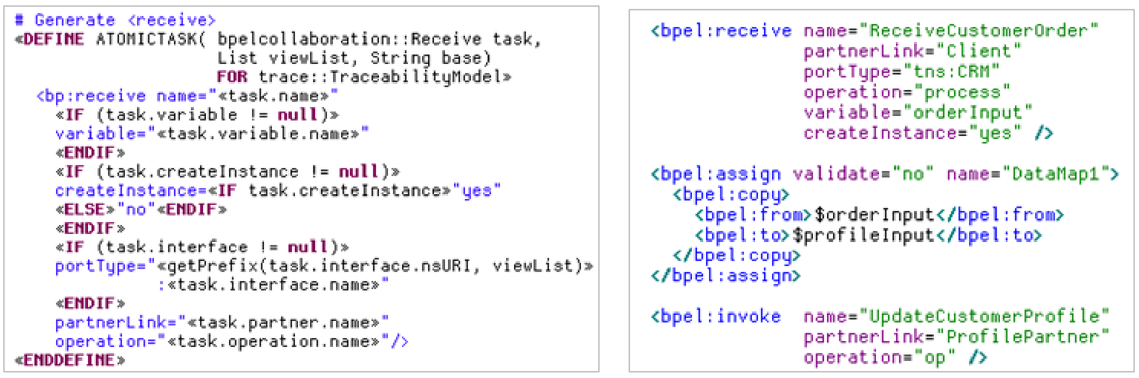


Figure 3.18: View-based model-to-code transformation (aka code generation)

Existing process views can be used by the code generator to produce schematic code and runtime configurations of the process implementation. The generated schematic code and configurations sometime need to be complemented with manually written code (also called individual code) in order to fulfill particular business functionality. A typical scenario in a model-driven paradigm, according to the best practices of process-driven development, is not to embed certain special business logics in process models in order to enhance the level of abstraction, high reusability and interoperability of the models. To serve this purpose, the code generator can combine the individual code with the schematic code to produce final process implementation that can be deployed for execution in a process engine.



```
# Generate <receive>
«DEFINE ATOMICTASK( bpelcollaboration::Receive task,
                    List viewList, String base)
                    FOR trace::TraceabilityModel»
  <bp:receive name="«task.name»"
    «IF (task.variable != null)»
    variable="«task.variable.name»"
    «ENDIF»
    «IF (task.createInstance != null)»
    createInstance=«IF task.createInstance»"yes"
    «ELSE»"no"«ENDIF»
    «ENDIF»
    «IF (task.interface != null)»
    portType="«getPrefix(task.interface.nsURI, viewList)»
              :«task.interface.name»"
    «ENDIF»
    partnerLink="«task.partner.name»"
    operation="«task.operation.name»"/>
«ENDDEFINE»
```

a) Code generation template rule in XPand language

```
<bpel:receive name="ReceiveCustomerOrder"
              partnerLink="Client"
              portType="tns:CRM"
              operation="process"
              variable="orderInput"
              createInstance="yes" />

<bpel:assign validate="no" name="DataMap1">
  <bpel:copy>
    <bpel:from>$orderInput</bpel:from>
    <bpel:to>$profileInput</bpel:to>
  </bpel:copy>
</bpel:assign>

<bpel:invoke  name="UpdateCustomerProfile"
              partnerLink="ProfilePartner"
              operation="op" />
```

b) The generated schematic recurring BPEL code

Figure 3.19: Illustration of the code generation template rules and the generated schematic BPEL code

In Figure 3.19a, we present an excerpt of our template-based code generation rules in the XPand language along with the schematic BPEL code generated by using the template rules (see Figure 3.19b). Further details of the XPand language can be found in the openArchitectureWare framework documentation[122].

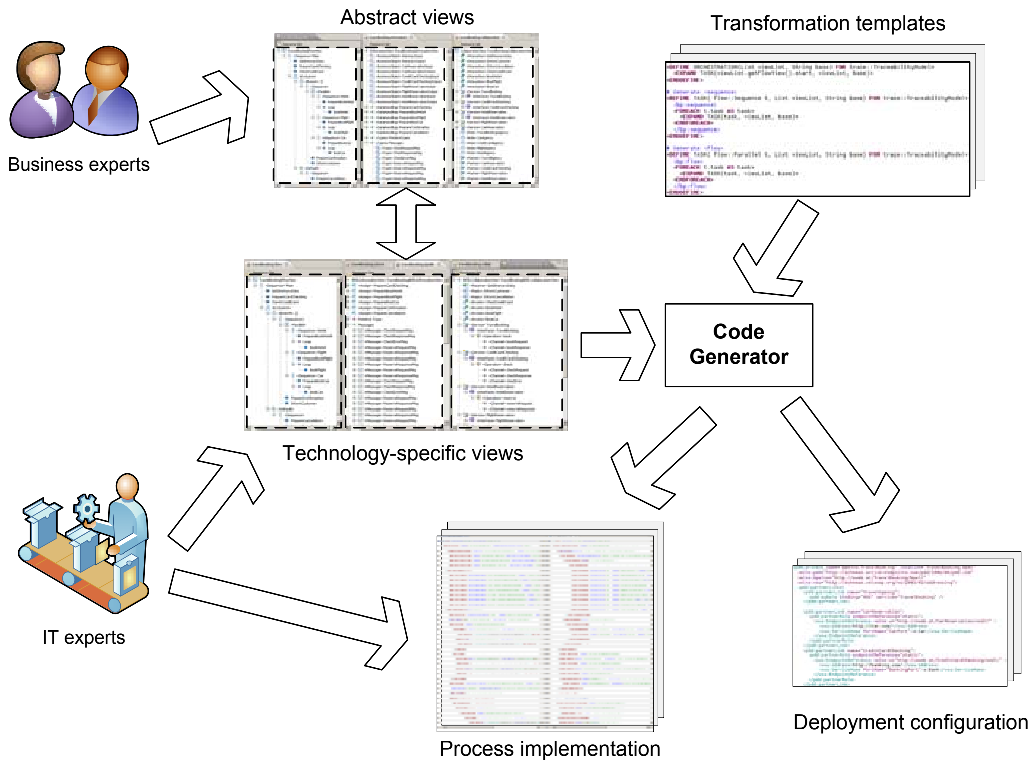


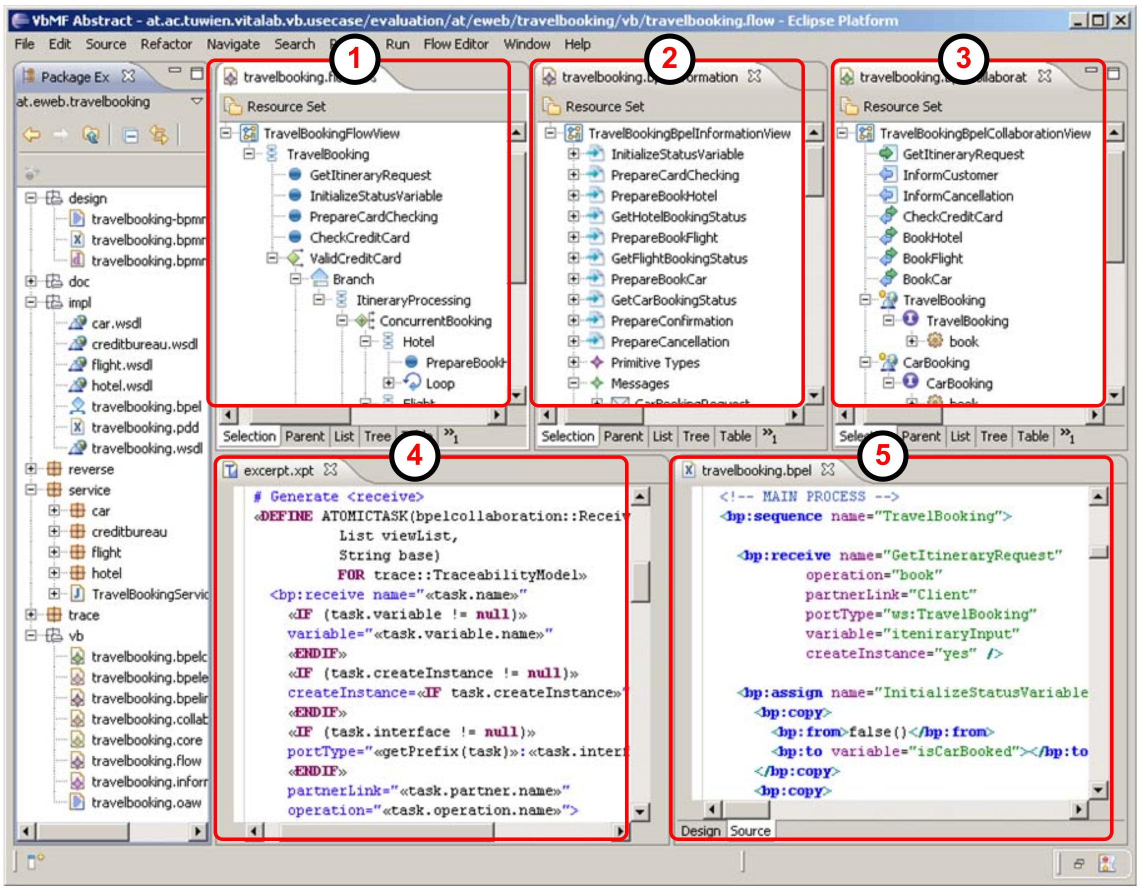
Figure 3.20: View-based, model-driven development toolchain

3.7 Tool support

The view models, components and model manipulation mechanisms mentioned above are the essential parts shaping the view-based, model-driven development toolchain shown in Figure 3.20. This toolchain supports stakeholders in modeling and implementing business process in a forward engineering approach.

In this toolchain, business experts, who are familiar with domain- and business-oriented concepts and knowledge, involve in process development by using the VbMF abstract views. The IT experts, who mostly work with technical details to implement and deploy business processes, refine the abstract views into technology-specific views. During the course of modeling or implementation, stakeholders, either business experts or IT experts, can utilize the view integration mechanism in order to achieve an appropriate view or a more thorough view of the process. After the stakeholders finish developing the corresponding VbMF views, the code generators can be used to produce schematic process code in terms of BPEL and WSDL as well as necessary process deployment configurations. The IT experts may involve in developing individual code (or so-called manually written code) that complements the generated schematic code with specific business logics or deployment configurations. The aforementioned development process can be iterated until process functionality is fully accomplished to fulfill a certain business goal.

We realized the aforementioned toolchain by facilitating the Eclipse Modeling Framework (EMF)[38]



excerpt.xpt

```
# Generate <receive>
«DEFINE ATOMICTASK(bpelcollaboration::Receiv
            List viewList,
            String base)
            FOR trace::TraceabilityModel»
  <bp:receive name="«task.name»"
    «IF (task.variable != null)»
    variable="«task.variable.name»"
    «ENDIF»
    «IF (task.createInstance != null)»
    createInstance=«IF task.createInstance»'
    «ENDIF»
    «IF (task.interface != null)»
    portType="«getPrefix(task)»:«task.interf
    «ENDIF»
    partnerLink="«task.partner.name»"
    operation="«task.operation.name»">
```

travelbooking.bpel

```
<!-- MAIN PROCESS -->
<bp:sequence name="TravelBooking">

  <bp:receive name="GetItineraryRequest"
          operation="book"
          partnerLink="Client"
          portType="ws:TravelBooking"
          variable="iteniraryInput"
          createInstance="yes" />

  <bp:assign name="InitializeStatusVariable
    <bp:copy>
      <bp:from>false()</bp:from>
      <bp:to variable="isCarBooked"></bp:to
    </bp:copy>
    <bp:copy>
```

Figure 3.21: The proof-of-concept tooling of the view-based, model-driven approach for process-driven SOAs: (1) FlowView Editor, (2) (Bpel)InformationView Editor, (3) (Bpel)CollaborationView Editor, (4) Code generation template rules, and (5) Generated process code

and openArchitectureWare MDD framework[122] (see Figure 3.21). The greatest advantage of using the Eclipse Modeling Framework is that we gain a better integration and interoperability with existing development tools developed based on EMF Ecore, a MOF-compliant meta-model, and XMI – a standard for serializing models. The View-based Modeling Framework provides different editors for manipulating views, such as the FlowView Editor, the (Bpel)CollaborationView Editor, the (Bpel)InformationView Editor, and so on. For the sake of demonstration, we mainly use the tree-based editors which are generated from our view models by the EMF generator. The template-based code generation rules are developed using the XPand language editor provided in openArchitectureWare framework[122]. Using these rules, we can automatically generate process implementations including BPEL and WSDL descriptions.

Many tasks of the modeling toolchain can be automated through the workflow engine provided by openArchitectureWare MDD framework. The openArchitectureWare workflow engine takes a declarative XML-based configuration, namely, a workflow. A workflow is typically made up of various workflow components which are POJOs derived from oAW predefined interfaces. These workflow components can be used for reading and instantiating models, checking them for OCL-based constraint violations, transforming models, and the finally, for generating code. VbMF provided a predefined workflow for loading process views, checking the conformity of views against the corresponding view models, and generating process implementations in BPEL and WSDL as well as process deployment configurations for some BPEL engines such as ActiveBPEL[1] and Apache ODE[9] (see Listing 3.1).

```
<workflow>
  <!-- Declare VbMF view models -->
  <bean class="org.eclipse.mwe.emf.StandaloneSetup" >
    <registerEcoreFile value="framework/model/vb/core.ecore"/>
    <registerEcoreFile value="framework/model/vb/flow.ecore"/>
    <registerEcoreFile value="framework/model/vb/collaboration.ecore"/>
    <registerEcoreFile value="framework/model/vb/information.ecore"/>
    <registerEcoreFile value="framework/model/vb/bpelcollaboration.ecore"/>
    <registerEcoreFile value="framework/model/vb/bpelinformation.ecore"/>
    <registerEcoreFile value="framework/model/vb/bpelevent.ecore"/>
  </bean>
  <component class="org.eclipse.mwe.emf.Reader">
    <uri value="${core}"/>
    <modelSlot value="core"/>
  </component>
  <component class="org.eclipse.mwe.emf.Reader">
    <uri value="${flow}"/>
    <modelSlot value="fv"/>
  </component>
  <component class="org.eclipse.mwe.emf.Reader">
    <uri value="${information}" />
    <modelSlot value="iv" />
  </component>
  <component class="org.eclipse.mwe.emf.Reader">
    <uri value="${collaboration}" />
    <modelSlot value="cv" />
  </component>
  <component class="org.eclipse.mwe.emf.Reader">
    <uri value="${event}" />
    <modelSlot value="ev" />
  </component>
  <!-- Generate WSBPEL 2.0 process implementation -->
  <component class="org.openarchitectureware.xpand2.Generator">
    <metaModel idRef="vbmm" />
    <expand value="framework::template::wsbpel::WSBPEL({fv, iv, cv, ev}) FOR core" />
  </component>
  <!-- Generate ActiveBPEL deyployment configuration -->
  <component class="org.openarchitectureware.xpand2.Generator">
    <metaModel idRef="vbmm" />
    <expand value="framework::template::pdd::PDD({fv, iv, cv, ev}) FOR core" />
  </component>
  <component class="org.openarchitectureware.xpand2.Generator">
    <metaModel idRef="vbmm" />
    <expand value="framework::template::wsdlCatalog::wsdlCatalog({fv, iv, cv, ev}) FOR
         core" />
  </component>
  <!-- Generate Apache ODE configuration -->
  <component class="org.openarchitectureware.xpand2.Generator">
    <metaModel idRef="vbmm" />
    <expand value="framework::template::odeDD::DD({fv, iv, cv, ev}) FOR core" />
  </component>
</workflow>
```

Listing 3.1: wsbpel.oaw: A workflow definition for generating process code and deployment configurations

3.8 Discussion

There are several existing standardization efforts on proposing process modeling languages such as BPMN[121], EPC[34,80,142], IDEF3[96], and BPDM[120], and executable languages such as BPEL[67,109], XPDL[181], and BPML[16]. The process modeling languages are often of the same abstraction level as the VbMF abstract layer whilst the executable languages are correspondent to the technology-specific layer. The commonality of these languages is to consider the business process model as a whole. They do not support the separation of the process model's concerns in order to deal with the significant complexity of process development as the number of involving services and processes increase. In addition, there are no explicit links between a certain process modeling and an executable language. As a consequence, it requires additional effort to maintain the integrity and consistency of the models, or to validate the models[100,125]. White suggests an interesting approach that uses BPMN diagrams as high-level representations of BPEL processes in order to bridge the gap between process design and implementation[182]. As this approach utilizes the existing process modeling/implementation languages, the aforementioned issues which are caused by the intrinsic characteristics of these languages, remain unsolved.

The concept of architectural views is not new. However, to the best of our knowledge, there are only a few approaches that exploit the notion of views to support business process modeling and development. Mendling et al.[102] present a view integration approach inspired by the idea of schema integration in database design. Process models based on Event-Driven Process Chains (EPCs)[34,80,142] 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 hard to apply it in order to integrate two different types of models, for instance, to merge a control flow model with a data model. Thus, the authors mainly focus on the merging of process behaviors (i.e., the control flow).

The AMFIBIA approach[10,11] presents a meta-model for formalizing different aspects of business processes and provides an open framework to integrate various formalisms through a central notion of *interface*. The major contribution in AMFIBIA is to exploit dynamic interactions of those aspects. AMFIBIA's framework 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. Using view extension and integration mechanisms in our framework, the integrity and consistency between models can be verified earlier at the model level. Nonetheless, AMFIBIA offers no support for the separation of abstraction levels and adaptation to stakeholders' interests.

The ISO Reference Model for Open Distributed Processing (RM-ODP)[74] is a standard reference model which defines a set of different *view points* such as enterprise, information, computational, engineering, and technology viewpoints. Each viewpoint has its own language and a clear semantics. These concepts, similar to those in AMFIBIA and our approach, are defined based on the principle of separation of concerns to help stakeholders thinking from different perspectives in order to manage complexity of distributed applications. The advantage of our approach compared to these approaches is that our view-based model-driven framework does not only support the separation of process concerns but also the separation of process models into different levels of abstraction, for instance, abstract and technology-specific layers.

IDEF3[96] is a scenario-based framework proposed for modeling business processes. IDEF3 provides two fundamental views: the *process-centered* and the *object-centered* view. The process-centered view provides a graphical representation that supports domain experts and analysts in describing business processes with respect to events, activities, and their relationships. The object-centered view is a mean for capturing information about objects of various kinds and their transformations during the course of a particular process. These two views are essentially similar to our *FlowView* and *InformationView* model. The other process concerns such as service and process interactions, transactions, event handling, etc., have not been considered in IDEF3.

Our work shares some concepts with the approach proposed by van der Aalst et al.[159]. In this approach, the authors develop a conceptual SOA-based architecture framework around the idea of modularization which is close to the service component architecture (SCA)[13], an industry standard for SOAs. The key concept in this approach is the notion of *components*, which is more or less equivalent to our *process* concept of the Core model, and the relationships between components. The authors emphasize the separation of activities from data elements, but neither mention the capability of extending or integrating other concerns that could be parts of a business process nor the separation of abstraction levels.

Dubray proposes WSPER[76] – an abstract SOA framework which is based on some similar concepts as those presented by val der Aalst et al.[159], such as *service*, *resource*, and *assembly*. WSPER is intentionally designed to provide an adequate formalism for SOAs and a programming language based on this formalism. In particular, WSPER offers three meta-models: a *service meta-model* for capturing services, interfaces, and operations, a *resource meta-model* for describing data and various data types including unstructured, semi-unstructured, or structured, and a *assembly meta-model* for specifying different kinds of deployment units derived from SCA[13], such as assemblies and components. WSPER mainly focuses on services and composite services, and therefore, provides neither adequate representations for high-level stakeholders such as business experts nor the ability for extending to other concerns such as transactions, event handling, etc.

There are a number of approaches using UML extensions (e.g., UML Profiles) for modeling business processes[88,146]. These approaches merely focus on high-level, domain- and business-oriented concepts and knowledge, such as customers, process goals and enterprise goals, services and products, etc., rather than the actual design and implementation of processes. Skogan et al.[147] propose another approach for process-driven modeling by using UML extensions. In this approach, a toolchain is devised to extract and formalize WSDL descriptions using UML models. Service compositions are captured by UML Activity Diagrams (AD) 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.

Schmit and Dustdar present an approach to transactional Web Services modeling[143]. Although this approach considers only one concern of a business process model, it is an initial effort on supporting separation of process views into layers and maintaining references between various layers. The transaction model described in this approach can be integrated and extended in VbMF to represent the transaction concerns of processes (see Section 3.4).

Aspect-oriented software development (AOSD) is an emerging programming paradigm that increases modularity by supporting the explicit separation of cross-cutting concerns[48,78]. Charfi et al. present an aspect-oriented approach to BPEL, AO4BPEL,[24–27] for improving the modularity of composite Web Services specifications and supporting dynamic adaptation of service compositions. Out of the advantages of AO4BPEL for supporting better modularity and dynamic adaptability[24–27], this approach only works at a low level of abstraction, i.e., executable languages such as BPEL, or VbMF technology-specific layer and offers no support for high-level process modeling languages such as BPMN, EPC, or UML AD. Cappelli et al.[22] suggest a better aspect-oriented approach that offers a meta language AOPML to modularize and represent the cross-cutting concerns of business processes. AOPML allows developers to associate various aspects, for instance, logging, auditing, etc., to a certain process model language. The authors have chosen BPMN, a process modeling language, for demonstrating how AOPML works in reality. However, we observe that the involving aspects are too technical such that the business experts, who use BPMN to design the business processes, hardly work with. This is also a common characteristic of aspect-oriented approaches: they merely work at the code level rather than supporting higher levels of abstraction.

Table 3.3 and 3.4 summarize qualitative comparisons of our view-based, model-driven approach for process-driven SOAs, VbMF, and a number of closely related work.

	Support for separation of concerns	**Support for separation of abstraction levels**	**Support for adaptation of stakeholder interests**
AMFIBIA by Kindler et al.[10,11]	AMFIBIA supports the separation of various aspects of processes including the control, organization, and information aspect. Further aspects can also be added into AMFIBIA.	Not supported	Not supported
AO4BPEL by Charfi et al.[24–27]	AO4BPEL supports the modularity of aspects of process implementation.	Not supported	Not supported
AOPML by Cappelli et al.[22]	AOPML supports the modularity of aspects of process implementation.	Not supported	Not supported
RM-ODP by ISO[74]	RM-ODP provides five generic and complementary viewpoints on the system and its environment.	Not supported	Not supported
Transactional WS by Schmit et al.[143]	This approach introduces a layered architecture including *structure*, *transaction*, *security* and *workflow*.	Not supported	Not supported
UML-based by Skogan et al.[147]	Not supported	This approach based on MDA/UML[113] and offers an extension of the UML Activity Diagram for describing Web Service compositions at high-level. These composition models are claimed to be independent of executable languages and platforms.	Not supported
View integration by Mendling et al.[102]	This approach merely offers mechanisms for merging two different views.	Not supported	Not supported
VbMF	VbMF initially supports separation of concerns by the notion of views and offers different (semi-)formalizations of process concerns in terms of view models.	VbMF initially supports separation of abstraction levels by a realization of the MDD paradigm that provides two fundamental layers: abstract and technology-specific view model.	According to particular needs, knowledge, and experiences, the stakeholders can work at either abstract layer or technology-specific layer. In addition, stakeholders can employ the view integration mechanism to see a richer view or a thorough view of the process.

Table 3.3: The comparison of related work of VbMF

	Option for extensibility	**Support for view integration**	**Support for code generation**	**Tool support**
AMFIBIA by Kindler et al.[10,11]	The authors claim that additional aspects can be added and integrated into AMFIBIA but have not elaborated how this can be done.	Not supported	Not supported	The authors stated that a prototypical aspect-oriented implementation has been developed on top of EMF. However, no implementation details are described in[10,11].
AO4BPEL by Charfi et al.[24–27]	Additional aspects, which are merely technical, can be associated with process models.	Not supported	Not supported	AO4BPEL offers an aspect deployment tool for deployment, undeployment, and listing deployed aspects.
AOPML by Cappelli et al.[22]	Additional aspects, which are merely technical, can be associated with process models.	Not supported	Not supported	Not supported
RM-ODP by ISO[74]	The conceptual framework provided in RM-ODP needs further efforts on realizing in particular software systems.	Not supported	Not supported	RM-ODP mainly offers a conceptual framework aiming at the distribution, interoperation, and portability of software systems.
Transactional WS by Schmit et al.[143]	Not supported	Not supported	Not supported	Not supported
UML-based by Skogan et al.[147]	Not supported	Not supported	Yes. Executable code for Web Service compositions, such as BPEL, can be generated from high-level composition models. As a consequence, the abstraction of these compositions models is likely similar to that of the executable languages.	Not explicitly supported
View integration by Mendling et al.[102]	Not supported	Yes. Two behavioral views can be merged according to the similarity of activity semantics.	Not supported	Not supported
VbMF	VbMF provides the extension mechanisms for expanding the framework with additional concerns or refining existing view models.	View integration is basically accomplished by using the name-based matching algorithm to produce a richer view or a thorough view of processes.	Process implementation, such as BPEL and WSDL, can be generated from technology-specific views through template-based transformation rules.	A prototypical Eclipse-based workbench based on MOF-compliant EMF Ecore and openArchitectureWare (cf. Section 3.7). XMI standard[119] is utilized for model persistence, and thereby, better support integration and interoperability of existing MOF-compliant tools.

Table 3.4: The comparison of related work of VbMF (cont'd)

3.9 Summary

In this chapter, we presented our view-based, model-driven approach for process-driven SOAs. We elaborated on how the notion of views have been exploited for dealing with the complexity of the horizontal dimension, i.e., the dimension of different concerns, of process-driven SOA development, and how the model-driven development paradigm is leveraged for the separation of abstraction levels.

We illustrated the realization of these concepts in terms of a View-based Modeling Framework for process-driven SOA development. In this framework, the notion of views is central. We developed a number of view models for (semi-)formalizing process concerns, such as the control flow, service interactions, and data processing as well as additional concerns, such as human interactions and transaction. In order to provide view models which are more appropriate and relevant to the various stakeholders' interests, we devised a model-driven stack that organizes these view models into abstract and technology-specific layer. The abstract layer includes view models that offer high-level concepts and structures such that business experts can easily understand and manipulate them to accomplish a certain business goal. The technology-specific layer consists of view models that are merely relevant to IT experts who are responsible for implementing, deploying, and maintaining the processes. This combination of the separation of concerns principle and the separation of abstraction levels offers a flexible, extensible methodology for process-driven SOA development.

Furthermore, our framework also offers different modeling and development mechanisms to stakeholders. The various editors that embody the view extension and view integration mechanisms support stakeholders in creating and manipulating view models and views. In addition, the code generator generates process implementations from views, and therefore, enhances the automation and productivity in process development. These mechanisms, along with the aforementioned concepts, shape out forward engineering toolchain for process-driven SOAs.

Chapter 4

View-based Reverse Engineering

Grasp the subject, the words will follow.
— *Cato the Elder (234BC–149BC)*

4.1 Introduction

In the context of process-driven SOAs, many companies have built up an enormous repository of existing process code in executable languages, such as BPEL/WSDL, jPDL, BPML, and FDL. There are three important issues that the companies are confronted with concerning the evolution of those processes. First, the legacy process code contains many tangled concerns, such as the control flow, service invocations, data processing, fault and event handling, transactions, and so forth. Second, the languages used to describe these processes are rather technology-specific, and therefore, the abstract, high-level representations are not explicitly available at the code level. Third, there is no explicit link between process design languages, such as BPMN, EPC, and UML Activity Diagram, and process implementation languages, such as BPEL and BPML. Hence, due to the mismatches of syntax and semantics and many other reasons, the cohesion of the high-level description and process code is likely obscured.

As a consequence, this hinders the adaptation of business processes to the evolution of both business environment and technology. The stakeholders, by their knowledge, experience, and skills, involve in the evolution by analyzing the "as-is" processes and, if necessary, re-designing or re-developing them to adapt to new requirements, policies, technologies, etc. Unfortunately, a legacy process description, due to its complexity and technology specificity, as described in the previous paragraph, is often inappropriate to particular needs and knowledge of a certain stakeholder, especially domain/business experts. Thus, such stakeholders hardly understand, analyze, maintain, or reuse fragments of existing executable process descriptions.

Our view-based approach introduced in Chapter 3 can potentially resolve these issues. Using this approach, the process models are organized into separated and formalized concerns at different abstraction levels. In addition, view extension and integration mechanisms can be used to provide better view models tailored to the stakeholders' interests. However, for budgetary reasons, developing the process views, required in our approach, from scratch is a poor and costly option. A promising alternative is an (semi)-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 of the process models are recovered from the existing code. In this way, the reverse engineering approach helps stakeholders to gain more appropriate representations of the processes such that they can efficiently get involved in process re-development and maintenance at different abstraction levels.

An appropriate reverse engineering of business processes, on the one hand, has to be able to adapt and tailor process models to stakeholder knowledge and needs. On the other hand, it is also able to enhance the integration, reusability, and interoperability of various process models. The view-based reverse engineering approach presented in this chapter aims at achieving these goals. In the subsequent sections, we present this approach in detail and realize it in terms of a reverse engineering toolchain that fits into the View-based Modeling Framework described in Chapter 3.

4.2 The view-based reverse engineering approach

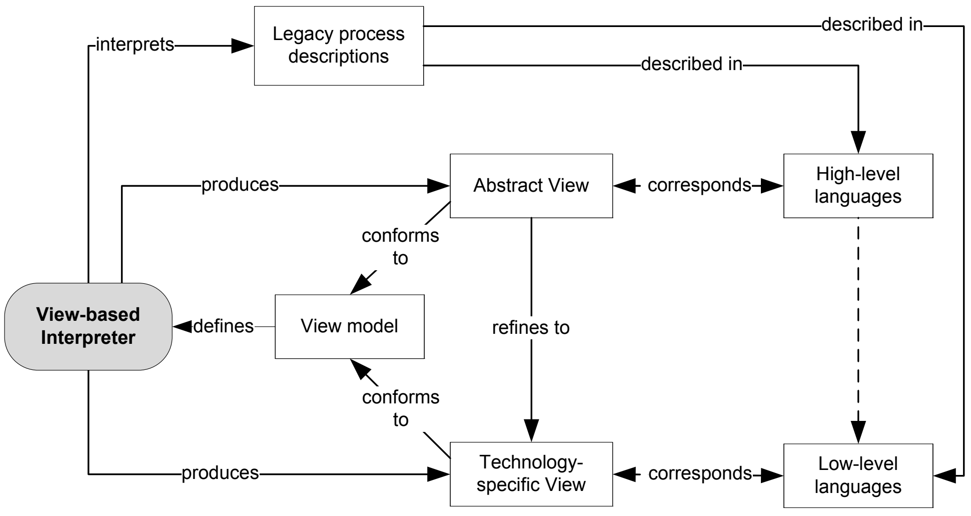


Figure 4.1: Overview of the view-based reverse engineering approach

Figure 4.1 depicts the overview of the view-based reverse engineering approach that comprises a number of view-based interpreters, such as *FlowView interpreter*, *InformationView interpreter*, and *CollaborationView interpreter*, and so on. Each interpreter is responsible for interpreting and recovering the corresponding view from existing process descriptions. Recall that in VbMF a particular view must conform to its view model. Therefore, the interpreter of a certain view is defined based on the view model to which the corresponding view conforms. For instance, a FlowView might consist of the modeling elements such as *Task*, *Parallel*, *Sequence*, *Exclusive*, *Branch*, *Default*, and *Loop* according to the FlowView model described in Figure 3.7. In order to recover the FlowView from process descriptions, the FlowView interpreter walks through the input descriptions to retrieve related information of these elements and ignores the others.

The major objective of the view-based reverse engineering approach is to recover relevant representations of different abstraction levels from existing legacy processes. In particular, our approach extracts relevant concerns of the process represented in terms of view models by using view interpreters. The resulting views can be further integrated to adapt to specific needs, knowledge, and experience of particular stakeholders. The approach is able to produce high-level, abstract views for business analysts and technology-specific views for IT experts from 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 technology-specific views as described in Figure 4.1. Additionally, the views obtained from the reverse engineering toolchain, in turn, can be manipulated using the view integration mechanisms, presented in the previous chapter, to produce richer views which fit better to stakeholders' interests, or to produce a more thorough view of the whole process. Any change on the views can be propagated by means of code generation provided in VbMF (cf. Section 3.5).

4.3 General approach for view extraction

The process descriptions comprise the specification of business functionality in a certain language, for instance, as we exemplify in this chapter, BPEL. Moreover, the process functionality also exposes through service interfaces, for instance, expressed in WSDL. To extract appropriate views from process descriptions, i.e., BPEL and WSDL specifications, we need high-level interpreters such as *FlowView interpreter*, *CollaborationView interpreter*, *InformationView interpreter* as well as technology-specific interpreters such as *BpelCollaborationView interpreter*, *BpelInformationView interpreter*, and so forth.

Our general approach to define view interpreters is based on the Partial Interpreter pattern[184]. 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, we concentrate on specific view models. The approach based on Partial Interpreter enables us to define modular, pluggable view 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 model elements. That is, the mapping specification contains all elements of a particular view 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 view models. Hence, they can be reused for many concrete view models.

In the subsequent sections, we present the details of the realization of the view-based reverse engineering in which abstract views such as FlowView and CollaborationView as well as technology-specific views such as BpelCollaborationView can be (semi-)automatically recovered from the existing process code.

4.4 View-based reverse engineering approach for process-driven SOAs

In this section, we clarify our view-based reverse engineering approach by investigating an exemplary process-driven SOA technology: BPEL and WSDL. BPEL/WSDL is chosen for exemplification because these are widely adopted languages for process implementation and service description 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 relevant view models and Partial Interpreter specifications.

4.4.1 Recovering abstract, high-level representations

4.4.1.1 Recovering the FlowView

In order to recover the FlowView from BPEL code, the FlowView interpreter, which is based on the FlowView model (see Figure 3.7), walks through the process description in BPEL and collects the information of atomic and structured activities. Then, it creates the elements in the FlowView and assigns their attributes with relevant values as specified in Table 4.1.

According to the specification of the FlowView model (see Figure 3.7), a FlowView includes the elements representing the orchestration of process tasks. Therefore, the FlowView interpreter only considers this aspect of BPEL processes and ignores the rests.

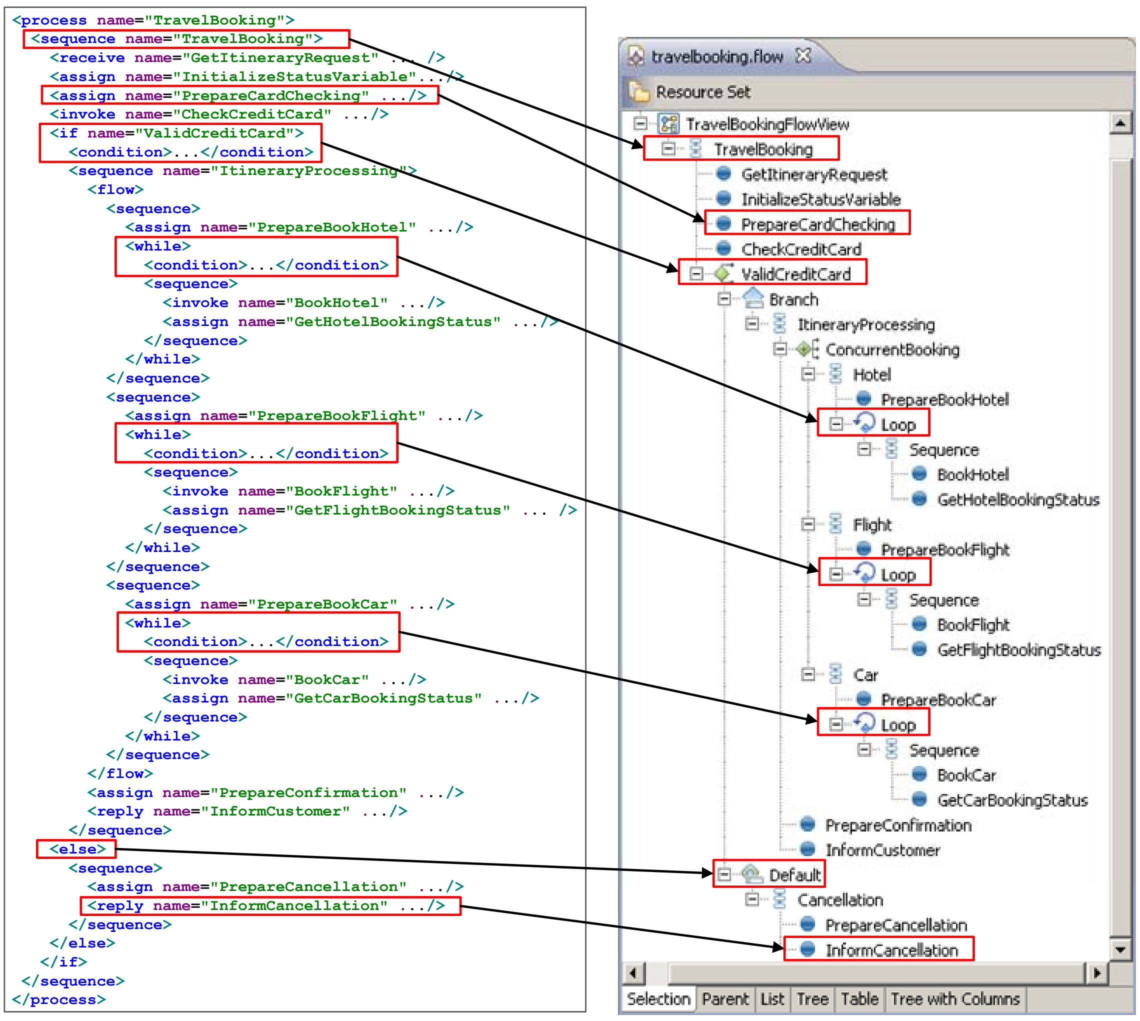
BPEL element	FlowView element
`<invoke|receive|reply|assign name="...">`	`fv::AtomicTask.setName()`
`<sequence name="...">`	`fv::Sequence/setName()`
`<flow name="...">`	`fv::Parallel/setName()`
`<if name="..."><condition>"..."</condition>`	`fv::Exclusive/setName()`
`<elseif><condition>"..."</condition>`	`fv::Branch/setCondition()`
`<else>`	`fv::Default`
`<while name="..." condition="...">`	`fv::Loop/setName()/setCondition()`

Table 4.1: Recovering the FlowView model from BPEL descriptions (the notion `fv::` is used to indicate the FlowView's namespace)

In particular, the hierarchy and the execution order of a BPEL process are essentially defined using the control structures, such as `<sequence>` – a step-by-step execution of activities, `<flow>` – a concurrent execution of activities, `<if>-<elseif>-<else>` – a conditionally exclusive branch, and `<while>` – a condition iteration of a certain task. These control structures can be nested and combined in arbitrary manners to represent various complex control flows in BPEL processes. With respect to their syntax and semantics specified in the BPEL specification[109], those structures are straightforward mapped into the FlowView model (see Table 4.1).

In addition, a BPEL process might have one or many primitive activities, such as `<invoke>` – a service invocation, `<receive>` – waiting for a message from partners, `<reply>` – sending back a response to a partner, and `<assign>` – assigning values to variables and properties. Those activities are mapped into the corresponding `AtomicTask` elements of the FlowView. By abstracting these different activities away from their concrete meanings such as "*receiving an XML message (i.e., an order) from a partner who plays a role of Customer (i.e., a customer)*", "*invoking a service or process via the operation x of the portType p provided by the partner who plays a role of a banking institution*", etc., the resulting FlowView is merely presented to the stakeholders in form of a "plain" business logic (i.e., an orchestration of process activities) accomplishing particular business goals. In this way, stakeholders, especially business experts, can better understand, analyze, and modify the FlowView to adapt to new requirements or changes.

Figure 4.2 depicts the recovering of the FlowView from Travel Booking BPEL code described in Listing 2.1 by using the FlowView interpreter. As the FlowView interpreter merely extracts the BPEL control structures such as `<sequence>`, `<flow>`, `<if>-<elseif>-<else>`, and `<while>` and the



```
<process name="TravelBooking">
  <sequence name="TravelBooking">
    <receive name="GetItineraryRequest" .../>
    <assign name="InitializeStatusVariable".../>
    <assign name="PrepareCardChecking" .../>
    <invoke name="CheckCreditCard" .../>
    <if name="ValidCreditCard">
      <condition>...</condition>
      <sequence name="ItineraryProcessing">
        <flow>
          <sequence>
            <assign name="PrepareBookHotel" .../>
            <while>
              <condition>...</condition>
              <sequence>
                <invoke name="BookHotel" .../>
                <assign name="GetHotelBookingStatus" .../>
              </sequence>
            </while>
          </sequence>
          <sequence>
            <assign name="PrepareBookFlight" .../>
            <while>
              <condition>...</condition>
              <sequence>
                <invoke name="BookFlight" .../>
                <assign name="GetFlightBookingStatus" ... />
              </sequence>
            </while>
          </sequence>
          <sequence>
            <assign name="PrepareBookCar" .../>
            <while>
              <condition>...</condition>
              <sequence>
                <invoke name="BookCar" .../>
                <assign name="GetCarBookingStatus" .../>
              </sequence>
            </while>
          </sequence>
        </flow>
        <assign name="PrepareConfirmation" .../>
        <reply name="InformCustomer" .../>
      </sequence>
    <else>
      <sequence>
        <assign name="PrepareCancellation" .../>
        <reply name="InformCancellation" .../>
      </sequence>
    </else>
   </if>
 </sequence>
</process>
```

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

a) Travel Booking process code in BPEL

b) The extracted TravelBooking FlowView

Figure 4.2: Recovering the FlowView from Travel Booking BPEL code

process activities such as `<assign>`, `<invoke>`, `<receive>`, and `<reply>`, and corresponding activities' names, the remaining information will be omitted in Figure 4.2.

4.4.1.2 Recovering the CollaborationView

The CollaborationView interpreter is realized using the same approach as the FlowView interpreter. However, the CollaborationView comprises not only the elements from the process descriptions (i.e., BPEL files) but also from the service interface descriptions of the processes (i.e., WSDL files). Hence, first of all, the interpreter has to collect all information of service interfaces, messages, roles, and partners from WSDL files. Then, the interpreter creates corresponding elements and relationships in the CollaborationView according the mapping rules given in Table 4.2. Next, the interpreter walks through the BPEL code to extract information from the collaborative activities. Each of the BPEL collaborative activities, such as `<invoke>`, `<receive>`, or `<reply>`, shall appear on the resulting

CollaborationView with the same name as in the FlowView. However, these activities shall contain additional collaborative attributes as depicted in Table 4.3.

WSDL element	CollaborationView element
`<definition>`	`cv::Service`
`<message name="...">`	`cv::Message/setName()`
`<portType name="...">`	`cv::Interface/setName()`
`<operation name="...">`	`cv::Operation/setName()`
`<input name="..." message="...">`	`cv::Channel`
`<output name="..." message="...">`	`cv::Channel`
`<partnerLinkType name="...">`	`cv::Partner/setName()`
`<Role name="...">`	`cv::Role/setName()`
`<service name="...">`	`cv::Service.setName()`

Table 4.2: Recovering the CollaborationView from WSDL descriptions (the notion `scv::` is used to indicate the CollaborationView's namespace.)

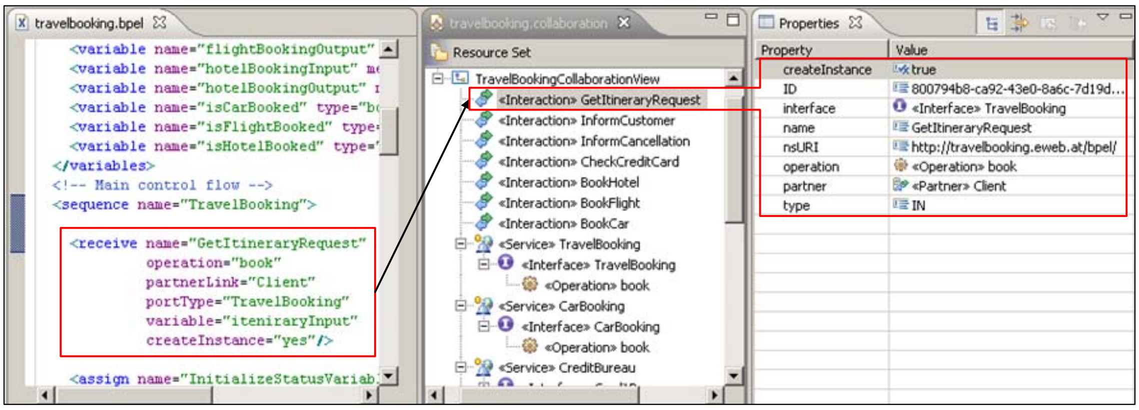
BPEL element	CollaborationView	BpelCollaborationView
`<invoke`	`cv::Interaction(INOUT)`	`bcv::Invoke`
`name="..."`	`+setName()`	`+setName()`
`partnerLink="..."`	`+setPartner()`	`setPartner()`
`portType="..."`	`+setInterface()`	`+setInterface()`
`correlation set="..."`		`+createCorrelation()`
`inputVariable="..."`		`+setInput()`
`outputVariable="..."`		`+setOutput()`
`operation="...">`		`+setOperation()`
`<receive`	`cv::Interaction(IN)`	`bcv::Invoke`
`name="..."`	`+setName()`	`+setName()`
`partnerLink="..."`	`+setPartner()`	`+setPartner()`
`portType="..."`	`+setInterface()`	`+setInterface()`
`correlation set="..."`		`+createCorrelation()`
`variable="..."`		`+setVariable()`
`operation="..."`		`+setOperation()`
`createInstance="...">`	`+setCreateInstance()`	`+setCreateInstance()`
`<reply`	`cv::Interaction(OUT)`	`bcv::Invoke`
`name="..."`	`+setName()`	`+setName()`
`partnerLink="..."`	`+setPartner()`	`+setPartner()`
`portType="..."`	`+setInterface()`	`+setInterface()`
`correlation set="..."`		`+createCorrelation()`
`variable="..."`		`+setVariable()`
`operation="...">`		`+setOperation()`
`<correlationSet`		`bcv::CorrelationSet`
`name="..."`		`+setName()`
`properties="...">`		`+setProperty()`
`<correlationSets>`		`bcv::CorrelationSets`
`<property`		`bcv::Property`
`name="..."`		`+setName()`
`type="...">`		`+setType()`
`<propertyAlias`		`bcv::PropertyAlias`
`propertyName="..."`		`+setProperty()`
`messageType="..."`		`+setMessageType()`
`part="..."`		`+setPart()`
`query="...">`		`+setQuery()`
`<partnerLink`	`cv::Role() &`	`(inherits from`
`myRole="..."`	`cv.setMyRole() cv::Role() &`	`the parents -- the`
`partnerRole="...">`	`cv::Partner.setPartnerRole()`	`CollaborationView`

(Continued on next page)

BPEL element	CollaborationView	BpelCollaborationView

Table 4.3: Recovering the CollaborationView and BpelCollaborationView from BPEL descriptions (the notions **cv::** and **bvc::** are used to indicate the namespaces of the CollaborationView and BpelCollaborationView, respectively)

In addition to creating the corresponding elements of the CollaborationView, the interpreter uses the information collected in the former step to establish the necessary relationships between these elements. For instance, the relationship between *cv::Interaction* and *cv::Partner* elements is derived from the association between the communication activities (e.g., `<invoke>`, `<receive>`, `<reply>`) and the corresponding `<partnerLink>`, or the relationship between *cv::Partner* and *cv::Role* is derived from the association among the `<partnerLinkType>` and `<role>` elements in the WSDL descriptions and the `<partnerLink>` elements in the BPEL code. In Figure 4.3, we illustrate the recovering of the CollaborationView from BPEL code of the Travel Booking process (see Listing2.1) by using the CollaborationView interpreter.



```
    <variable name="flightBookingOutput"
    <variable name="hotelBookingInput" m
    <variable name="hotelBookingOutput" 
    <variable name="isCarBooked" type="b
    <variable name="isFlightBooked" type
    <variable name="isHotelBooked" type=
  </variables>
  <!-- Main control flow -->
  <sequence name="TravelBooking">

    <receive name="GetItineraryRequest"
             operation="book"
             partnerLink="Client"
             portType="TravelBooking"
             variable="iteniraryInput"
             createInstance="yes"/>

    <assign name="InitializeStatusVariab
```

```
TravelBookingCollaborationView
  «Interaction» GetItineraryRequest
  «Interaction» InformCustomer
  «Interaction» InformCancellation
  «Interaction» CheckCreditCard
  «Interaction» BookHotel
  «Interaction» BookFlight
  «Interaction» BookCar
  «Service» TravelBooking
    «Interface» TravelBooking
      «Operation» book
  «Service» CarBooking
    «Interface» CarBooking
      «Operation» book
  «Service» CreditBureau
```

Property	Value
createInstance	true
ID	800794b8-ca92-43e0-8a6c-7d19d...
interface	«Interface» TravelBooking
name	GetItineraryRequest
nsURI	http://travelbooking.eweb.at/bpel/
operation	«Operation» book
partner	«Partner» Client
type	IN

a) Travel Booking BPEL code b) The extracted CollaborationView c) The Properties View

Figure 4.3: Recovering the CollaborationView from Travel Booking BPEL code

4.4.2 Recovering low-level, technology-specific representations

So far the abstract, high-level representations of process models such as FlowView and CollaborationView are recovered from existing BPEL and WSDL descriptions. Out of these abstract views, the technology-specific views can be derived using the refinement mechanism with the support of IT experts. However, existing process implementations, including BPEL and WSDL descriptions, embody low-level, technical representations of the processes as well. Therefore, instead of deriving the technology-specific views from the abstract counterparts, which is error-prone and time-consuming, we leverage the view-based reverse engineering approach to parse the process implementation and recover the embodied low-level, technology-specific representations of the process. Moreover, the low-level view interpreters are developed by reusing the realization of the abstract interpreters and enriched with additional feature according to the corresponding view models. This is likely the same manner that a technology-specific view inherits and enriches an abstract counterpart. This way, the integrity and consistency of an abstract view and the corresponding technology-specific views are maintained because they are both recovered from the same inputs. Nonetheless, maintaining of the consistency requires additional supports by the tooling and is one of our future works.

We illustrate the recovering of low-level, technology-specific views using the view-based reverse engineering approach in Table 4.3. The CollaborationView is a high-level representation of the BpelCollaborationView, which is at a lower level of abstraction. The BpelCollaborationView enriches the CollaborationView with either newly added elements or inherited elements with additional properties. In Table 4.3, the right column shows the elements of the BpelCollaborationView recovered from the BPEL descriptions. Many of those elements are partially inherited the information of the relevant elements of the CollaborationView shown in the middle column of Table 4.3. Thus, we can inherit the corresponding parsing elements from the CollaborationView interpreter and only develop parsers for additional features of the BpelCollaborationView.

4.5 Tool support

The view-based reverse engineering approach described in the previous sections has been realized to form a toolchain that supports stakeholders in extracting appropriate representations of process descriptions in terms of high-level or low-level, technology-specific view models (see Figure 4.4).

In this reverse engineering toolchain, existing process implementations in terms of BPEL and WSDL descriptions, and process deployment configurations in XML-based representations, are taken as inputs for the various view-based interpreters, respectively. The outcomes of these view-based interpreters are either VbMF abstract views – the high-level representations of processes or technology-specific views – the low-level representations of processes. The stakeholders, according to their particular needs, knowledge, and experience, can work with their representations of choice. For instance, business experts will mainly involve in the abstract, high-level process views whilst IT experts are more interested in the technology-specific views.

We have developed the view-based interpreters using XPath technology[168] to extract XML-based information from process implementations such as BPEL and WSDL and build up corresponding in-memory views. After that, these in-memory views are then serialized and stored in model repositories in terms of XMI-compliant descriptions[119] by using model serialization techniques provided in the Eclipse Modeling Framework[38]. These process views, in terms of XMI-compliant descriptions, can be (re-)used later in VbMF.

4.6 Discussion

Our work presented in this chapter is a *reverse engineering approach*[30], based on the separation of concerns by using the notion of views, and the separation of abstraction levels based on the MDD paradigm. This approach contributes a reverse engineering toolchain that complements the forward engineering toolchain provided in VbMF (cf. Section 3.7). Therefore, VbMF now can support *reengineering*[6,12]. The reverse engineering toolchain provides means for re-structuring and

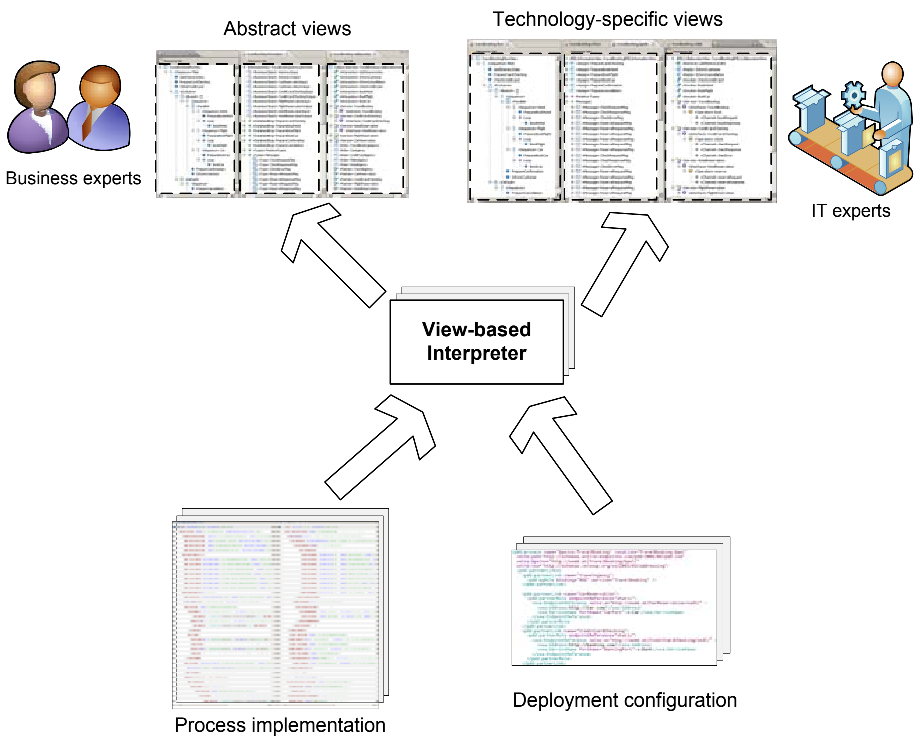


Figure 4.4: View-based reverse engineering toolchain for process-driven SOAs

modification whilst the forward engineering toolchain are provided in order to re-generate modified software systems.

In the context of reverse engineering, view-based approaches are an emerging area of interest. For instance, the approaches reported by Chebbi et al.[28], Chiu et al.[31], and Schulz and Orlowska[144] 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.[17] present an approach to process visualization using personalized views and a number of operations to customize the views. Zou et al.[191] 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.[77] 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 of which some provide high-level descriptions, examples are: BPMN[121], EPC[80] and abstract BPEL in WS-BPEL 2.0[109]. These languages provide high-level representations of processes and hide the technical details. These representations are relevant to the business analysts. However, these languages still comprise numerous tangled concerns such as the control flow, data handling, service invocations, transactions, to name but a few. As a consequence, the representations provided by these languages are hardly tailorable and customizable.

A number of recent research approaches concentrate on the transformations between process modeling and development languages. For instance, Mendling et al.[103] discuss the transformation of BPEL to EPCs. Ziemann et al.[188] present an approach to model BPEL processes using EPC-based models. Recker et al.[134] translate between BPMN and BPEL. Mendling et al.[101] 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 handling, transactions, and so forth.

All the above-mentioned approaches and standards have difficulties in handling the complexity of process models: Because the business process contains numerous tangled concerns, the complexity of process model increases as the number of process elements, such as message exchanges, service invocations, data handling tasks, transactions, etc. grows. Thus, on the one hand, these approaches are less efficient than our approach in dealing with huge existing process repositories, developed in other languages or dialects, or integrating arbitrary process modeling tools. On the other hand, these approaches have not offered the adaptability to different stakeholders' interests, the better reusability of legacy process code, as well as the interoperability process modeling and development languages. Our view-based reverse engineering approach poses a number of distinct characteristics, which are:

- *Enhancing adaptability in process development*: The adaptability of process representations to different requirements of stakeholders is supported by the *separation of process concerns* and the *separation of abstraction levels*, two major contributions of our view-based, model-driven approach. Moreover, it is enhanced by two methods developed in VbMF: *view extension* and *view integration* mechanisms (cf. Section 3.5). The view extension mechanisms (cf. Section 3.5) allows stakeholders to enrich existing view models with additional elements and extra attributes for the existing elements of the original view models. In 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 appropriate to the stakeholders's needs. Our view-based reverse engineering approach presented in this chapter complements these mechanisms by introducing respective interpreters for high-level and low-level view models to extract the corresponding views from the existing process code. Furthermore, process views extracted by the view-based reverse engineering are potential inputs for view integration (cf. Section 3.5) to produce new richer views by merging existing views to adapt to particular needs and knowledge of the stakeholders.

- *Enhancing interoperability of process development languages*: Interoperability 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. The huge divergence of process modeling languages raises a critical issue that deteriorates the interoperability of different process representations. Our view-based reverse engineering approach leverages the notion of views in a reverse engineering toolchain that recovers views from existing process code.
  The high-level views are platform-independent models used to capture abstract concepts of process descriptions. In other words, these views are at the same abstraction levels as existing process modeling languages, such as EPC, BPMN, and UML Activity Diagram. Thus, the reverse engineering methodology presented in this chapter can be extended to map process representations in these languages into VbMF high-level views. This has a number of advantages. Firstly, it uses one kind of modeling approach for all types of views. Secondly, it can potentially avoid semantic mismatch or transformation between modeling concepts. Nonetheless, this approach has the disadvantage that existing modeling representations (say realized in EPCs, BPMN or UML Activity Diagrams) would have to be mapped to the high-level view 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 FlowView – the central notion of process-driven approaches – represents basic control flow patterns that almost exist in all existing modeling language; the CollaborationView describes generic interactions that typically occur between a process and its partners; and the InformationView abstracts the various data objects and data processing. Hence, an adequate view interpreter for a particular language can distill these views from the process descriptions in that language in the same manner as those presented in the previous sections. Alternatively, our approach can probably be extended with a new view model, such as a control flow view of EPC, BPMN, or UML Activity Diagram.
  A high-level view then can be refined to a lower level view which covers the specifics of a particular technology. For example, the CollaborationView can be refined to the BpelCollaborationView. Using the same approach, we can define appropriate view models and interpreters for recovering corresponding views from process implementations in low-level, executable languages such as BPEL. As a consequence, a stakeholder can work on an appropriate view or

can examine a certain combination of several views instead of confronting with various kinds of process descriptions or digging into the implementation code in executable languages.

- *Enhancing reusability of process descriptions*: Reusability is the degree of using the existing software artifacts instead of developing new ones from scratch. Therefore, enhancing reusability leads to the increasing of productivity, IT cost reduction, and so on. Furthermore, software quality is improved because existing proven techniques and knowledge can be reused to develop new software or systems.
  SOAs are potentially enabling software reuse by specifying software functionality using standard service interfaces where service consumers and providers can communicate regardless of the underlying programming language and platform. Process-driven SOAs provide an efficient way of coordinating various services using processes to accomplish a specific business goal. Unfortunately, existing languages pose some intrinsic features that inhibit the reusability in process modeling and development. These languages consider business processes in a whole, and therefore, provide stakeholders with the process representations containing many tangled concerns. In order to reuse a certain excerpt of a process, the stakeholder has to go across various concerns, some of which are even not suitable for the stakeholder's expertise and skills. Moreover, existing process languages and tools do not support cross referencing between process languages, for instance, between UML Activity Diagram and EPC and BPMN and BPEL, or between two BPEL descriptions and so on. As a consequence, the reusability of process descriptions, either in high-level or low-level languages, are merely achieved by using the "copy-and-paste" approach, which is very tedious and error-prone.
  Using VbMF, process descriptions are separated into different views representing distinct process concerns. The integration of views are performed via name-based matching (see Section 3.5). Therefore, stakeholders can reuse particular existing VbMF elements or views by referencing them accordingly. Our view-based reverse engineering approach indirectly enhances the reusability of process descriptions by mapping existing process descriptions into corresponding views which can be re-used in VbMF later.

4.7 Summary

The view-based reverse engineering approach presented in this chapter can help the various stakeholders of a process-driven SOA to overcome two important issues. *Firstly*, it exploits the notion of views to deal with the complexity of existing process repositories and to adapt the process representations to the stakeholders' interests. *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 for developing the others. This has been achieved by developing a novel concept for a reverse engineering toolchain, based on partial interpreters and view models, and by seamlessly integrating this reverse engineering toolchain into our View-based Modeling Framework, which also supports means for forward engineering, such as view integration, view extension, and code generation. The view-based reverse engineering approach enables the reuse of existing process code, e.g. implemented in BPEL and WSDL, in the View-based Modeling Framework.

Chapter 5

View-based, Model-driven Traceability

“*When we try to pick out anything by itself we find that it is bound fast by a thousand invisible cords that cannot be broken, to everything in the universe.*”

— *John Muir (1838–1914)*

5.1 Introduction

In Chapter 2, we introduce the process-driven, service-oriented architecture (SOA) for realizing business processes. In process-driven SOAs, business processes are often designed in highly abstract and primarily notational modeling languages such as BPMN, EPC, or UML Activity Diagrams. Process designs are suitable for business experts to represent domain- and business-oriented concepts and functionality but mostly non-executable because many technical details are missing. Thus, IT experts necessarily need to be involved in the process development to transform the process designs into executable specifications. For example, IT experts can translate abstract, high-level concepts of process designs into concrete, fine-grained elements in executable process languages, such as BPEL, and specify the process interfaces in Web Service Description Language (WSDL). Additional deployment configurations might also need to be defined in order to successfully deploy and execute the implemented processes.

Understanding trace dependencies between process design and implementation is vital for change impact analysis, change propagation, documentation, and many other activities[148]. Unfortunately, artifacts created during the process development life cycle likely end up being disconnected from each other. This impairs the traceability of artifacts. We identify the following important factors that complicate the establishing and maintenance of trace dependencies :

- There are no explicit links between process design and implementation languages. This lack of dependency links is caused by not only syntactic and semantic differences but also the difference of granularity as these languages describe a process at various levels of abstraction.

- A substantial complexity is caused by tangled process concerns. Either the process design or implementation comprises numerous tangled concerns such as the control flow, data processing, service invocations, transactions, fault and event handling, etc. 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, data exchanges, and cross-concern references, and therefore, multiplies the difficulty of analyzing and understanding the trace dependencies.

- There is a lack of adequate tool support to create and maintain trace dependencies between process designs and implementations.

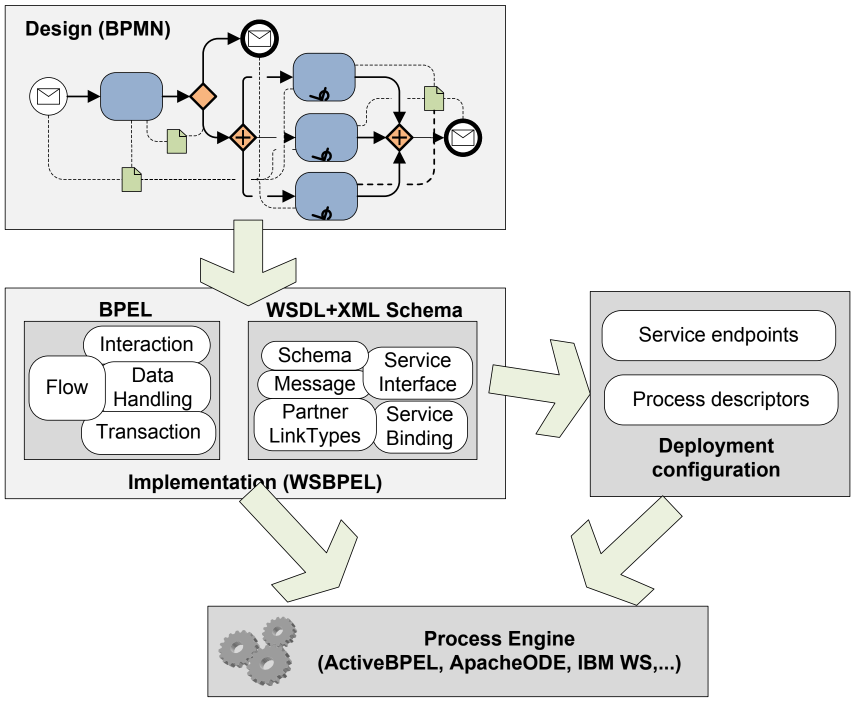


Figure 5.1: The Travel Booking process development

To illustrate the aforementioned factors we use the well-known Travel Booking process (see Figure 2.5, Listing 2.1, and Listing 2.2). Figure 5.1 shows the process development scenario from design to implementation and deployment. We summarize the statistics of the complexity in terms of the number of elements as well as their dependencies in Table 5.1. Even though the syntactic and semantic differences are omitted in Figure 5.1, the elements represented in executable process languages (here: BPEL and WSDL) are more concrete and of much finer granularity than the design counterparts (here expressed in BPMN). Practically, abstract, high-level model elements are often described or implemented by one or many technology-specific elements. For instance, a *Data Object* in BPMN is often represented by the corresponding variable in BPEL and the message type from WSDL or the XML Schema type. In addition, some artifacts which are necessary for describing specific features in process implementation or for successfully deploying the process have no corresponding elements

Design (BPMN)		Implementation (BPEL)		Deployment	
Task	7	BPEL activity	13	Partner Reference	5
		Correlation	7	Endpoint Reference	4
Control structure	6	Flow control	12	Service Reference	5
Control edge	17				
Data object	10	BPEL variable	10		
Data association	13	Message	11		
		XML schema type	11		
		Data handling	30		
Partner (pool)	5	PartnerLink	5		
Partner association	5	PartnerLinkType	5		
		PortType	5		
		Role	5		
		Binding	4		
		Service	4		
Total element	**63**	Total element	**122**	Total element	**14**
Dependency	**37**	Dependency	**145**	Dependency	**20**
Cross-concern dep.	**20**	Cross-concern dep.	**55**	Cross-concern dep.	**20**

Table 5.1: The complexity and dependency statistics of the Travel Booking process

in the process design. For instance, there are no corresponding design concepts or elements for the correlation of service invocations in BPEL, service bindings and service endpoints in WSDL, to name but a few. Existing process development approaches or tools merely support the stakeholders in importing, parsing, validating, and referencing elements between languages of the same level of abstraction, for instance, between BPEL, WSDL, and XML Schema, but have no support for cross references between process designs and implementations.

The complexity caused by numerous tangled process concerns such as the control flow, service and process interactions, data handling, transactions, and so forth, hinders the understanding and analyzing of trace dependencies. Table 5.1 shows the statistics of the cross-concern dependencies of process design (20/37), process implementation (55/145), and deployment configuration (20/20). These numbers mean: In order to thoroughly understand or analyze a certain concept of either a process design or an implementation, the developer has to go across numerous dependencies between various concerns, some of which are even not suitable for the developer's expertise and skills.

We present a view-based, model-driven traceability approach that supports stakeholders in (semi-)-automatically creating and maintaining traceability between process designs and implementations and/or deployment configurations. In the context of this chapter, BPMN[121], a standard for business process modeling, is used as a representative example of a process design language, whilst BPEL[109] and WSDL[170], which are very popular process/service modeling descriptions used by numerous companies today, are used as a representative examples for languages for implementing executable processes. Although establishing trace dependencies alone is not sufficient for tasks like change impact analysis or change propagation, it crucially lays the foundation for any such tasks. In this sense, our approach presented in this chapter is the initial effort that overcomes the aforementioned challenges to support (semi-)automatically eliciting and (semi-)formalizing trace dependencies among model artifacts in model-driven development (MDD) at different levels of granularity and abstraction. The (semi-)formalization of the trace dependencies is one of the features needed for the interoperability of tools utilizing them.

In our approach, we exploit the notion of views and the model-driven stack introduced in Chapter 3 in order to separate process representations (e.g., process designs or implementations) into different (semi-)formalized view models. In this way, stakeholders can be provided with tailored perspectives by view integration mechanisms (cf. Section 3.5) according to their particular needs, knowledge and experience. This is a significant step toward the support of adapting process representations and trace relationships to particular stakeholder interests. Additionally, view models are also organized into appropriate levels of abstraction: high-level, abstract views are suitable for business experts whilst low-level, technology-specific views are mostly used by IT experts. Given these levels of abstraction, process designs are adequately aligned with the abstract view models, and the implementation counterparts are lined up with the technology-specific view models. This can be done in a (semi-)-automatic manner using the view-based reverse engineering approach described in Chapter 4. Such mappings produce trace dependencies between designs and the view models, and between the view models and the source code that implements the processes. These dependencies are parts of the traceability meta-model which is the key component of our traceability approach. Moreover, the traceability meta-model also supports stakeholders in capturing intrinsic dependencies between view models and view elements.

In Section 5.2, we presents our view-based, model-driven traceability approach along with the details of the traceability meta-model. A CRM Fulfillment process from an industrial case study is exemplified to illustrate our traceability approach and the realization of the approach in Section 5.3. Section 5.4 is dedicated for discussing the related work.

5.2 View-based, model-driven traceability framework

5.2.1 Fundamentals concepts

In the previous section we introduce the view-based modeling framework (VbMF) which supports stakeholders in modeling and developing processes using various perspectives which are tailored for their particular needs, knowledge, and skills at different levels of abstraction. We propose in this section our view-based, model-driven traceability approach (VbTrace) in terms of a traceability

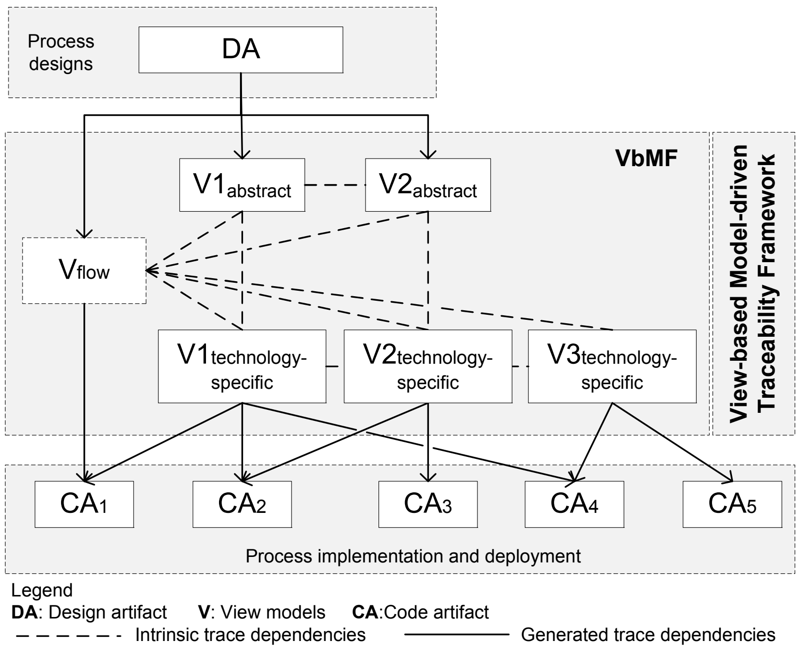


Figure 5.2: Overview of the view-based, model-driven traceability approach

framework which is an additional dimension to the model-driven stack of VbMF (see Figure 5.2).

VbTrace supports stakeholders in establishing and maintaining trace dependencies between the process designs and implementations (i.e., process code artifacts) via VbMF. The trace dependencies between process design and abstract, high-level views and those between low-level, technology-specific views and code artifacts can be automatically derived during the mappings of process designs and implementations into VbMF views using an extended version of the view-based reverse engineering approach presented in Chapter 3. These trace dependencies are represented by the solid arrows in Figure 5.2. The relationships between a view and its elements are intrinsic whilst the relationships between different views are established by using the name-based matching mechanism for integrating views (cf. Section 3.5). These relationships are indicated in Figure 5.2 by dashed lines because they are merely derived from the view models and mechanisms provided by VbMF. Therefore, we will concentrate more on the former kind of trace dependencies, i.e., the trace dependencies between process designs and implementations and view models. Nonetheless, the case study in Section 5.3 will illustrate a complete consolidation of the aforementioned kinds of trace dependencies as a whole. In the subsequent sections, we present the view-based traceability meta-model that is a (semi-)formalization of trace dependencies between process development artifacts. Based on the traceability meta-model, we extend and use the components and mechanisms provided by VbMF to shape a view-based, model-driven traceability framework that supports stakeholders in (semi-)automatically establishing and maintaining the corresponding trace dependencies .

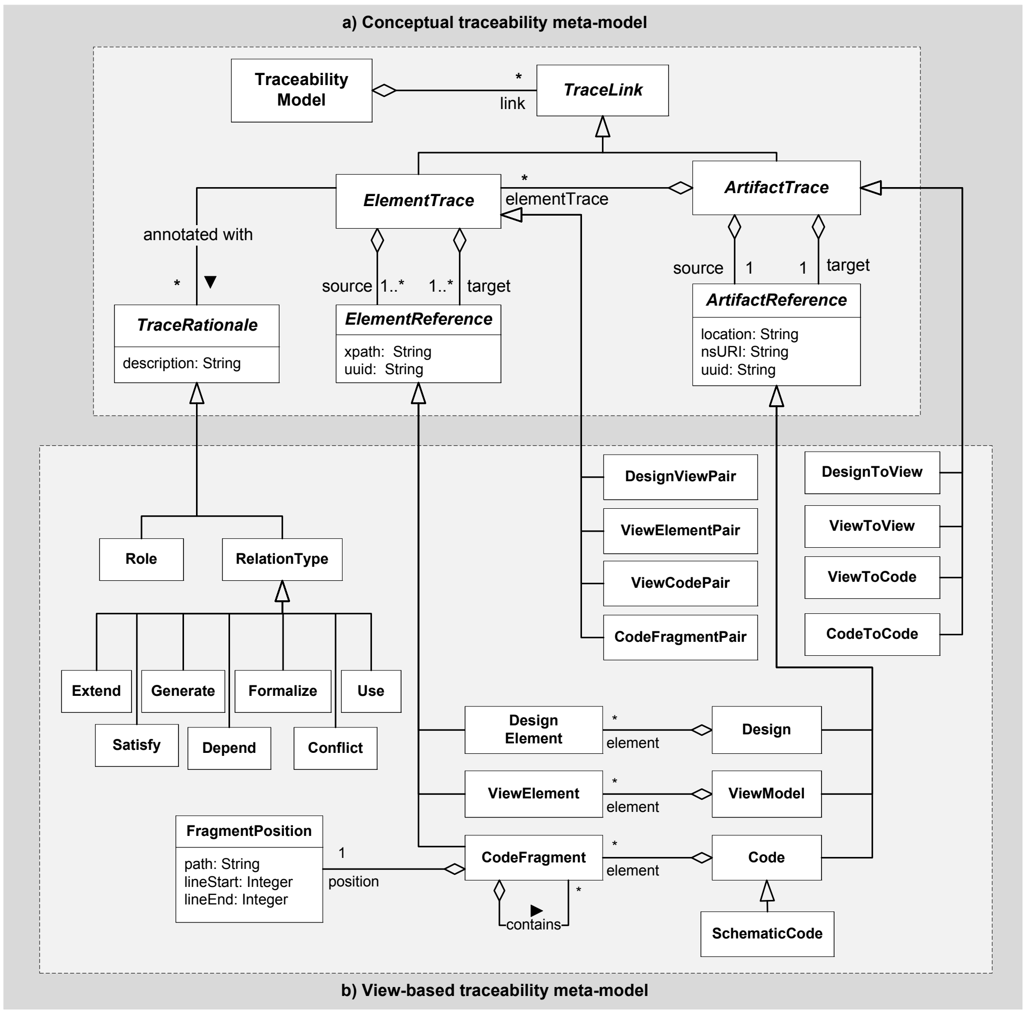


Figure 5.3: View-based traceability meta-models: (a) the conceptual traceability meta-model, and (b) the view-based, model-driven traceability meta-model

5.2.2 View-based traceability meta-model

At the heart of VbTrace, we devise a traceability meta-model that provides concepts for precisely eliciting trace dependencies between process development artifacts. This traceability meta-model is designed to be rich enough for representing trace relations from process design to implementation and be extensible for further customizations and specializations. Figure 5.3a shows the conceptual overview of the meta-model that defines a *TraceabilityModel* containing a number of *TraceLinks*. There are two kinds of *TraceLinks* representing the dependencies at different levels of granularity: *ArtifactTraces* describing the relationships between artifacts such as BPMN diagrams, view models, BPEL and WSDL files, and so on; *ElementTraces* describing the relationships between elements of the same or different artifacts such as BPMN notations, view elements, BPEL activities, WSDL messages, XML Schema elements, and so forth. The source and target of an *ArtifactTrace* are *ArtifactReferences* each of which consisting of either the location path, the namespace URI, or the

UUID[*] of the corresponding artifact. An artifact may contain a number of elements described by the *ElementReference* meta-class. Every *ElementReference* holds either an XPath expression[168] or a UUID which is a universal reference of the underlying actual element.

Each *ElementTrace* might adhere to some *TraceRationales* that comprehend the existence, semantics, causal relations, or additional functionality of the link. The *TraceRationale* is open for extension and must be specialized later depending on specific usage purposes, for instance, for reasoning on trace dependencies concerning the traceability types[116,148]: dependency, require, transform, extend, generalize/refine, implement, generate, use, etc., or setting up dependency priorities or development roles associated with the trace link. Figure 5.3b depicts the extensibility of *TraceRationales* by a number of concrete realizations such as *Role* standing for stakeholders roles and *RelationType* which is further specialized by several types of commonly used trace dependencies[116,148].

The traceability meta-model explained so far provides abstract and generic concepts shaping the basis for a typical traceability approach. In the context of our traceability approach, these abstract concepts are refined to represent trace dependencies of the various view models at different levels of granularity (see Figure 5.3b). We devise four concrete types of *TraceLinks*: *DesignToViews* represent traceability between process designs and VbMF, *ViewToViews* describe internal relationships of VbMF, i.e., relationships between view models and view elements, *ViewToCodes* elicit the traceability from VbMF to process implementations, and finally, *CodeToCodes* describe the relationships between the generated schematic code and the associated individual code.

Languages used for designing processes typically comprise highly abstract, notational elements that business experts are familiar with. A process design artifact presented in the traceability meta-model by the *Design* meta-class. Each *Design* includes several *DesignElements* standing for process design notational elements. The mapping from process designs onto the VbMF abstract layer produces trace links of the *DesignToView* type. Moreover, each *DesignToView* maintains one or many *DesignViewPairs* which are responsible for tracing the mapping relationships at the level of elements, i.e., mapping from design elements to view model elements.

One of the important modeling artifacts provided by VbMF is the *ViewModel* that embodies a number of *ViewElements*. Because there is probably a dependency between two view models, we use *ViewElementPairs* to capture the relationships between view elements of those view models in a fine-grained manner. In particular, a *ViewToView* inherits the two associations from its parent *ArtifactTrace* and holds a number of *ViewElementPairs* standing for the finer granularity of the traceability among view model elements.

In VbMF, the technology-specific view models are rarely developed from scratch but might be gradually refined from existing abstract view models. As such, extracting of trace links is straightforward because VbMF provides the necessary information concerning model refinements. The technology-specific view models can also be extracted from process implementations using the

[*]UUID: Universally Unique Identifier

reverse engineering approach presented in Chapter 4. In contrast, process implementations (i.e., code artifacts) can be automatically produced from technology-specific view models by VbMF code generators. By extending the reverse engineering interpreters and the code generators, we obtain the relevant trace links in terms of *ViewToCodes*, and even finer grained relationships at the level of code fragments by using *ViewCodePairs* that keep references from *ViewElements* to generated *CodeFragments*. A *CodeArtifact* is composed of one or many *CodeFragments* each of which might contain other code fragments. For instance, a WSDL[170] file is a *CodeArtifact* that has a number of fragments such as XML schema definition, message types, service interfaces, service bindings, and service implementations.

Code artifacts generated from the model-driven stack of VbMF are mostly schematic recurring code that needs to be augmented by manually written code, for instance, using the patterns suggested in[150]. Therefore, the traceability meta-model provides another concept for code association, the *CodeToCode* meta-class. Each *CodeToCode* should hold a reference between a certain *SchematicCode* and one of its required manually written *Code* instances.

Last but not least, the abstract *TraceRationale* concept is realized and extended by, but not limited to, a number of popular trace relationships such as *Extend, Generate, Implement* and *Use* that can be employed to augment the semantics of the trace dependencies explained above. Additional rationales or semantics can be derived in the same manner for any further requirements.

```
-- artifact-to-artifact traces
context DesignToView inv:
  source.isKindOf(Design) and target.isKindOf(ViewModel)
context ViewToView
  inv: source.isKindOf(ViewModel) and target.isKindOf(ViewModel)
context ViewToCode
  inv: source.isKindOf(ViewModel) and target.isKindOf(Code)
context CodeToCode
  inv: source.isKindOf(SchematicCode) and target.isKindOf(Code)
-- element-to-element traces
context DesignViewPair
  -- each source must be an element of container's sources
  inv: source->forAll(container.source.element->includes(source))
  -- each target must be an element of container's targets
  inv: target->forAll(container.target.element->includes(target))
context ViewElementPair
  -- similar to those for DesignViewPair
  inv: source->forAll(container.source.element->includes(source))
  inv: target->forAll(container.target.element->includes(target))
context ViewCodePair
  -- each source must be an element of container's sources
  inv: source->forAll(container.source.element->includes(source))
  -- each fragment must belong to the set of container's fragments
  inv: fragment->forAll(container.fragment->includes(fragment) or container.fragment->
       collect(subFragment)->includes(fragment))
context CodeFragmentPair
  -- each fragment must belong to the set of container's fragments
  inv: fragment->forAll(container.fragment->includes(fragment) or container.fragment->
       collect(subFragment)->includes(fragment))
  inv: fragment->forAll(container.fragment->includes(fragment) or container.fragment->
       collect(subFragment)->includes(fragment))
```

Listing 5.1: OCL constraints for the traceability meta-model

Note that the relationships between *Design* and *DesignElement*, between *View* and *ViewElement*, and between *Code* and *CodeFragment* in the traceability meta-model are merely presented for clarification purpose because those relationships can be straightforwardly derived from process design artifacts, VbMF modeling artifacts, and code artifacts, respectively. Toward more strictly modeling of aforementioned traceability links, Listing 5.1 presents OCL constraints[118] for the meta-classes of the traceability meta-model that are required for specifying more precise semantics as well as for the verification of traceability model instances built upon the meta-model.

In summary, the traceability meta-model provides essential concepts for eliciting trace dependencies at different abstraction levels ranging from process design artifacts to abstraction levels of VbMF view models down to code artifacts of process implementations. Each trace link between two levels of abstraction can also support elicitation of the differences of granularity, such as pairing design elements and view elements, or view elements and code fragments. Furthermore, the traceability meta-model is open for extension to finer granularity by deriving new subclasses of pairings such as *DesignViewPair*, *ViewElementPair*, and *ViewCodePair*, or for adding new higher or lower abstraction levels by deriving new sub-types of the *TraceLink*, *ArtifactTrace*, or *ElementTrace* meta-classes. In the subsequent sections, we present the view-based traceability architecture along with the components and mechanisms that supports stakeholders in (semi-)automatically establishing and maintaining the trace dependencies based on the concepts of the aforementioned meta-model.

5.2.3 View-based, model-driven traceability framework architecture

The view-based, model-driven traceability framework architecture shown in Figure 5.4 extends the components of VbMF (see Figure 3.4) in order to acquire traceability relationships. For instance, the extended *View/Instance Editors* produces trace links between view models, between view models and elements, as well as between elements of different view models. These relationships, as mentioned above, are intrinsic parts of VbMF views, and therefore, are straightforwardly extracted. In addition, the extended *View Interpreters* can be utilized for collecting trace dependencies between process designs and view models, and between view models extracted from process implementations and the corresponding implementations. Last but not least, the extended *Code Generator* can establish

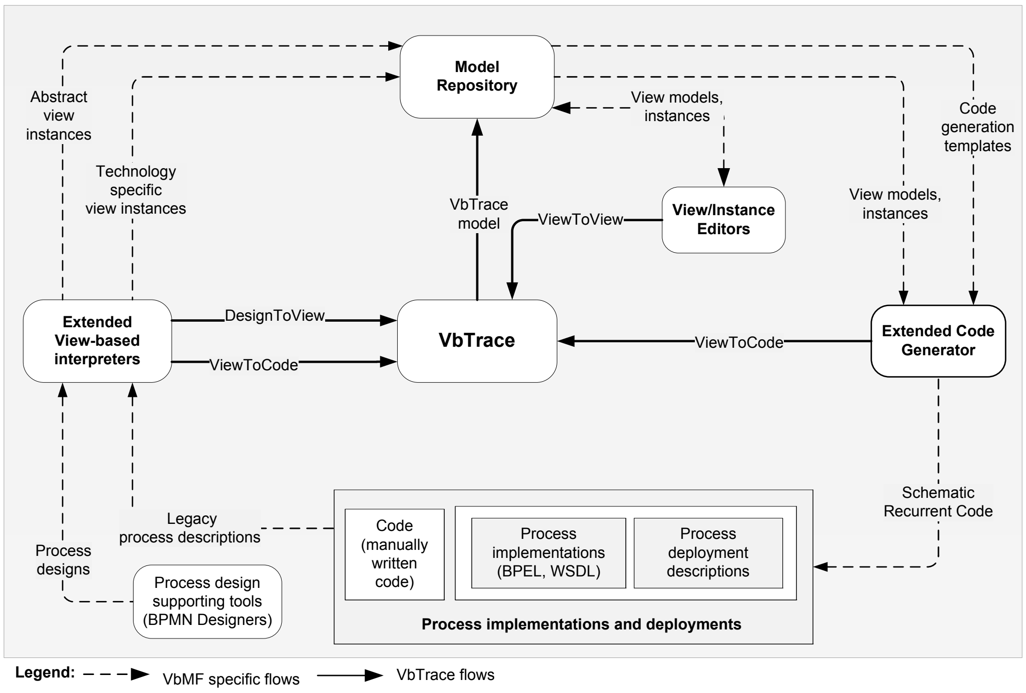

Model Repository
Abstract view instances
Technology specific view instances
View models, instances
Code generation templates
View/Instance Editors
VbTrace model
ViewToView
View models, instances
Extended View-based interpreters
DesignToView
ViewToCode
VbTrace
ViewToCode
Extended Code Generator
Schematic Recurrent Code
Legacy process descriptions
Process designs
Process design supporting tools (BPMN Designers)
Code (manually written code)
Process implementations (BPEL, WSDL)
Process deployment descriptions
Process implementations and deployments
Legend:
VbMF specific flows
VbTrace flows

Figure 5.4: View-based, model-driven traceability framework architecture

trace links from view models used for generating executable process code to the resulting source code artifacts. These extended components retrieve the aforementioned trace dependencies and deliver them to the *VbTrace* as instances of the traceability meta-model. The traceability meta-model and its instances are models themselves, and therefore, can be persisted in the model repository of VbMF for later use and maintenance. The model repository is one of our ongoing works, but beyond the scope of this work.

5.2.4 View-based modeling and traceability toolchain

The traceability meta-model and components mentioned above are essential parts forming the view-based modeling and traceability toolchain shown in Figure 5.5. In this toolchain, process design are mapped into VbMF abstract views whilst process implementations are aligned with VbMF technology-specific views by extending the view-based reverse engineering approach presented in Chapter 3. During these mappings, the extended view-based interpreters are able to establish the relevant trace dependencies *DesignToViews* and *ViewToCodes*, respectively, as well as the fine-grain relationships that are *DesignViewPairs* and *ViewCodePairs*.

Nonetheless, the *ViewToCodes* and *ViewCodePairs* can also be derived in the course of the generation of process implementations (e.g., BPEL and WSDL code) and deployment configurations (e.g., process descriptors for deploying and executing processes in the ActiveBPEL[1], an open source BPEL engine), from VbMF technology-specific views. The transformation templates specify the rules for generation code from VbMF models. We extend these templates to generate the relevant trace

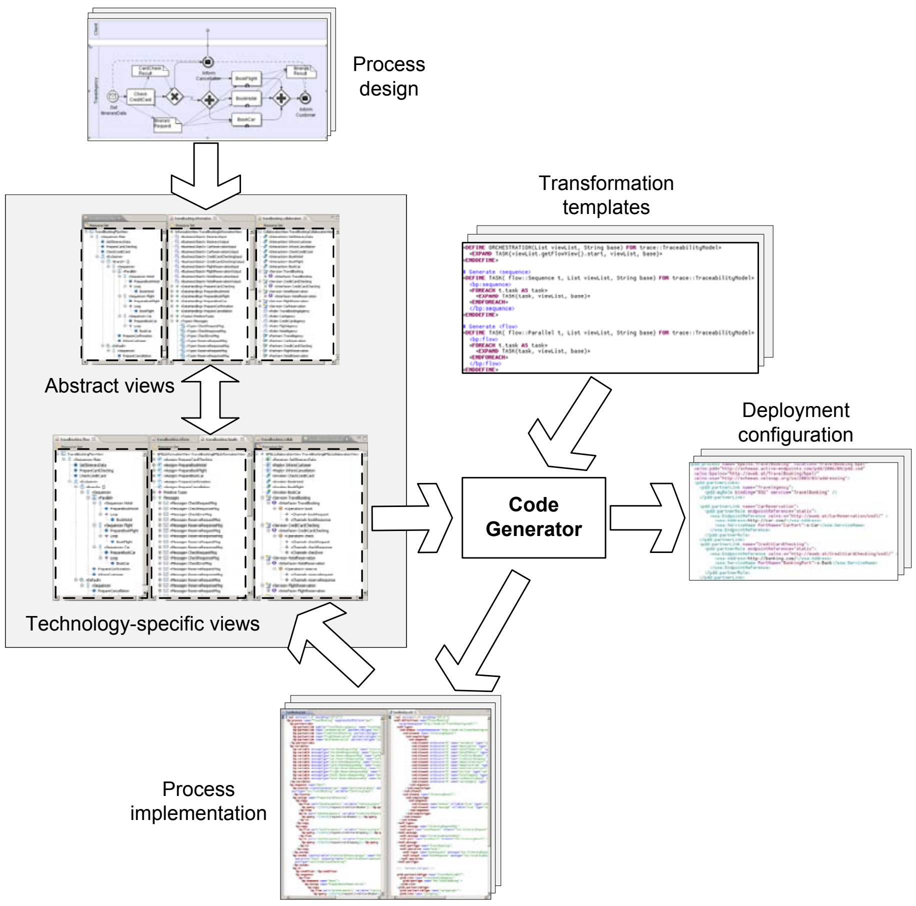
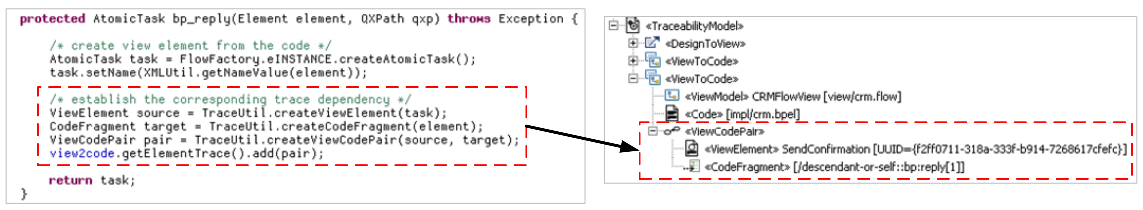


Figure 5.5: View-based modeling and traceability toolchain

dependencies between the view models, view elements, and the generated code artifacts and code fragments.

In the following sections, we elaborate on extending the view-based interpreters and code generation templates using some scenarios in which trace dependencies are established by using our extended code generators and extended view-based interpreters.

5.2.4.1 Establishing trace dependencies using extended view-based interpreters



```
protected AtomicTask bp_reply(Element element, QXPath qxp) throws Exception {

    /* create view element from the code */
    AtomicTask task = FlowFactory.eINSTANCE.createAtomicTask();
    task.setName(XMLUtil.getNameValue(element));

    /* establish the corresponding trace dependency */
    ViewElement source = TraceUtil.createViewElement(task);
    CodeFragment target = TraceUtil.createCodeFragment(element);
    ViewCodePair pair = TraceUtil.createViewCodePair(source, target);
    view2code.getElementTrace().add(pair);

    return task;
}
```

```
«TraceabilityModel»
    «DesignToView»
    «ViewToCode»
    «ViewToCode»
        «ViewModel» CRMFlowView [view/crm.flow]
        «Code» [impl/crm.bpel]
        «ViewCodePair»
            «ViewElement» SendConfirmation [UUID={f2ff0711-318a-333f-b914-7268617cfefc}]
            «CodeFragment» [/descendant-or-self::bp:reply[1]]
```

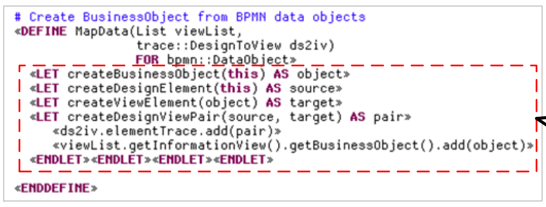
a) Extended FlowView interpreter for extracting FlowView from BPEL code and establishing corresponding trace links

b) Trace dependencies produced from the extended FlowView interpreter

Figure 5.6: Establishing traceability between VbMF views and process implementations

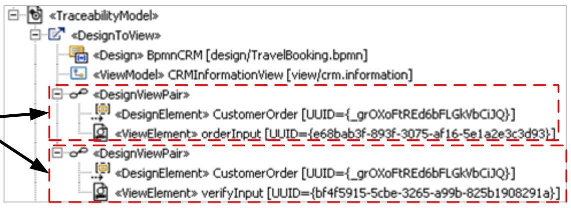
The view-based reverse engineering approach presented in Chapter 4 can be utilized for extracting

VbMF views from existing process implementations in BPEL and WSDL. We extend this approach such that the relevant trace dependencies are also established during the course of the reverse engineering process. Figure 5.8a presents an excerpt of the FlowView interpreter in Java code that can extract *AtomaticTasks* of the FlowView from BPEL code. In addition, we instrument additional Java code for creating trace dependencies between the BPEL code fragment and the resulting *AtomicTasks*. The generated trace dependencies are parts of the *Traceability model* shown in Figure 5.6b. This approach can also be applied for the other view-based interpreters such as the CollaborationView, InformationView, BpelCollaborationView, and BpelInformationView interpreters in order to automatically establish the relevant trace dependencies between BPEL and WSDL descriptions and VbMF views. For better supporting stakeholders in reasoning and analyzing the resulting trace dependencies, for instance, change impact analysis, *Generate* and *Formalize* are automatically annotated to each trace dependency.



```
# Create BusinessObject from BPMN data objects
«DEFINE MapData(List viewList,
                trace::DesignToView ds2iv)
                FOR bpmn::DataObject»
  «LET createBusinessObject(this) AS object»
  «LET createDesignElement(this) AS source»
  «LET createViewElement(object) AS target»
  «LET createDesignViewPair(source, target) AS pair»
     «ds2iv.elementTrace.add(pair)»
     «viewList.getInformationView().getBusinessObject().add(object)»
  «ENDLET»«ENDLET»«ENDLET»«ENDLET»

«ENDDEFINE»
```

a) Extended template rules for mapping a ***DataObject*** (BPMN) to a ***BusinessObject*** (InformationView) and establishing corresponding trace links

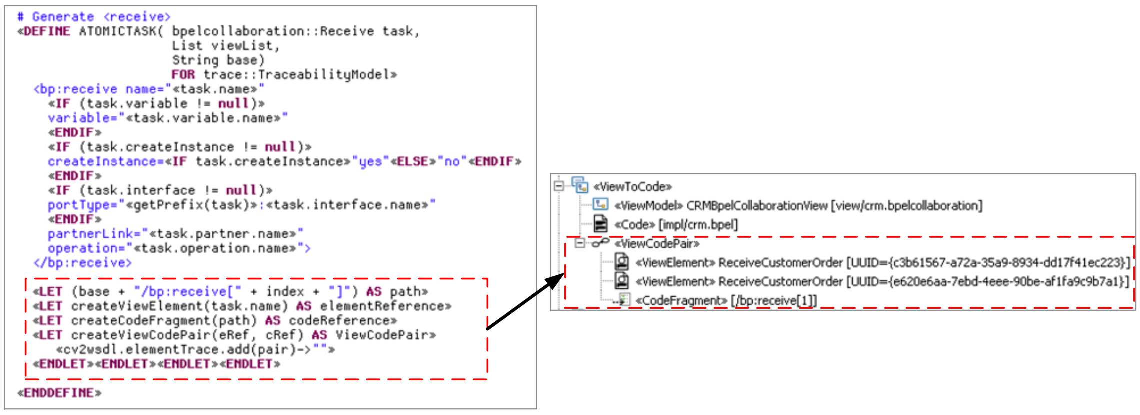


```
«TraceabilityModel»
  «DesignToView»
    «Design» BpmnCRM [design/TravelBooking.bpmn]
    «ViewModel» CRMInformationView [view/crm.information]
    «DesignViewPair»
      «DesignElement» CustomerOrder [UUID={_grOXoFtREd6bFLGkVbCiJQ}]
      «ViewElement» orderInput [UUID={e68bab3f-893f-3075-af16-5e1a2e3c3d93}]
    «DesignViewPair»
      «DesignElement» CustomerOrder [UUID={_grOXoFtREd6bFLGkVbCiJQ}]
      «ViewElement» verifyInput [UUID={bf4f5915-5cbe-3265-a99b-825b1908291a}]
```

b) Trace dependencies produced from the mapping of BPMN elements onto VbMF views

Figure 5.7: Mapping process designs to VbMF views and establishing trace dependencies

Although the view-based reverse engineering approach is exemplified using process implementation languages such as BPEL and WSDL, it is extensible and applicable for mapping the concepts of a process design into VbMF abstract views. Let us recall that VbMF view models at the abstract layer are intentionally designed for business experts. As a result, the concepts embodied in these view models have a close relationship to the elements of languages used for designing processes, such as BPMN, UML Activity Diagram, EPCs, and so on. A minor difference of these high-level view interpreters to the view interpreters mentioned above is that we realize the view-based reverse engineering approach using openArchitectureWare Xpand and Xtend languages[122] due to their sufficient transformation mechanisms. Figure 5.7a shows an excerpt of the template-based transformation rules written in XPand language that maps a BPMN *Data Object* into a *BusinessObject* element of the InformationView. In addition, the transformation rules also generate relevant trace dependencies between the design and view elements. We illustrate in Figure 5.7b a part of the traceability model comprising two *DesignToView* trace links between the design and the FlowView of the CRM Fulfillment process from the case study presented in Section 5.3. These trace dependencies are augmented with the *Formalize* of type *TraceRationale*.



```
# Generate <receive>
«DEFINE ATOMICTASK( bpelcollaboration::Receive task,
                    List viewList,
                    String base)
                    FOR trace::TraceabilityModel»
  <bp:receive name="«task.name»"
    «IF (task.variable != null)»
    variable="«task.variable.name»"
    «ENDIF»
    «IF (task.createInstance != null)»
    createInstance=«IF task.createInstance»"yes"«ELSE»"no"«ENDIF»
    «ENDIF»
    «IF (task.interface != null)»
    portType="«getPrefix(task)»:«task.interface.name»"
    «ENDIF»
    partnerLink="«task.partner.name»"
    operation="«task.operation.name»">
  </bp:receive>

  «LET (base + "/bp:receive[" + index + "]") AS path»
  «LET createViewElement(task.name) AS elementReference»
  «LET createCodeFragment(path) AS codeReference»
  «LET createViewCodePair(eRef, cRef) AS ViewCodePair»
     «cv2wsdl.elementTrace.add(pair)->""»
  «ENDLET»«ENDLET»«ENDLET»«ENDLET»

«ENDDEFINE»
```

```
«ViewToCode»
  «ViewModel» CRMBpelCollaborationView [view/crm.bpelcollaboration]
  «Code» [impl/crm.bpel]
  «ViewCodePair»
    «ViewElement» ReceiveCustomerOrder [UUID={c3b61567-a72a-35a9-8934-dd17f41ec223}]
    «ViewElement» ReceiveCustomerOrder [UUID={e620e6aa-7ebd-4eee-90be-af1fa9c9b7a1}]
    «CodeFragment» [/bp:receive[1]]
```

a) Extended template rules for generating BPEL <**receive**> from VbMF technology-specific views and establishing corresponding trace links

b) Trace dependencies produced from code generation

Figure 5.8: Generating process code from VbMF views and establishing trace dependencies

5.2.4.2 Establishing trace dependencies using extended code generators

Code generation (or so-called *model-to-code* transformation) is an important step of any realization of the MDD paradigm to gain productivity and ensure better software quality[150]. The results of code generation process are often the schematic, recurring code that shapes the skeleton of the software or systems. Some manually written code (aka individual code) might augment the generated schematic code in order to realize the individual parts of the business logic[150]. VbMF provides a template-based code generation approach that is able to generate schematic implementations of processes in terms of BPEL and WSDL descriptions. This approach has been realized in VbMF using the openArchitectureWare Xpand and Xtend languages[122]. We extend the template-based code generation rules in VbMF such that the trace dependencies between the involved views, view elements, and generated code fragments are automatically established. Figure 5.8a presents an excerpt of the VbMF code generation rules for generating BPEL `<invoke>` elements from VbMF technology-specific views along with our instrumented rules for generating trace dependencies. The resulting trace dependencies are illustrated in Figure 5.8b containing a *ViewToCode* trace link between the VbMF BpelCollaborationView and BPEL `<invoke>` fragment extracted from the case study (cf. Section 5.3). Although these trace dependencies are generated in the opposite direction to those extracted from the reverse engineering of process implementations, they share similar semantics and rationales of the trace relations that are *Generate* and *Formalize*.

5.3 Tool support and case study

In this section, we illustrate the realization of the aforementioned concepts in VbTrace via an industry case study adapted from the CRM Fulfillment process (cf. Section 6.2.1). Figure 5.9 shows the design of the CRM Fulfillment process in terms of a BPMN diagram. The process is implemented using process-driven SOA technology: BPEL and WSDL. BPMN, BPEL, and WSDL are used for exemplification because these are likely the most popular process and service description languages that are widely adopted in research and industry today. Nevertheless, our approach is not limited to those but is generally applicable for other process-driven SOA technologies. To illustrate the process deployment configurations, we exemplify a specific BPEL engine, namely, ActiveBPEL, and develop the necessary configurations for the deployment, enactment and monitoring of the CRM Fulfillment process.

In the subsequent sections, we first quickly introduce the tool support for our traceability approach. Next, we present in detail the steps of establishing and maintaining appropriate traceability meta-data between process designs and VbMF, among VbMF views, and between VbMF views and process implementations. At the end of this section, we introduce a sample of using traceability path derived from the traceability model for better understanding and analyzing the relationships of process development artifacts.

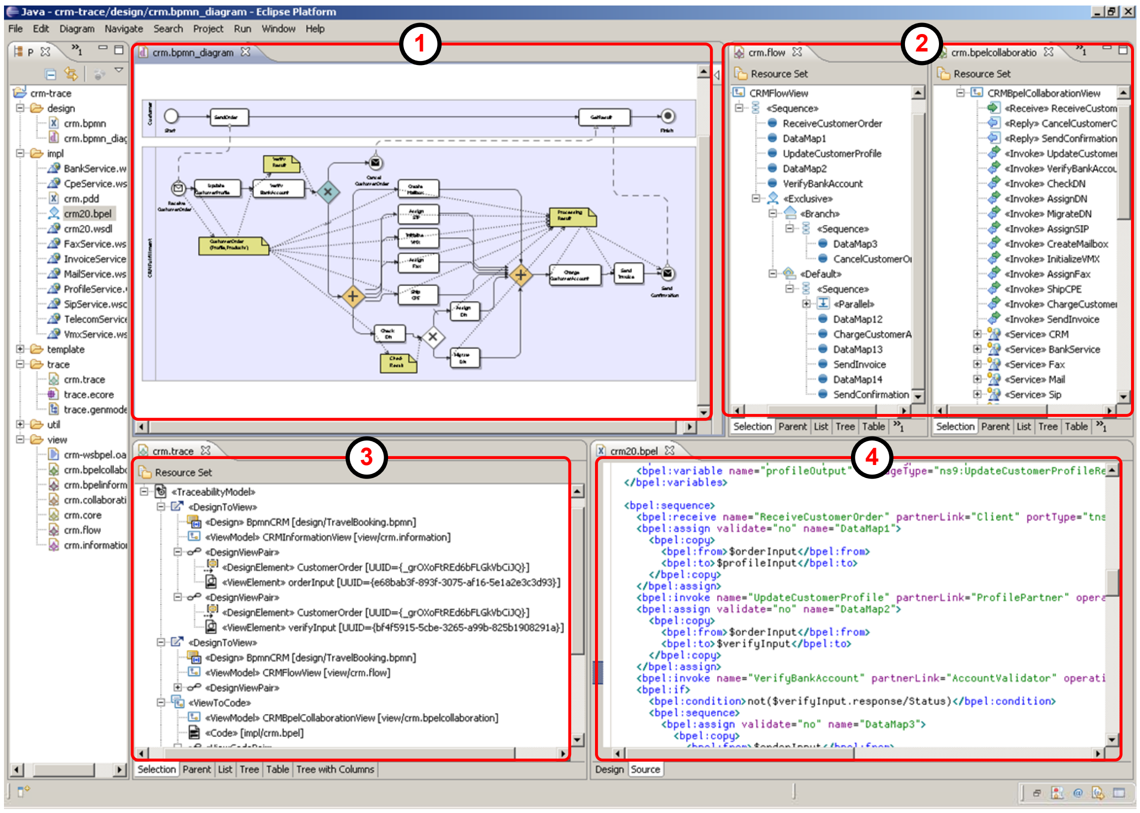
5.3.1 View-based, model-driven integrated development environment

We have realized the concepts of our view-based, model-driven traceability approach presented based on the aforementioned VbMF implementation and integrated them with VbMF in terms of a view-based, model-driven integrated development environment. In order to effectively reuse and extend VbMF concepts and mechanisms, the traceability framework is derived from the EMF Ecore meta-model. The biggest advantage of using Eclipse Modeling Framework is that we gain better interoperability with the Eclipse BPMN Modeler[39] which is developed based on EMF Ecore.

For the sake of demonstration, we use the BPMN diagrams designed in the Eclipse BPMN Modeler to represent the process design, and extend the tree-based editor generated by EMF for presenting and manipulating *Traceability models* from now on. The components of our traceability framework, such as *Extended Code Generators* and *Extended View-based Interpreters* (see Figure 5.4), are derived from corresponding VbMF components (see Figure 3.4) using the mechanisms described in Section 5.2.4. Figure 5.9 shows the design of the CRM Fulfillment process in terms of a BPMN diagram in Eclipse BPMN Modeler.

5.3.2 CRM Fulfillment process development and traceability

Figure 5.9 depicts one of development perspective of the CRM Fulfillment process using our view-based, model-driven integrated environment which is an Eclipse-based workbench. The stakeholders can create and manipulate process views in the various VbMF view editors or extract views from process designs and implementations using the built-in view-based reverse engineering. Given these process views, stakeholders can generate process implementations such as process code in BPEL, service interfaces of processes in WSDL and process deployment configurations by using the


Java - crm-trace/design/crm.bpmn_diagram - Eclipse Platform
File Edit Diagram Navigate Search Project Run Window Help
1
2
3
4
crm.bpmn_diagram
crm.flow
crm.bpelcollaboratio
crm.trace
crm20.bpel
Resource Set
Selection Parent List Tree Table Tree with Columns
Design Source

```
CRMFlowView
  «Sequence»
    ReceiveCustomerOrder
    DataMap1
    UpdateCustomerProfile
    DataMap2
    VerifyBankAccount
    «Exclusive»
      «Branch»
        «Sequence»
          DataMap3
          CancelCustomerOr
      «Default»
        «Sequence»
          «Parallel»
          DataMap12
          ChargeCustomerA
          DataMap13
          SendInvoice
          DataMap14
          SendConfirmation
```

```
CRMBpelCollaborationView
  «Receive» ReceiveCustom
  «Reply» CancelCustomerC
  «Reply» SendConfirmation
  «Invoke» UpdateCustomer
  «Invoke» VerifyBankAccou
  «Invoke» CheckDN
  «Invoke» AssignDN
  «Invoke» MigrateDN
  «Invoke» AssignSIP
  «Invoke» CreateMailbox
  «Invoke» InitializeVMX
  «Invoke» AssignFax
  «Invoke» ShipCPE
  «Invoke» ChargeCustomer
  «Invoke» SendInvoice
  «Service» CRM
  «Service» BankService
  «Service» Fax
  «Service» Mail
  «Service» Sip
```

```
«TraceabilityModel»
  «DesignToView»
    «Design» BpmnCRM [design/TravelBooking.bpmn]
    «ViewModel» CRMInformationView [view/crm.information]
    «DesignViewPair»
      «DesignElement» CustomerOrder [UUID={_grOXoFtREd6bFLGkVbCiJQ}]
      «ViewElement» orderInput [UUID={e68bab3f-893f-3075-af16-5e1a2e3c3d93}]
    «DesignViewPair»
      «DesignElement» CustomerOrder [UUID={_grOXoFtREd6bFLGkVbCiJQ}]
      «ViewElement» verifyInput [UUID={bf4f5915-5cbe-3265-a99b-825b1908291a}]
  «DesignToView»
    «Design» BpmnCRM [design/TravelBooking.bpmn]
    «ViewModel» CRMFlowView [view/crm.flow]
    «DesignViewPair»
  «ViewToCode»
    «ViewModel» CRMBpelCollaborationView [view/crm.bpelcollaboration]
    «Code» [impl/crm.bpel]
```

```
    <bpel:variable name="profileOutput"      geType="ns9:UpdateCustomerProfileRe
  </bpel:variables>

  <bpel:sequence>
    <bpel:receive name="ReceiveCustomerOrder" partnerLink="Client" portType="tns
    <bpel:assign validate="no" name="DataMap1">
      <bpel:copy>
        <bpel:from>$orderInput</bpel:from>
        <bpel:to>$profileInput</bpel:to>
      </bpel:copy>
    </bpel:assign>
    <bpel:invoke name="UpdateCustomerProfile" partnerLink="ProfilePartner" opera
    <bpel:assign validate="no" name="DataMap2">
      <bpel:copy>
        <bpel:from>$orderInput</bpel:from>
        <bpel:to>$verifyInput</bpel:to>
      </bpel:copy>
    </bpel:assign>
    <bpel:invoke name="VerifyBankAccount" partnerLink="AccountValidator" operati
    <bpel:if>
      <bpel:condition>not($verifyInput.response/Status)</bpel:condition>
      <bpel:sequence>
        <bpel:assign validate="no" name="DataMap3">
          <bpel:copy>
```

Figure 5.9: CRM Fulfillment process in view-based, model-driven integrated environment: (1) The process design in BPMN Modeler (2) VbMF views, (3) Traceability view, and (4) Generated process implementation

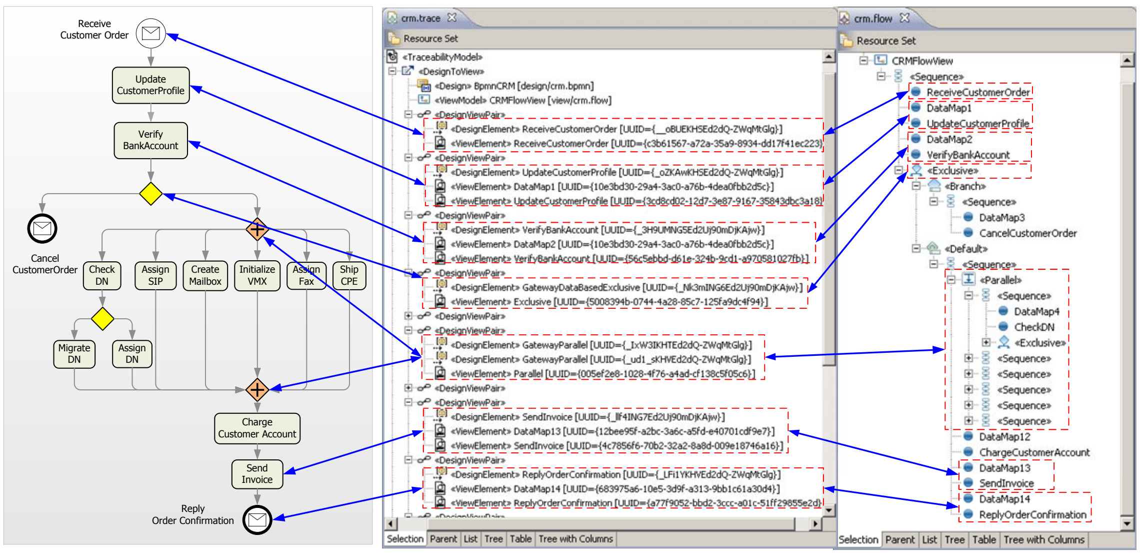
predefined template-based code generation rules. Moreover, the code generation templates can also be customized according to further needs by using the the XPand language editor[122]. In addition, the trace dependencies established during the course of process development are presented to the stakeholders in the Traceability view.

The subsequent sections present the various scenarios to demonstrate how relevant trace dependencies between process designs and VbMF views, between VbMF views, and between VbMF views and process implementations are established during the course of modeling and developing the CRM Fulfillment process.

5.3.2.1 Scenario 1: Traceability between process design and VbMF views

The CRM Fulfillment process design is a BPMN diagram that comprises a number of notational elements such as a pool, tasks, data objects, and sequential flow connectors (see Figure 5.9). For the sake of readability and demonstration, we adapt the design of CRM Fulfillment process and omit the *Data Objects* which are irrelevant in this scenario.

In the context of process-driven SOAs, VbMF leverages the FlowView model as the central notation because this model represents the orchestration of the process activities. We demonstrate the



```
«TraceabilityModel»
  «DesignToView»
    «Design» BpmnCRM [design/crm.bpmn]
    «ViewModel» CRMFlowView [view/crm.flow]
    «DesignViewPair»
      «DesignElement» ReceiveCustomerOrder [UUID={__oBUEKHSEd2dQ-ZWqMtGlg}]
      «ViewElement» ReceiveCustomerOrder [UUID={c3b61567-a72a-35a9-8934-dd17f41ec223}
    «DesignViewPair»
      «DesignElement» UpdateCustomerProfile [UUID={_oZKAwKHSEd2dQ-ZWqMtGlg}]
      «ViewElement» DataMap1 [UUID={10e3bd30-29a4-3ac0-a76b-4dea0fbb2d5c}]
      «ViewElement» UpdateCustomerProfile [UUID={3cd8cd02-12d7-3e87-9167-35843dbc3a18}
    «DesignViewPair»
      «DesignElement» VerifyBankAccount [UUID={_3H9UMNG5Ed2Uj90mDjKAjw}]
      «ViewElement» DataMap2 [UUID={10e3bd30-29a4-3ac0-a76b-4dea0fbb2d5c}]
      «ViewElement» VerifyBankAccount [UUID={56c5ebbd-d61e-324b-9cd1-a970581027fb}]
    «DesignViewPair»
      «DesignElement» GatewayDataBasedExclusive [UUID={_Nk3mING6Ed2Uj90mDjKAjw}]
      «ViewElement» Exclusive [UUID={5008394b-0744-4a28-85c7-125fa9dc4f94}]
    «DesignViewPair»
    «DesignViewPair»
      «DesignElement» GatewayParallel [UUID={_IxW3IKHTEd2dQ-ZWqMtGlg}]
      «DesignElement» GatewayParallel [UUID={_ud1_sKHVEd2dQ-ZWqMtGlg}]
      «ViewElement» Parallel [UUID={005ef2e8-1028-4f76-a4ad-cf138c5f05c6}]
    «DesignViewPair»
    «DesignViewPair»
      «DesignElement» SendInvoice [UUID={_llf4ING7Ed2Uj90mDjKAjw}]
      «ViewElement» DataMap13 [UUID={12bee95f-a2bc-3a6c-a5fd-e40701cdf9e7}]
      «ViewElement» SendInvoice [UUID={4c7856f6-70b2-32a2-8a8d-009e18746a16}]
    «DesignViewPair»
      «DesignElement» ReplyOrderConfirmation [UUID={_LFi1YKHVEd2dQ-ZWqMtGlg}]
      «ViewElement» DataMap14 [UUID={683975a6-10e5-3d9f-a313-9bb1c61a30d4}]
      «ViewElement» ReplyOrderConfirmation [UUID={a77f9052-bbd2-3ccc-a01c-51ff29855e2d}
```

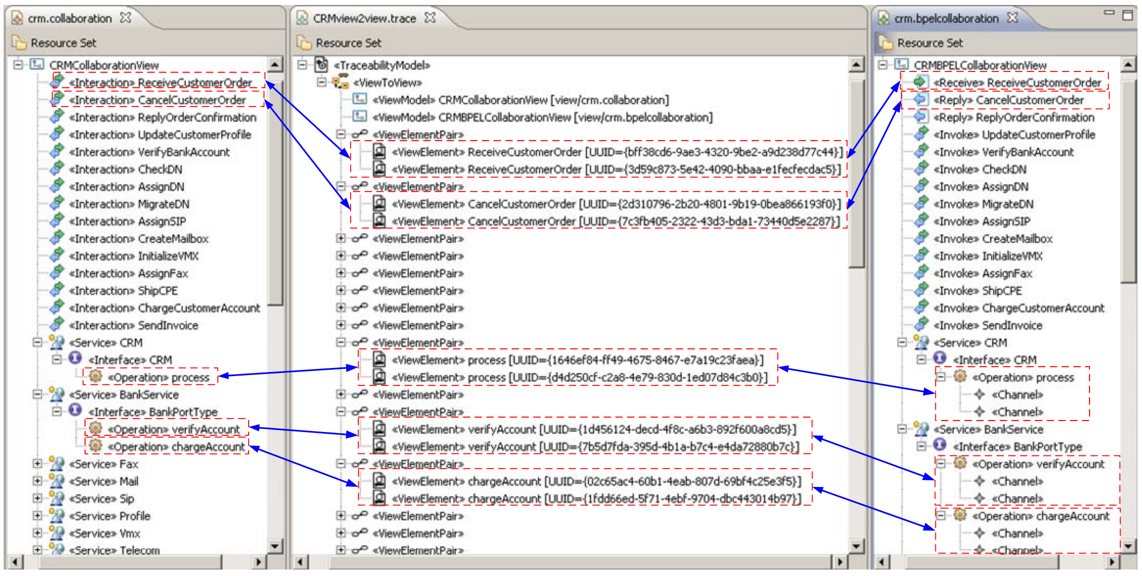
```
CRMFlowView
  «Sequence»
    ReceiveCustomerOrder
    DataMap1
    UpdateCustomerProfile
    DataMap2
    VerifyBankAccount
    «Exclusive»
      «Branch»
        «Sequence»
          DataMap3
          CancelCustomerOrder
      «Default»
        «Sequence»
          «Parallel»
            «Sequence»
              DataMap4
              CheckDN
              «Exclusive»
            «Sequence»
            «Sequence»
            «Sequence»
            «Sequence»
            «Sequence»
          DataMap12
          ChargeCustomerAccount
          DataMap13
          SendInvoice
          DataMap14
          ReplyOrderConfirmation
```

a) CRM Fulfillment process design b) CRM Traceability model c) The CRM Fulfillment FlowView

Figure 5.10: Traceability between CRM Fulfillment process design and FlowView

mapping of the BPMN design onto the FlowView of the CRM Fulfillment process along with the trace dependencies established during the mapping (see Figure 5.10) by using the approach mentioned in Section 5.2.4. The trace dependencies includes trace links at coarse-grained levels, i.e., between the BPMN diagram and the FlowView model, or at finer granularities, e.g., between a BPMN task and a FlowView's *Atomic Task*, between a BPMN *GatewayDataBasedExclusive* and a conditional switch, namely, *Exclusive* of the FlowView, and so on. Taking the same approach of mapping the CRM Fulfillment process design onto the FlowView, we have developed more view-based interpreters for extracting abstract view models from the process design and establishing tracing relationships.

Note that VbMF is a realization of the *separation of concerns* principle. In VbMF, the FlowView model merely represents the control structures, i.e., the orchestration concern of business processes, which describe the execution order of process activities in order to accomplish a certain business goal. However, the FlowView does not contain any details of these tasks. Other views, according to their specific syntaxes and semantics, provide the concrete definitions of each of FlowView's tasks. For instance, a service invocation task of the FlowView is realized in a CollaborationView or an extension of the CollaborationView whilst a data processing task is defined in an InformationView or an extension of the InformationView. In this way, the FlowView model aims at supporting stakeholders, especially business experts, to quickly design the business functionality by orchestrating named activities rather than being stuck with other details such as performing remote invocations, activity compensation, and so on. These details are accordingly defined in abstract view models and/or refined down to technology-specific view models by the relevant IT experts. As a consequence, these views, regardless whether they are abstract or technology-specific, can be integrated with the FlowView using the view integration mechanism in order to produce richer views or a thorough view of the whole process with respect to the particular needs, knowledge, and skills of stakeholders.



```
crm.collaboration
Resource Set
CRMCollaborationView
  «Interaction» ReceiveCustomerOrder
  «Interaction» CancelCustomerOrder
  «Interaction» ReplyOrderConfirmation
  «Interaction» UpdateCustomerProfile
  «Interaction» VerifyBankAccount
  «Interaction» CheckDN
  «Interaction» AssignDN
  «Interaction» MigrateDN
  «Interaction» AssignSIP
  «Interaction» CreateMailbox
  «Interaction» InitializeVMX
  «Interaction» AssignFax
  «Interaction» ShipCPE
  «Interaction» ChargeCustomerAccount
  «Interaction» SendInvoice
  «Service» CRM
    «Interface» CRM
      «Operation» process
  «Service» BankService
    «Interface» BankPortType
      «Operation» verifyAccount
      «Operation» chargeAccount
  «Service» Fax
  «Service» Mail
  «Service» Sip
  «Service» Profile
  «Service» Vmx
  «Service» Telecom
```

```
CRMview2view.trace
Resource Set
«TraceabilityModel»
  «ViewToView»
    «ViewModel» CRMCollaborationView [view/crm.collaboration]
    «ViewModel» CRMBPELCollaborationView [view/crm.bpelcollaboration]
    «ViewElementPair»
      «ViewElement» ReceiveCustomerOrder [UUID={bff38cd6-9ae3-4320-9be2-a9d238d77c44}]
      «ViewElement» ReceiveCustomerOrder [UUID={3d59c873-5e42-4090-bbaa-e1fecfecdac5}]
    «ViewElementPair»
      «ViewElement» CancelCustomerOrder [UUID={2d310796-2b20-4801-9b19-0bea866193f0}]
      «ViewElement» CancelCustomerOrder [UUID={7c3fb405-2322-43d3-bda1-73440d5e2287}]
    «ViewElementPair»
    «ViewElementPair»
    «ViewElementPair»
    «ViewElementPair»
    «ViewElementPair»
    «ViewElementPair»
    «ViewElementPair»
      «ViewElement» process [UUID={1646ef84-ff49-4675-8467-e7a19c23faea}]
      «ViewElement» process [UUID={d4d250cf-c2a8-4e79-830d-1ed07d84c3b0}]
    «ViewElementPair»
    «ViewElementPair»
      «ViewElement» verifyAccount [UUID={1d456124-decd-4f8c-a6b3-892f600a8cd5}]
      «ViewElement» verifyAccount [UUID={7b5d7fda-395d-4b1a-b7c4-e4da72880b7c}]
    «ViewElementPair»
      «ViewElement» chargeAccount [UUID={02c65ac4-60b1-4eab-807d-69bf4c25e3f5}]
      «ViewElement» chargeAccount [UUID={1fdd66ed-5f71-4ebf-9704-dbc443014b97}]
    «ViewElementPair»
    «ViewElementPair»
    «ViewElementPair»
```

```
crm.bpelcollaboration
Resource Set
CRMBPELCollaborationView
  «Receive» ReceiveCustomerOrder
  «Reply» CancelCustomerOrder
  «Reply» ReplyOrderConfirmation
  «Invoke» UpdateCustomerProfile
  «Invoke» VerifyBankAccount
  «Invoke» CheckDN
  «Invoke» AssignDN
  «Invoke» MigrateDN
  «Invoke» AssignSIP
  «Invoke» CreateMailbox
  «Invoke» InitializeVMX
  «Invoke» AssignFax
  «Invoke» ShipCPE
  «Invoke» ChargeCustomerAccount
  «Invoke» SendInvoice
  «Service» CRM
    «Interface» CRM
      «Operation» process
        «Channel»
        «Channel»
  «Service» BankService
    «Interface» BankPortType
      «Operation» verifyAccount
        «Channel»
        «Channel»
      «Operation» chargeAccount
        «Channel»
        «Channel»
```

a) CRM Fulfillment CollaborationView b) CRM Traceability model c) The CRM Fulfillment BpelCollaborationView

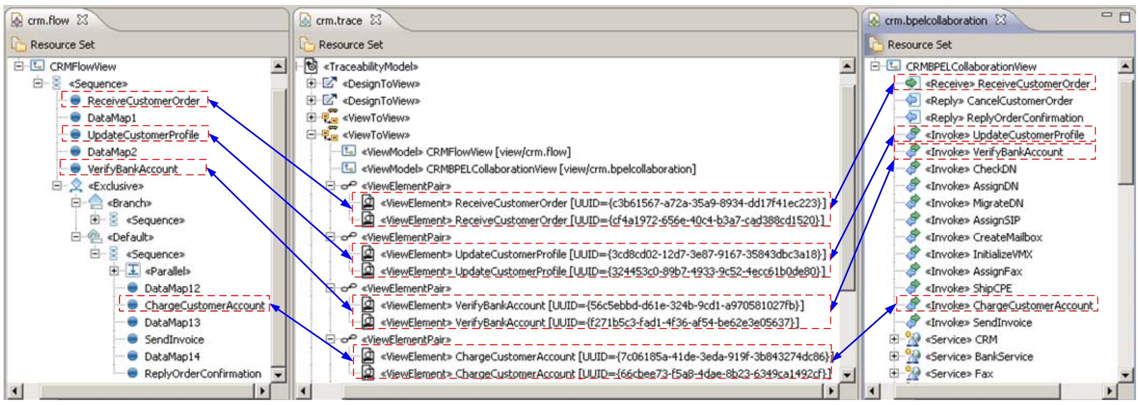
Figure 5.11: Traceability between CRM CollaborationView and BpelCollaborationView

5.3.2.2 Scenario 2: Traceability between VbMF views

View models at the abstract layer of VbMF are intentionally designed for business experts alike who are not familiar or able to work with the technical details. As such, these models supplement the FlowView with adequate concepts and perspectives. In other words, the abstract views can be considered platform-independent models (PIMs)[51,150] that have close relationships with process designs rather than the implementation counterparts. In the model-driven stack of VbMF, an abstract view can be gradually refined down to its corresponding technology-specific view. For instance, the BpelCollaborationView is a stepwise refinement of the more abstract CollaborationView (cf. Figure 3.15). Thus, refinement relationships are important for tracing from process design to implementations. We track these relationships by using trace links of the type *ViewToView* for supporting the traceability between two view models, and a number of *ViewElementPairs* each of which holds references to the corresponding view elements.

Figure 5.11 shows an illustration of establishing the trace dependencies out of the refinement of the CRM CollaborationView (Figure 5.11a) down to the CRM BpelCollaborationView (Figure 5.11(c)) described by the *ViewToView* and its constituent *ViewElementPairs*. For the sake of readability, we only present a number of selected trace dependencies and use the arrows to depict the links described by each dependency. Each trace dependency is augmented with the *Refine* of type *TraceRationale*.

Additionally, VbMF views can be integrated to produce richer views. For instance, a certain stakeholder might need to see the orchestration of the CRM Fulfillment process activities along with the



```
crm.flow
Resource Set
CRMFlowView
  «Sequence»
    ReceiveCustomerOrder
    DataMap1
    UpdateCustomerProfile
    DataMap2
    VerifyBankAccount
    «Exclusive»
      «Branch»
        «Sequence»
      «Default»
        «Sequence»
          «Parallel»
          DataMap12
          ChargeCustomerAccount
          DataMap13
          SendInvoice
          DataMap14
          ReplyOrderConfirmation
```

```
crm.trace
Resource Set
«TraceabilityModel»
  «DesignToView»
  «DesignToView»
  «ViewToView»
  «ViewToView»
    «ViewModel» CRMFlowView [view/crm.flow]
    «ViewModel» CRMBPELCollaborationView [view/crm.bpelcollaboration]
    «ViewElementPair»
      «ViewElement» ReceiveCustomerOrder [UUID={c3b61567-a72a-35a9-8934-dd17f41ec223}]
      «ViewElement» ReceiveCustomerOrder [UUID={cf4a1972-656e-40c4-b3a7-cad388cd1520}]
    «ViewElementPair»
      «ViewElement» UpdateCustomerProfile [UUID={3cd8cd02-12d7-3e87-9167-35843dbc3a18}]
      «ViewElement» UpdateCustomerProfile [UUID={324453c0-89b7-4933-9c52-4ecc61b0de80}]
    «ViewElementPair»
      «ViewElement» VerifyBankAccount [UUID={56c5ebbd-d61e-324b-9cd1-a970581027fb}]
      «ViewElement» VerifyBankAccount [UUID={f271b5c3-fad1-4f36-af54-be62e3e05637}]
    «ViewElementPair»
      «ViewElement» ChargeCustomerAccount [UUID={7c06185a-41de-3eda-919f-3b843274dc86}]
      «ViewElement» ChargeCustomerAccount [UUID={66cbee73-f5a8-4dae-8b23-6349ca1492cf}]
```

```
crm.bpelcollaboration
Resource Set
CRMBPELCollaborationView
  «Receive» ReceiveCustomerOrder
  «Reply» CancelCustomerOrder
  «Reply» ReplyOrderConfirmation
  «Invoke» UpdateCustomerProfile
  «Invoke» VerifyBankAccount
  «Invoke» CheckDN
  «Invoke» AssignDN
  «Invoke» MigrateDN
  «Invoke» AssignSIP
  «Invoke» CreateMailbox
  «Invoke» InitializeVMX
  «Invoke» AssignFax
  «Invoke» ShipCPE
  «Invoke» ChargeCustomerAccount
  «Invoke» SendInvoice
  «Service» CRM
  «Service» BankService
  «Service» Fax
```

a) CRM Fulfillment FlowView   b) CRM Traceability model   c) The CRM Fulfillment BpelCollaborationView

Figure 5.12: Traceability between CRM FlowView and BpelCollaborationView

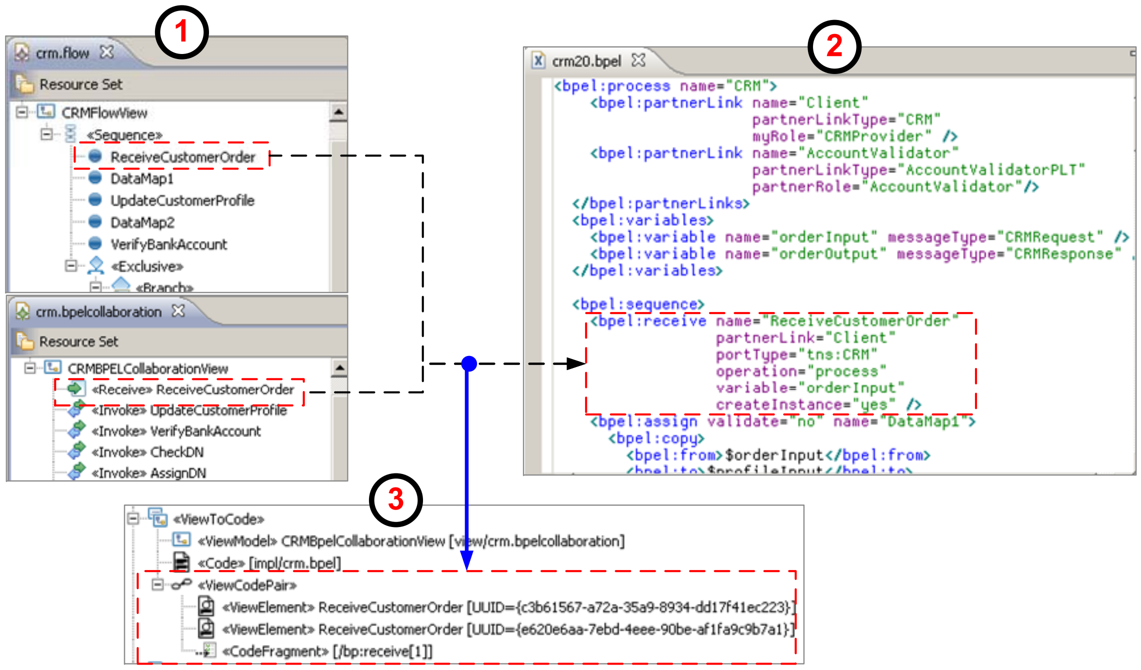
interactions between the process and other processes or services. The combination of the FlowView model and either the CollaborationView or the BpelCollaborationView based on the name-based matching approach (cf. Section 3.5) can offer such a perspective. Figure 5.12 shows an illustration of establishing the trace relationships out of such combinations. The main purpose of view integration is to enhance the flexibility of VbMF for providing various adapted and tailored perspectives of the process model. Because those perspectives might be used by the stakeholders for analyzing and manipulating the process model, we track down the relationships caused by the above-mentioned combination in the traceability according to specific stakeholders' actions and augment them with the *Dependency* type.

5.3.2.3 Scenario 3: Traceability between VbMF views and process implementations

In the previous sections, we illustrate the methods for establishing the traceability path connecting the CRM Fulfillment process design to VbMF view models at the abstract layer down to the technology-specific layer. The relationships between view models and process implementations, however, can be achieved in two different ways. On the one hand, schematic code of process implementations or process deployment descriptors can be generated from the technology-specific views (such as the BpelCollaborationView, BpelInformationView, etc.) at the final step of VbMF top-down toolchain (cf. Section 3.7). On the other hand, the reverse engineering approach (cf. Chapter 4) can automatically extract view models from existing (legacy) process implementations. Regardless of using any of these methods, the trace dependencies need to be recorded to maintain appropriate relationships between view models and process implementations to fully accomplish the traceability path from process designs to the implementation counterparts (see Figure 5.13)*.

Furthermore, a number of process engine specific descriptors are necessary for successfully deploying and executing the CRM Fulfillment process. ActiveBPEL[1] is exemplified as the reference process

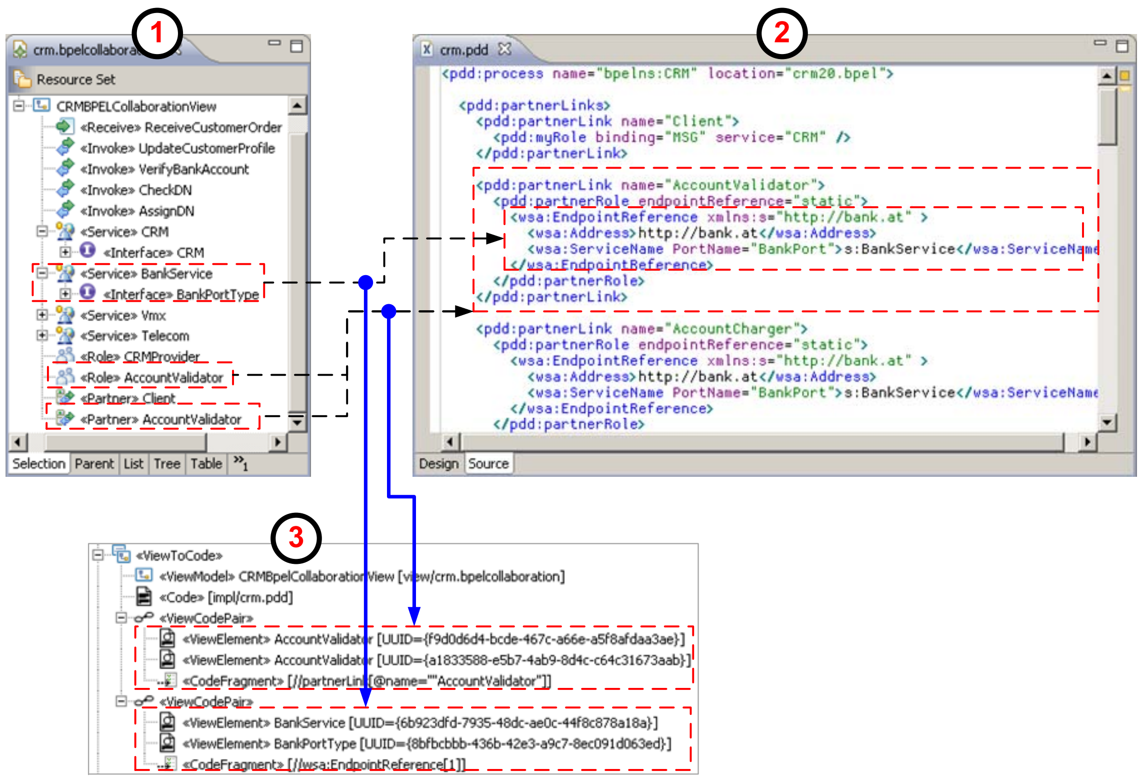
*For the sake of readability, we omitted irrelevant elements and namespace bindings in the BPEL and WSDL code and process deployment configurations



```
<bpel:process name="CRM">
    <bpel:partnerLink name="Client"
                      partnerLinkType="CRM"
                      myRole="CRMProvider" />
    <bpel:partnerLink name="AccountValidator"
                      partnerLinkType="AccountValidatorPLT"
                      partnerRole="AccountValidator"/>
  </bpel:partnerLinks>
  <bpel:variables>
    <bpel:variable name="orderInput" messageType="CRMRequest" />
    <bpel:variable name="orderOutput" messageType="CRMResponse"
  </bpel:variables>

  <bpel:sequence>
    <bpel:receive name="ReceiveCustomerOrder"
                  partnerLink="Client"
                  portType="tns:CRM"
                  operation="process"
                  variable="orderInput"
                  createInstance="yes" />
    <bpel:assign validate="no" name="DataMap1">
      <bpel:copy>
        <bpel:from>$orderInput</bpel:from>
```

```
«ViewToCode»
  «ViewModel» CRMBpelCollaborationView [view/crm.bpelcollaboration]
  «Code» [impl/crm.bpel]
  «ViewCodePair»
    «ViewElement» ReceiveCustomerOrder [UUID={c3b61567-a72a-35a9-8934-dd17f41ec223}]
    «ViewElement» ReceiveCustomerOrder [UUID={e620e6aa-7ebd-4eee-90be-af1fa9c9b7a1}]
    «CodeFragment» [/bp:receive[1]]
```

Figure 5.13: Traceability between VbMF views and process implementations



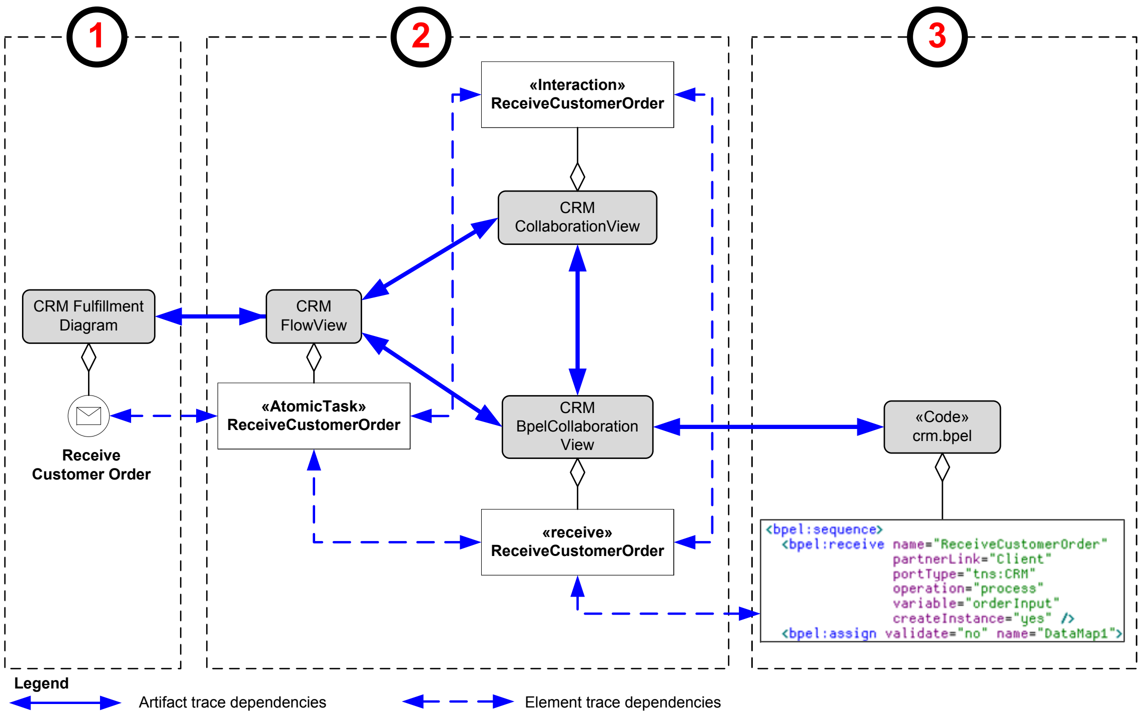
```
<pdd:process name="bpelns:CRM" location="crm20.bpel">

  <pdd:partnerLinks>
    <pdd:partnerLink name="Client">
      <pdd:myRole binding="MSG" service="CRM" />
    </pdd:partnerLink>

    <pdd:partnerLink name="AccountValidator">
      <pdd:partnerRole endpointReference="static">
        <wsa:EndpointReference xmlns:s="http://bank.at" >
          <wsa:Address>http://bank.at</wsa:Address>
          <wsa:ServiceName PortName="BankPort">s:BankService</wsa:ServiceName
        </wsa:EndpointReference>
      </pdd:partnerRole>
    </pdd:partnerLink>

    <pdd:partnerLink name="AccountCharger">
      <pdd:partnerRole endpointReference="static">
        <wsa:EndpointReference xmlns:s="http://bank.at" >
          <wsa:Address>http://bank.at</wsa:Address>
          <wsa:ServiceName PortName="BankPort">s:BankService</wsa:ServiceName
        </wsa:EndpointReference>
      </pdd:partnerRole>
```

```
«ViewToCode»
  «ViewModel» CRMBpelCollaborationView [view/crm.bpelcollaboration]
  «Code» [impl/crm.pdd]
  «ViewCodePair»
    «ViewElement» AccountValidator [UUID={f9d0d6d4-bcde-467c-a66e-a5f8afdaa3ae}]
    «ViewElement» AccountValidator [UUID={a1833588-e5b7-4ab9-8d4c-c64c31673aab}]
    «CodeFragment» [//partnerLink[@name=""AccountValidator"]]
  «ViewCodePair»
    «ViewElement» BankService [UUID={6b923dfd-7935-48dc-ae0c-44f8c878a18a}]
    «ViewElement» BankPortType [UUID={8bfbcbbb-436b-42e3-a9c7-8ec091d063ed}]
    «CodeFragment» [//wsa:EndpointReference[1]]
```

Figure 5.14: Traceability between VbMF views and process deployment descriptors

engine to deploy and execute the CRM Fulfillment process . Utilizing the extended code generators mentioned in Section 5.2.4, VbMF can generate the process deployment descriptors and establish the trace links. Figure 5.14 depicts the trace dependencies created during the course of generating process deployment descriptor required by the ActiveBPEL engine from VbMF views.

5.3.3 Leveraging VbTrace – A sample traceability path


1
2
3
«Interaction» ReceiveCustomerOrder
CRM CollaborationView
CRM Fulfillment Diagram
CRM FlowView
«AtomicTask» ReceiveCustomerOrder
Receive Customer Order
CRM BpelCollaboration View
«Code» crm.bpel
«receive» ReceiveCustomerOrder
Legend
Artifact trace dependencies
Element trace dependencies

```
<bpel:sequence>
  <bpel:receive name="ReceiveCustomerOrder"
                partnerLink="Client"
                portType="tns:CRM"
                operation="process"
                variable="orderInput"
                createInstance="yes" />
  <bpel:assign validate="no" name="DataMap1">
```

Figure 5.15: A sample traceability path from the CRM Fulfillment process design (1) through VbMF views (2) to process implementations (3)

Establishing trace dependencies alone is not sufficient for tasks like change impact analysis, change propagation, artifact understanding, and so on, it is the important factor for supporting any such tasks[148]. In this section, we examine a sample traceability path based on the traceability model created in the previous sections to illustrate how our traceability approach can support these tasks. Figure 5.15 depicts a simple traceability path from the CRM Fulfillment process design through the VbMF framework to the process implementations. The traceability path comprises the trace dependencies between the process design and VbMF views mentioned in Section 5.3.2.1 followed by the relationships among VbMF views retrieved in Section 5.3.2.2. The process implementation is reached at the end of the traceability path by using the trace dependencies between VbMF technology-specific views and process code described in Section 5.3.2.3.

Let us assume that there is a business expert working on the BPMN Modeler in order to analyze the CRM Fulfillment process functionality and change the process design. These changes must be accordingly reflected in the process implementations in BPEL and WSDL and even in process deployment configuration. Without our traceability approach, the IT experts have to look into the BPMN diagram and manipulate BPEL, WSDL code and process descriptors manually. This is time consuming, error-prone and complex because there is no explicit links between these languages. In addition, the stakeholders have to go across numerous dependencies between various tangled concerns, some of which might be not relevant to the stakeholders expertise (cf. the statistics in Table 5.1). Using our approach, the business experts can analyze and manipulate business processes by using either the process designer or the VbMF abstract views such as the FlowView, CollaborationView, InformationView, and so on, depending on their needs and knowledge. The IT experts, who mostly work on either technology-specific views or process code, can better analyze and assess coarse-grained or fine-grained implications of these changes based on the traceability path connecting the process design and implementation at different levels of granularity.

5.4 Discussion

Being extensively studied in literature and industry, dependency relationships between designs and implementations are often used for tracing through development artifacts, supporting change impact analysis, artifacts understanding, and other tasks[148]. Spanoudakis and Zisman[148] presented a summary of the state-of-the-art traceability approaches that focus on the tracing between stakeholders and requirements[55] between requirements[4,55,86,129,133,167,190], between requirements and designs[33,40,81,86,132,133], between designs presented in[40,133,167,189], between requirements and code[7,40,94,133], and between code artifacts[133]. There are only a few approaches for supporting traceability between designs (e.g., UML Class diagrams) and code[33,40,86,94,133]. Each of the aforementioned design-to-code traceability approaches, however, merely focus on specific types of dependencies, for instance, *overlap relations*[40], *evolution relations*[33,94,133], and *generalization/refinement relations*[86]. These approaches do not mention the extensibility to other types of dependencies or the ability to cover different levels of granularity of trace dependencies.

The difference of abstraction and granularity and the diversity of language syntaxes and semantics hinder the automation of establishing and maintaining the traceability between high-level artifacts, such as requirements or design specifications, and very low-level artifacts, such as executable code. There are few promising efforts on supporting automatic generation of trace dependencies that use information retrieval techniques[7,8,58,89,90,95] or rule-based traceability[92,132,149]. To the best of our knowledge, the traceability retrieved from the aforementioned approaches does not cover the difference of granularity at multiple levels of abstraction, which is the first challenge we described in Section 5.1. In addition, Ramesh and Jarke[133] and Lindvall and Sandahl[87] suggested that a traceability approach only supports the representation of different trace dependencies between artifacts, but the interpretation, analysis, and understanding of the relationships extremely depend on the stakeholders. According to his specific needs, knowledge, and experience, a stakeholder might be interested in different types of dependencies of different levels of abstraction. Most of traceability approaches described above, except the one proposed by Gotel and Finkelstein[55], have not provided adequate support for different stakeholder interests.

Recently, model-driven development (MDD)[51,113,150], which gradually gains widespread adoption in both industry and research, provides an efficient paradigm to potentially reconcile the difference of granularity at various levels of abstraction by introducing a number of intermediate (semi)-formal modeling layers, such as the platform-independent models (PIMs) and platform-specific models (PSMs)[51,113]. Each modeling layers can provide different abstractions of systems and software which are tailored to specific stakeholders' knowledge and experience. Moreover, model transformations provide better facilities for the automation of creating and maintaining traceability relationships[2,52]. A number of research approaches have exploited these advantages for establishing and maintaining traceability between development artifacts[3,18,93,107,110,178], to name but a few, in order to support reducing the gap between design and implementation.

The lightweight traceability approach TRACES can support tracing requirements across different models and levels of abstraction[3]. Based on the assumption that each artifact has a unique identifier, and code is fully generated from the models (which is hard to achieve in reality[150]) TRACES offers mechanisms for eliciting traceability links from requirements to models, i.e., PIM and PSM, and from the models to code[3]. Mäder et al.[93] analyze and classify Unified Process (UP) artifacts to establish a traceability link model that helps in (semi)-automatically establishing and verifying traceability links in Unified Process development projects along with a set of rules for management of the links. Leveraging model transformations, Naslavsky et al.[107] propose an approach for creating fine-grained traceability among model-based testing artifacts in order to support result evaluation, coverage analysis, and regression testing. Oldevik and Neple[110] present an approach for handling text-based traceability links in model-to-code transformations (M2T) (aka code generations) which is a key contribution to OMG MOF Model to Text Transformation standardization effort[115]. This approach provides a meta-model including a set of concepts for traceability between model elements and locations in code artifacts. The corresponding part of our traceability meta-model, i.e., the trace dependencies between views and code artifacts, is inspired by the M2T approach[110]. Walderhaug et al.[178] present a generic approach for traceability in MDD aiming to enhance sharing and integrating of traceability information from different development tools. The authors propose a high-level representation of the traceability process in the course of software development that provides general concepts for representing different kinds of stakeholders and artifacts used for traceability, such as trace model, trace repository, and the interactions between the stakeholders and these artifacts. Bondé et al.[18] propose an approach that offers a traceability meta-model for representing the relations between artifacts and the transformation operations associated with these relations. Once the traceability is accomplished, it then can be used to enforce the interoperability of models at different levels of abstraction, for instance, between a PIM and PSM.

Our work has the commonalities with the MDD-based traceability approaches in using traceability meta-models for (semi-)formalizing trace dependencies in order to enhance the interoperability of tools. In contrast to the related work, we introduce the combination of the *separation of concerns principles* realized by the notion of views and the *separation of abstraction levels* realized by the MDD paradigm as a better solution for supporting traceability in process-driven SOAs. We exploit the notion of views to efficiently represent different process concerns such that stakeholders are provided with tailored perspectives by view integration mechanisms according to their specific needs, knowledge, and experience. This is a significant step toward the support of adapting process representations and trace dependencies to particular stakeholder interests. In addition, the *separation of abstraction levels* offers appropriate intermediate layers to gradually bring the business experts working at high levels of abstraction close to the IT experts working at lower levels of abstraction. Using the separation of process concerns in terms of (semi-)formalized views and the view integration mechanism, the refinement between two adjacent abstraction levels can be alleviated in a better and more flexible manner. Obviously, the relationships between our modeling artifacts such as views and view elements are intrinsic and can be retrieved straightforwardly from the view models.

Leveraging these modeling concepts and mechanisms, we perform the mapping of process designs (here: BPMN) onto high-level, abstract views and process implementations (here: BPEL and WSDL) onto low-level, technology-specific views and devise a traceability meta-model that is rich enough to represent the trace dependencies from design to implementation through different abstraction levels and different granularity. Furthermore, our framework is quite open for extensibility, such as adding more traceability relationships at finer granularity with adequate specializations of the *ArtifactTrace* and the *ElementTrace*, adding more intermediate view model layer, or adding more appropriate specializations of the *TraceRationale* meta-class to support enhancing traceability reasoning or change impact analysis.

In the area of process-driven development, there are a number of approaches that define transformations between different languages[100,101,103,125,134,188]. These approaches partially provide the link between process design and implementation. However, most of these approaches focus on only one process concern, namely, the orchestration concern, and ignore other significant ones, such as collaborations, data processing, fault handling, and so on. As a consequence, each of these approaches is applicable for transforming of control structures of two specific kinds of languages, for instance, BPMN and BPEL[125,134], EPC and BPEL[103,188], and so forth. As a consequence, these approaches offer neither the extensibility to support the various process concerns, except the control flow, nor the traceability of these concerns of processes. Nonetheless, our traceability approach benefits from different algorithms described originally in those approaches for mapping the control flow of process design onto our FlowView model (cf. Scenario 1).

Table 5.2 and 5.3 present qualitative comparisons of our view-based, model-driven traceability approach, VbTrace, and a number of selected related work which are most closely related to VbTrace, such as the *MDD-based* traceability approaches that utilize model-driven paradigm with modeling layers ranging from high-level into low-level and/or exploit model transformations for traceability between design and implementation[3,93,110,178]. The comparison criteria are adapted a paper of Spanoudakis and Zisman[148].

	Support for multiple trace relations	Generation of trace relations	Relation representation	Support adaptation of stakeholder interests
TRACES by Alesky et al.[3]	TRACES offers the *Reference* or *Realize* relations between requirements and models and the *Generate* relations between models and code. *Realize* relations can be inferred from above relations.	Creation of explicit traceability links between requirements and models is the responsibility of the modeler. Traceability links can also be automatically retrieved from code generation.	TRACES does not explicitly present a formalization of trace dependencies.	Not supported yet
UP traceability by Mäder et al.[93]	This approach focuses on four trace relations: *Refine*, *Realize*, *Verify*, and *Define*.	Trace dependencies are established with adequate interventions of stakeholders of UP.	Existing UML notational elements are used for representing trace dependencies. Trace dependencies are stored in the source code as annotations.	Not supported yet
Model-based testing by Naslavsky et al.[107]	This approach concentrates on the trace relations between models and test artifacts.	The generation of test cases and relevant traceability model is currently manual.	The traceability model describes the relationships achieved in a typical model transformation and is persisted as an Ecore model.	Not supported yet
M2T by Oldevik et al.[110]	M2T focuses on the trace relations between models, i.e., PSMs, and text generated from the models, i.e., *Generate* relations.	Trace dependencies are automatically generated from model-to-code transformations.	M2T defines a meta-model providing a set of concepts for traceability between model elements and locations in code artifacts.	Not supported yet
Generic MDD traceability by Walderhaug et al.[178]	As this approach presents a high-level and generic solution to traceability for MDD, no concrete trace dependencies are considered. Thus, further efforts are required to specialize and adapt the traceability and repository models to particular needs.	This approach merely refers to other traceability approaches for achieving trace dependencies such as M2T.	A number of high-level meta-model are defined to represent general concepts of traceability, such as trace model, artifact type, trace type, etc. as well as the stakeholders interactions.	This approach offers a number of pre-defined stakeholders roles, such as developer, trace user, traceability manager, and tool supplier, who involve in the MDD paradigm but does not provide mechanisms to support the adaptation of trace dependencies to the various stakeholders interests.

(Continued on next page)

	Support for multiple trace relations	**Generation of trace relations**	**Relation representation**	**Support adaptation of stakeholder interests**
VbTrace	Support multiple relation types by the annotation of adequate *TraceRationales* (cf. Section 5.2.2).	Trace dependencies between process design and views, between views and code are accomplished (semi-)automatically, between views and view elements are retrieved straightforwardly (cf. Section 5.3.2).	(Semi-)formalized representation in terms of a rich, extensible traceability meta-model aiming at supporting interoperability and sharing of trace dependencies (cf. Section 5.2.2).	The combination of separation of concerns and separation of abstraction levels is a significant step toward support the adaptation of process representations and trace dependencies to various stakeholder interests. Moreover, the *Role* meta-class, a specialization of *TraceRationale*, can be annotated to the trace links to better support adaptation and reasoning of trace dependencies and traceability path with respect to particular stakeholders.

Table 5.2: The comparison of related work of VbTrace

	Support intermediate modeling layers	**Support for multiple granularities**	**Extensibility options**	**Tool support**
TRACES by Alesky et al.[3]	TRACES supports traceability between requirement and model elements and between model elements and code but does not mention any support for intermediate modeling layers (i.e., model and code merely have an equivalent abstraction).	Not supported	Not supported	A prototypical development environment which can be integrated with Eclipse and CodeBeamer for development and traceability. Models are stored in either XML or XMI formats.
UP traceability by Mäder et al.[93]	UP traceability focuses on traceability between four abstraction levels of UP: requirement, analysis, design, and implementation models.	UP traceability aims to support the traceability between basic UML artifacts of UP abstraction levels.	UP traceability is merely bound to UP/UML.	Not supported yet
Model-based testing by Naslavsky et al.[107]	This approach uses model-based control flow graphs and test generation hierarchy meta-model as the intermediate layers between UML sequence diagrams and testing artifacts.	This approach concentrates on fine-grained trace dependencies.	Not supported yet	Not supported yet
M2T by Oldevik et al.[110]	Not supported yet	M2T only generates relations between model elements and code blocks.	Not supported yet	Not supported yet

(Continued on next page)

	Support intermediate modeling layers	**Support for multiple granularities**	**Extensibility options**	**Tool support**
Generic MDD traceability by Walderhaug et al.[178]	This approach needs further efforts to specialize the traceability model	This approach needs further efforts to specialize the traceability model	The extensibility of this approach is potential, however, requires additional efforts.	This approach merely offers a generic, high-level traceability approach aiming at supporting sharing and integration of tools. No concrete tool supports are mentioned.
VbTrace	View-based modeling framework offers the separation of views into different intermediate modeling layers at different levels of abstraction.	The traceability meta-model is rich and extensible for supporting representation of many levels of granularity of trace dependencies (cf. Section 5.2.2)	VbTrace offers the extensibility of granularity levels and intermediate modeling layers by refining either *TraceLink*, *ArtifactTrace* or *ElementTrace*, and of the semantics of the trace relations by specializing *TraceRationle* (cf. Section 5.2.2). Moreover, VbTrace is not bound to the process concerns exemplified in chapter such as the control flow, process collaboration, and data handling, but is extensible to other concerns such as transactions, event handling, human interactions, etc.[97] and Section 3.4	A prototypical Eclipse-based integrated environment based on EMF MOF-compliant Ecore and openArchitectureWare MDD for process-driven SOA development (cf. Section 5.3.1). XML Metadata Interchange (XMI) standard[119] is utilized for model persistence, and thereby, better support traceability sharing and interoperability of MOF-compliant tools.

Table 5.3: The comparison of related work of VbTrace (cont'd)

5.5 Conclusion

Traceability support in process-driven SOAs development suffers from the challenging gap due to the fact that there is no explicit links between process design and implementation languages because of the differences of syntaxes and semantics and the difference of granularity and abstraction levels. In addition, the substantial complexity caused by various tangled process concerns and the lack of adequate tool support have multiplied the difficulty of bridging this gap. The view-based, model-driven traceability approach is our effort to overcome the issues mentioned above and support stakholders in (semi-)automatically eliciting and (semi-)formalizing trace dependencies between development artifacts in process-driven SOAs at different levels of granularity and abstraction. A proof-of-concept Eclipse-based tool support has been developed and illustrated via the CRM Fulfillment process extracted from an industrial case study.

The view-based, model-driven traceability framework presented so far lays a solid foundation for change impact analysis, artifact understanding, change management and propagation, and other activities. Our ongoing work is to complement this framework with a model repository that alleviates collaborative model-driven development and traceability sharing with different stakeholders as well as tool integrations.

Chapter **6**

Evaluation

“ *The most interesting theoretical problems come from implementing real systems.* ”

— *Leslie Lamport*

6.1 Introduction

So far we have introduced the view-based, model-driven approach, view-based reverse engineering approach, and view-based, model-driven traceability approach for process-driven SOA development, and a unified modeling/development framework – VbMF and VbTrace – to build business processes. In this chapter we show how these concepts can be utilized in reality by aligning VbMF/VbTrace with a number of industrial use cases in order to demonstrate the pragmatic use our approach.

The discussion in this chapter comprises two main parts. In the first part, we propose a scenario-driven evaluation approach, in which we instantiate VbMF/VbTrace in a number of development scenarios that cover the design time of the process life cycle. In each scenario, we demonstrate and qualitatively analyze the way that VbMF/VbTrace supports stakeholders in modeling and implementing processes. The stakeholders can apply any of these scenarios or combine some, or even all, of them according to particular context in the course of process development life cycle. The second part presents a quantitative analysis of our approach.

6.2 Scenario-driven evaluation

6.2.1 Scenario 1 – Greenfield

The CRM Fulfillment process is a part of the customer relationship management (CRM), billing, and provisioning systems of an Austrian Internet Service Provider[43]. The main business functionality of the CRM Fulfillment process is to handle a customer order of the company's bundle of Internet and telecom services including a network subscriber line (e.g., xDSL), email addresses, Web-based administration (VMX), directory number, fax number, and SIP URL for VoIP communications.

The process uses a wide variety of in-house services and services provided by various partners. The company has developed and deployed in-house services for customer relationship information management, assigning fax numbers, SIP URLs, and mail boxes, initializing VMX, and sending postal invoices to customers.

The process uses a credit bureau service provided by a third party business partner of the financial institution that acquires, stores, and protects credit information of individual and companies. The credit bureau service can verify a customer's payment account for accuracy and validity and charge the payment according to the customer's purchase order. Customer premise equipment (CPE) partners supply services for ordering and shipping home modems or routers. Telecom partners offer services for checking, assigning, and migrating customer directory numbers (DN). These services expose their functionalities in terms of WSDL interfaces that can be orchestrated using BPEL processes .

The company has decided to adopt SOA to address challenges in the telecommunication market which are the intensity of competition, a shorter time-to-market to meet a window of opportunity that cannot miss, and the constant request on change of both business environment and technology. Using an SOA approach will allow for a more rapid implementation, better integration with the functionality provided by the partners, and greater flexibility and extensibility for future changes that the business might need. A process-driven SOA is leveraged to orchestrate services to accomplish the main business functionality of the CRM Fulfillment process.

This scenario assumes the company starts developing the CRM Fulfillment process from scratch. Thus, the scenario is so-called a *greenfield*. In traditional development paradigms such as component-based, object-oriented, etc., a greenfield scenario is seldom in reality because most of companies have some prior development. Nevertheless, a greenfield scenario plays an important role in the field of process-driven SOAs and model-driven development (MDD). On the one hand, the SOA adoption of companies is merely initiative and growing, and therefore, many companies shall start a SOA-based project from the beginning. On the other hand, a greenfield scenario is a showcase for the forward engineering process development. The forward engineering development, i.e., moving from models to code, will repeatedly occur in a typical MDD paradigm[150]. The forward engineering, following up a reverse engineering, is also necessary in a re-engineering approach for re-generating certain parts of the target software and systems.

The results of this greenfield scenario are the CRM Fulfillment process implementation as well as deployment configurations used to deploy the process in a BPEL engine for enactment. The most widely-adopted process development in industry today is to utilize BPMN for designing and BPEL/WSDL for realizing the process[*]. Our VbMF approach to a greenfield is described by the forward engineering toolchain shown in Figure 3.20. In Table 6.1, we present a comparison of

[*]The WS-BPEL 2.0 standard provides the concept of abstract process as high-level representations of business processes though, it lacks suitable notations. Hence, BPMN is still a prominent process design language.

Phase	BPMN-BPEL/WSDL approach	VbMF approach
Design	Business experts design the process using BPMN diagrams.	Business experts model the process using the VbMF FlowView. Optionally, the business experts, sometimes together with IT experts, can specify some high-level details using the high-level views such as CollaborationView, InformationView, HumanView, etc.
Implementation	IT experts, sometimes together with business experts, translate the BPMN diagrams into executable process descriptions in BPEL/WSDL.	IT experts either refine the abstract views, if they exist, or create low-level, technology-specific views such as BpelCollaborationView and BpelInformationView. According to particular process functionality, IT experts might have to describe other process concerns such as transactions, event handling, and human interactions by using TransactionView, BpelEventView, and BPEL4PeopleView, respectively. After that, process implementations can be automatically generated from VbMF views.
Deployment	IT experts define deployment configurations and deploy the implemented process in a BPEL process engine	Deployment configurations are generated. The process then can be deployed in a BPEL process engine.

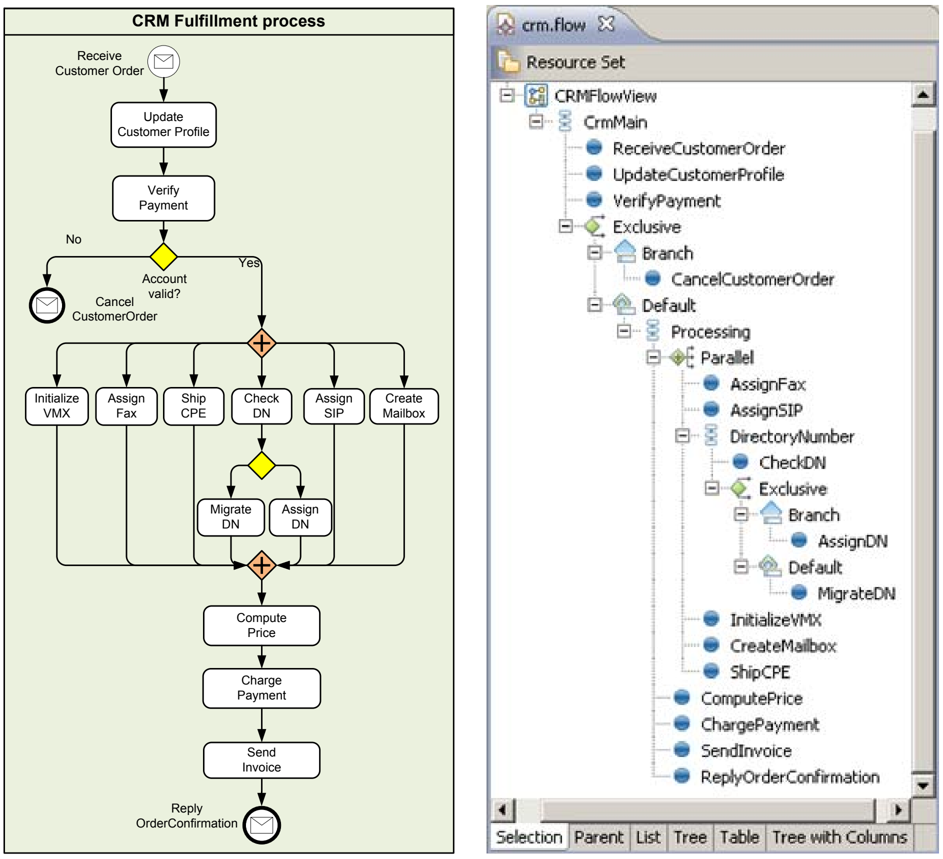
Table 6.1: Comparison of VbMF and the industry-driven approach for process development

this industry-driven approach and our VbMF approach in different process development phases: modeling, implementation, and deployment. In the subsequence sections, we illustrate and compare two approaches during the course of developing a "green-field" CRM Fulfillment process.

6.2.1.1 Modeling phase

In the modeling phase, the BPMN language is often used by business experts to describe process functionality due to its friendly notational syntax and control structures. Figure 6.1a depicts the CRM Fulfillment process design in terms of a BPMN diagram.

As we explained in Section 3.3.2, the building blocks of our FlowView are the basic patterns supported by most of existing process modeling languages. Beside these control structures, the FlowView offers the concept of *AtomicTask* which is an abstract of a human interaction, a data processing task, or an invocation of functions, services, processes and subprocesses. Note that there is no technical detail comprised in an *AtomicTask* according to the FlowView model specification (cf. Section 3.3.2). As a result, the business experts can be quickly acquainted with, and work with, the concepts and control structures provided by the FlowView model. Figure 6.1b shows a FlowView of the CRM Fulfillment



(a) **(b)**

Figure 6.1: Modeling the CRM Fulfillment process at high-level: (a) Using BPMN; and, (b) Using VbMF FlowView

process that represents the same functionality as that of the BPMN diagram depicted in Figure 6.1a.

6.2.1.2 Modeling high-level data handling and communications

After making a sketch design of the CRM Fulfillment process, the business experts, depending on their particular knowledge and experience, can also add some high-level details for describing necessary business objects or the interactions with various process partners. Figure 6.2 depicts the aforementioned BPMN diagram annotated with additional elements for this purpose. The business experts add *Data Object*s, such as Order, VerifyResult, CheckResult, and Price, to specify the data required produced by activities. Each *Data Object* is linked to an activity through an *Association* represented by a dotted line with a line arrowhead. The direction of an *Association* indicates the flow of data transfer. Moreover, a number of *Lane*s, such as CRM, CreditBureau, Sales, Provisioning, Telecom, and Customer Premise Equipment Partner are used to represent process partners. An interaction between the process and a partner is described by a *Message Flow* – a red, dashed line with an open arrowhead. Note that the CRM Fulfillment process design in BPMN constitutes all

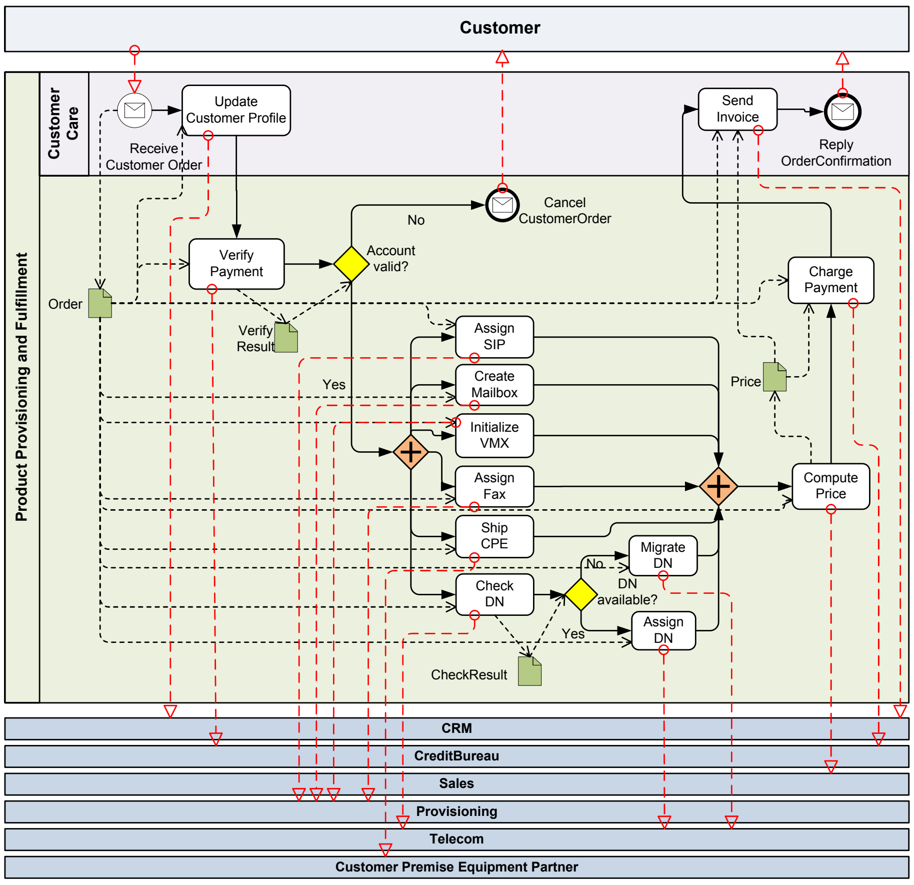


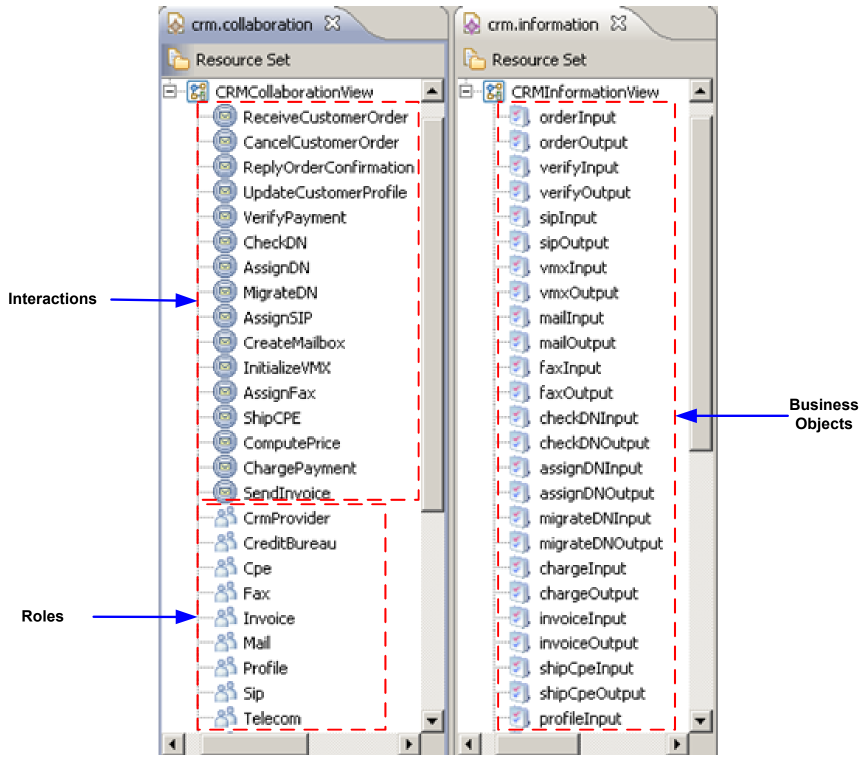
Figure 6.2: Modeling of data objects and communications with BPMN

these tangled process concerns, i.e., the control structure, data handling, message exchanges with partners, etc. The complexity of the process design increases along with the number of *DataObjects* and partners. As a consequence, the readability of the process design is significantly reduced.

Instead of polluting the process design by different tangled process concerns as that of the BPMN diagram, our approach offers separate views for modeling high-level data handling and communications, which are the *InformationView* and *CollaborationView*, respectively. In Figure 6.3, we illustrate a CollaborationView and an InformationView of the CRM Fulfillment process. The CollaborationView defines the interactions with process partners and the roles that each partner plays whilst the InformationView consists of business objects which are being exchanged with process partners or manipulated inside the process.

6.2.1.3 Modeling low-level process views

The CRM Fulfillment process design described in the BPMN language is not executable. Thus, it must be translated into a process executable language. The defacto industry standard language for implementing executable processes is BPEL[67,109]. The IT experts, sometime with the business experts' supports, shall interpret the functionality given in the BPMN design and develop a BPEL process to



crm.collaboration
Resource Set
CRMCollaborationView
ReceiveCustomerOrder
CancelCustomerOrder
ReplyOrderConfirmation
UpdateCustomerProfile
VerifyPayment
CheckDN
AssignDN
MigrateDN
AssignSIP
CreateMailbox
InitializeVMX
AssignFax
ShipCPE
ComputePrice
ChargePayment
SendInvoice
CrmProvider
CreditBureau
Cpe
Fax
Invoice
Mail
Profile
Sip
Telecom

crm.information
Resource Set
CRMInformationView
orderInput
orderOutput
verifyInput
verifyOutput
sipInput
sipOutput
vmxInput
vmxOutput
mailInput
mailOutput
faxInput
faxOutput
checkDNInput
checkDNOutput
assignDNInput
assignDNOutput
migrateDNInput
migrateDNOutput
chargeInput
chargeOutput
invoiceInput
invoiceOutput
shipCpeInput
shipCpeOutput
profileInput

Figure 6.3: Modeling high-level data handling and communications using VbMF: The CollaborationView (left) and InformationView (right)

realize such functionality. Due to the lack of explicit links between process design languages, say, BPMN, and executable languages, say, BPEL, the implementation of the CRM Fulfillment process is tedious and error-prone.

In contrast to the industry-driven approach, VbMF offers separate extension views for representing technology-specific concepts, such as BpelCollaborationView, BpelInformationView, Transaction-View, BpelEvent View, BPEL4People, etc. These technology-specific views are not disconnected from the high-level representations but rather contains concepts which are refined from the corresponding concepts in the high-level counterparts (cf. Section 3.5). Figure 6.4 presents two extension views which are BpelCollaborationView and BpelInformationView for the CRM Fulfillment process. Although process views are independently developed, the dependencies between these views are established and maintained using the view-based traceability approach described in Chapter 5.

6.2.1.4 Code generation and process deployment

Code generation (or so-called model-to-code transformation) is often used to generate recurring target, and maybe, executable, code from models. VbMF offered a template-based technique for generating process implementation in terms of BPEL and WSDL (cf. Section 3.6). Using the VbMF code generators, the stakeholders can produce not only the process BPEL code and service interfaces in WSDL but also the process deployment configurations.

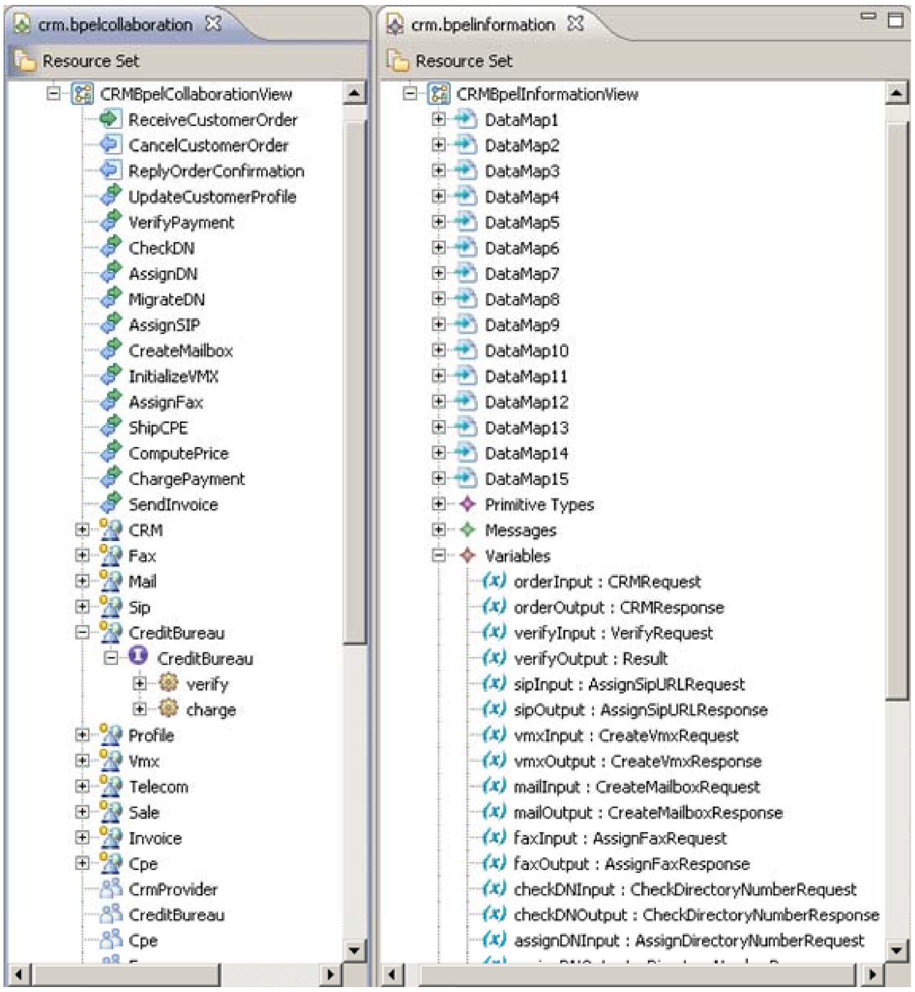


Figure 6.4: Refining the high-level views into BPEL-specific views: The BpelCollaborationView (left) and BpelInformationView (right)

```
<workflow>
  <!-- Define properties for the CRM Fulfillment process views -->
  <property name="BASEDIR" value="evaluation/at/eweb/crm/vb" />
  <property name="core" value="${BASEDIR}/crm.core" />
  <property name="flow" value="${BASEDIR}/crm.flow" />
  <property name="information" value="${BASEDIR}/crm.bpelinformation" />
  <property name="collaboration" value="${BASEDIR}/crm.bpelcollaboration" />
  <property name="output" value="output/crm" />

  <!-- Call the VbMF predefined workflow to generate BPEL/WSDL code -->
  <component file="framework/workflow/wsbpel.oaw" inheritAll="true"/>
</workflow>
```

Listing 6.1: A workflow for generating BPEL/WSDL code from the CRM Fulfillment process views

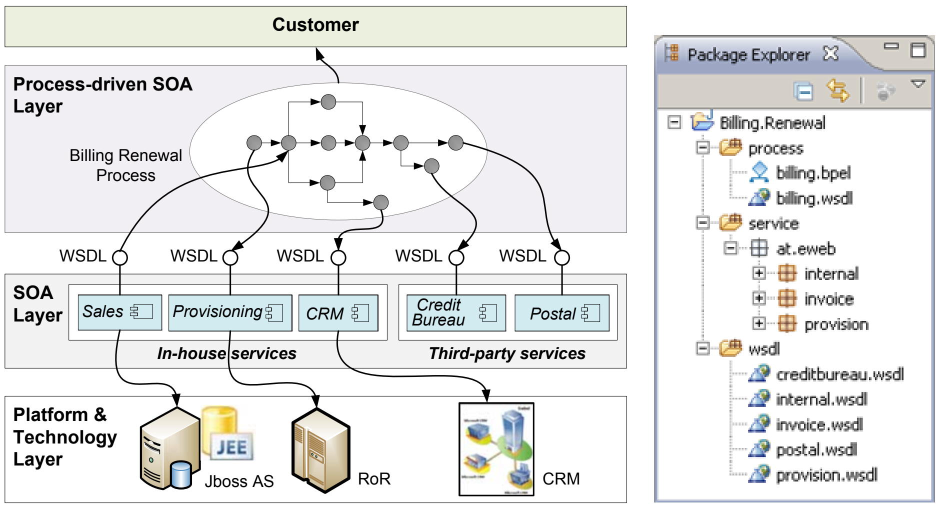
In order to utilize the VbMF code generator, the IT experts shall create a simple workflow that defines appropriate properties such as the input paths pointing to the corresponding CRM Fulfillment process views and the output path for storing the generated code and configurations (see Listing 6.1). At the end of the workflow, we only need to invoke the workflow provided by VbMF (cf. Listing 3.1).

Supporting code generation from models, i.e., the process views, is one of the biggest advantages of VbMF comparing to the industry-driven approach. Rather than either tediously interpreting and translating process designs into process implementations or directly involving in the process code, the stakeholders just work with relevant process views according to their particular interests, knowledge, and experience. Process code as well as deployment configurations are automatically generated after all, and then, can be quickly deployed for testing. This development process can repeatedly occur. By generating process code and deployment artifacts from the process views, the overall implementation time of building the CRM Fulfillment process is reduced. This way, our approach supports better business agility by shortening the development cycle and enhancing the adaptation to functional requirement changes.

6.2.2 Scenario 2 – Legacy processes

In this scenario, we exemplify a part of the billing and provisioning system of a domain registrar and hosting provider[43]. The billing and provisioning system has been designed and implemented based on the SOA paradigm. The provisioning system manages a wide variety of services including: credit bureau services (cash clearing, bank or credit card validation and payment, etc.), domain services (whois, domain registration, domain transfer, etc.), hosting services (Web and email hosting, provisioning, maintenance, etc.), and retail/wholesale services (order, upgrade, customer service and support, etc.). The core functionality of the billing and provisioning system has been realized by a number of diverged technologies such as Java/J2EE, Ruby On Rails (RoR), and a Customer Relationship Management system. The company has developed a BPEL process, namely, Billing Renewal process, in order to integrate and orchestrate its core functionality and the services provided by external business partners. Figure 6.5 presents the overall architecture of the billing and provisioning system and the existing artifacts of the Billing Renewal process.

The Billing Renewal process is triggered by a notification from the internal provisioning system when the customer's payment is due. Then, the process checks if the customer's payment can be automatically transferred or must be manually performed by the customer. In case of an automatic payment, the customer's payment is charged via credit bureau services provided by a financial business partner. Subsequently, the customer contract is extended and a postal invoice will be send to the customer's billing address. If the payment must be done manually, the process will send a first payment request to the customer. Within one month, either the payment is cleared or the second payment request will be sent. When the second request is sent, a temporary suspension of the domain, hosting, customer support and services, and customer contract is also conducted. The



(a) (b)

Figure 6.5: Existing artifacts of the Billing Renewal process: (a) Overview of the billing renewal system; and, (b) The Billing Renewal process development artifacts: BPEL, WSDL, and Java code

last payment reminding will follow the second one unless the payment arrives within two months. After three trials, if no payment is received, the customer's contract will be terminated, all support and services are stopped, the corresponding domains are unregistered, and the hosting account and space are freed. The process informs a failed billing to the customer and ends.

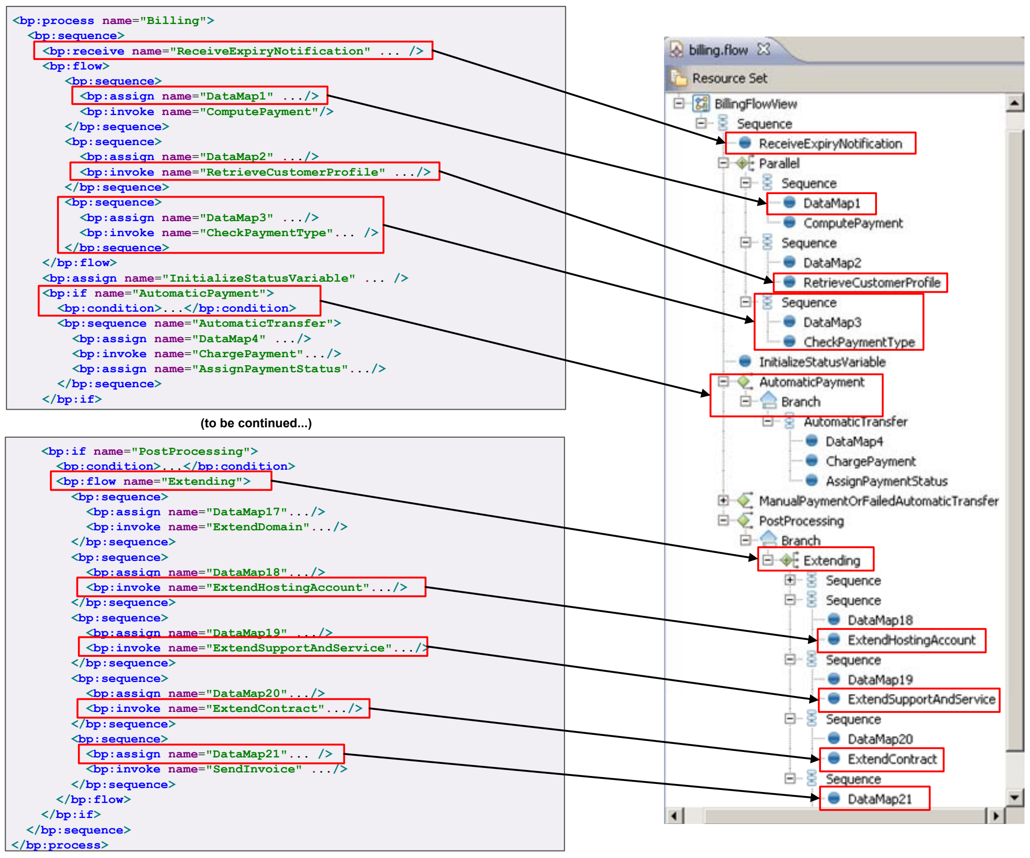
6.2.2.1 Understanding the current business status and challenges

The company wants to make the Billing Renewal process flexible and adaptable to a changing business situation, for instance, adding a new service to the existing product bundle or replacing an existing product with a newer one. Changes in business logic should be made in real time without requiring further development effort of the process. In order to satisfy these requirements, a business analyst firstly identifies requirements and analyzing these requirements from a business perspective. Then, he has to analyze the existing "as-is" Billing Renewal process. He might also want to communicate with key stakeholders to help identify gaps and areas for improvements. Based on the feedback from stakeholders, he defines necessary changes to adapt to the new requirements. In the technical landscape, the IT experts must ensure a proper translation of those requirements to process implementations.

The biggest challenge that the stakeholders confronted with is that the "as-is" process description contains too much technical details such that the business analysts hardly analyze and manipulate as well as to communicate with the other stakeholders. Moreover, the stakeholders have to go across numerous dependencies between various tangled concerns, some of which are even not suitable for their expertise and skills, in order to thoroughly understand and manipulate certain excerpts of the process description. The staekholders needs a way to diagram the current Billing Renewal process, so that they can easily identify gaps and areas for business process improvement and optimization.

As we discussed in Section 4.6, existing approaches provide limited support for extracting appropriate representations of business processes from process code, and mostly consider one process concern – the control structure. In the subsequent sections, we illustrate our view-based reverse engineering approach to support stakeholders on recovering view models from existing process implementation, say, BPEL and WSDL. These view models then can be used by the stakeholders for analyzing the business logic of the process, communicating with other stakeholders, and manipulating the process models to adapt to new business requirements. Changes made in the process views then can be propagated into lower level of abstractions or re-generated process code using the forward engineering demonstrated in the greenfield scenario.

6.2.2.2 Recovering high-level representations



```
<bp:process name="Billing">
  <bp:sequence>
    <bp:receive name="ReceiveExpiryNotification" ... />
    <bp:flow>
       <bp:sequence>
         <bp:assign name="DataMap1" .../>
         <bp:invoke name="ComputePayment"/>
       </bp:sequence>
       <bp:sequence>
         <bp:assign name="DataMap2" .../>
         <bp:invoke name="RetrieveCustomerProfile" .../>
       </bp:sequence>
       <bp:sequence>
         <bp:assign name="DataMap3" .../>
         <bp:invoke name="CheckPaymentType"... />
       </bp:sequence>
    </bp:flow>
    <bp:assign name="InitializeStatusVariable" ... />
    <bp:if name="AutomaticPayment">
      <bp:condition>...</bp:condition>
      <bp:sequence name="AutomaticTransfer">
        <bp:assign name="DataMap4" .../>
        <bp:invoke name="ChargePayment".../>
        <bp:assign name="AssignPaymentStatus".../>
      </bp:sequence>
    </bp:if>
```

(to be continued...)

```
    <bp:if name="PostProcessing">
      <bp:condition>...</bp:condition>
      <bp:flow name="Extending">
        <bp:sequence>
          <bp:assign name="DataMap17".../>
          <bp:invoke name="ExtendDomain".../>
        </bp:sequence>
        <bp:sequence>
          <bp:assign name="DataMap18".../>
          <bp:invoke name="ExtendHostingAccount".../>
        </bp:sequence>
        <bp:sequence>
          <bp:assign name="DataMap19" .../>
          <bp:invoke name="ExtendSupportAndService".../>
        </bp:sequence>
        <bp:sequence>
          <bp:assign name="DataMap20".../>
          <bp:invoke name="ExtendContract".../>
        </bp:sequence>
        <bp:sequence>
          <bp:assign name="DataMap21"... />
          <bp:invoke name="SendInvoice" .../>
        </bp:sequence>
      </bp:flow>
    </bp:if>
  </bp:sequence>
</bp:process>
```

a) Billing Renewal process code in BPEL

```
billing.flow
Resource Set
BillingFlowView
  Sequence
    ReceiveExpiryNotification
    Parallel
      Sequence
        DataMap1
        ComputePayment
      Sequence
        DataMap2
        RetrieveCustomerProfile
      Sequence
        DataMap3
        CheckPaymentType
    InitializeStatusVariable
    AutomaticPayment
      Branch
        AutomaticTransfer
          DataMap4
          ChargePayment
          AssignPaymentStatus
    ManualPaymentOrFailedAutomaticTransfer
    PostProcessing
      Branch
        Extending
          Sequence
          Sequence
            DataMap18
            ExtendHostingAccount
          Sequence
            DataMap19
            ExtendSupportAndService
          Sequence
            DataMap20
            ExtendContract
          Sequence
            DataMap21
```

b) The extracted BillingRenewal FlowView

Figure 6.6: A FlowView extracted from the Billing Renewal BPEL code

Using the view-based reverse engineering approach described in Chapter 4, the core business functionality of the Billing Renewal process is extracted and presented to the stakeholders in terms of a FlowView (see Figure 6.6b) by using the FlowView interpreter (cf. Section 4.4.1). Note that the resulting FlowView contains a "plain" business logic (i.e., an orchestration of process tasks) to achieve the business goal of the Billing Renewal process. The Billing Renewal process description in BPEL (see Figure 6.6a) are intentionally simplified to only show the elements that the FlowView interpreter examined. The other elements which are omitted also do not appear in the extracted FlowView. In other words, the resulting FlowView offers an abstract, high-level perspective of the Billing Renewal process that the stakeholders, especially the business experts, can better understand, analyze, and manipulate to adapt to new business requirements or changes.

Besides the FlowView which represents the orchestration logic of the Billing Renewal process, stakeholders can also recover other high-level representations for modeling data, communications, human interactions, etc., using the view-based interpreters provided by VbMF (cf. Section 4.4.1). These views can be used integrated to provide the stakeholders a richer view or a more thorough view of the Billing Renewal process according to their specific interests and expertise.

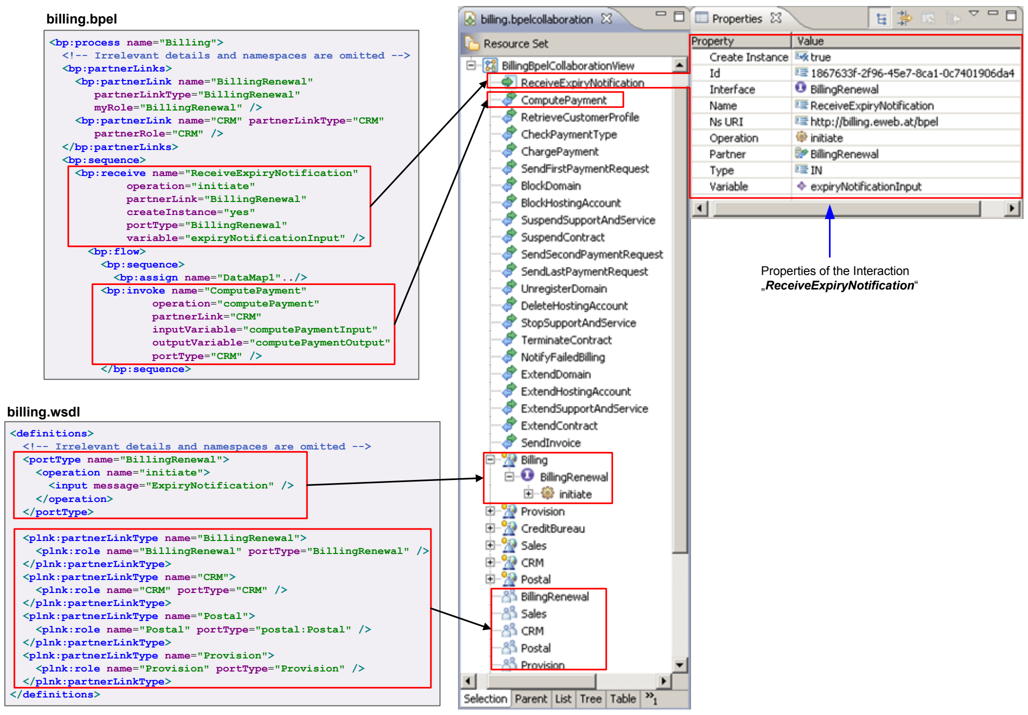
6.2.2.3 Recovering low-level representations

So far the abstract, high-level representations of the Billing Renewal process such as FlowView, CollaborationView, and InformationView are recovered from existing process implementations, i.e., the BPEL and WSDL descriptions. The existing BPEL and WSDL descriptions also embody low-level, technical representations of the processes. Therefore, VbMf can also extract technology-specific representations from process implementation in the same manner. Figure 6.7 depicts the recovering of a BpelCollaborationView and BpelInformationView using the view-based reverse engineering approach. As a high-level view and its low-level counterpart are recovered from the Billing Renewal process implementation by using the same approach, the consistency between high-level and low-level representations of a certain concern is ensured (cf. Section 4.4.2).

6.2.2.4 View manipulation and code re-generation

```
<workflow>
  <!-- Define properties for the Billing Renewal process views -->
  <property name="BASEDIR" value="evaluation/at/eweb/billing/vb" />
  <property name="core" value="${BASEDIR}/billing.core" />
  <property name="flow" value="${BASEDIR}/billing.flow" />
  <property name="information" value="${BASEDIR}/billing.bpelinformation" />
  <property name="collaboration" value="${HBASEDIR}/billing.bpelcollaboration" />
  <property name="event" value="${BASEDIR}/billing.bpelevent" />
  <property name="output" value="output/billing" />
  <!-- Call the VbMF predefined workflow to generate BPEL/WSDL code -->
  <component file="framework/workflow/wsbpel.oaw" inheritAll="true"/>
</workflow>
```

Listing 6.2: A workflow for generating BPEL/WSDL code from the Billing Renewal process views



billing.bpel

```
<bp:process name="Billing">
  <!-- Irrelevant details and namespaces are omitted -->
  <bp:partnerLinks>
    <bp:partnerLink name="BillingRenewal"
       partnerLinkType="BillingRenewal"
       myRole="BillingRenewal" />
    <bp:partnerLink name="CRM" partnerLinkType="CRM"
       partnerRole="CRM" />
  </bp:partnerLinks>
  <bp:sequence>
    <bp:receive name="ReceiveExpiryNotification"
           operation="initiate"
           partnerLink="BillingRenewal"
           createInstance="yes"
           portType="BillingRenewal"
           variable="expiryNotificationInput" />
      <bp:flow>
        <bp:sequence>
          <bp:assign name="DataMap1"../>
        <bp:invoke name="ComputePayment"
               operation="computePayment"
               partnerLink="CRM"
               inputVariable="computePaymentInput"
               outputVariable="computePaymentOutput"
               portType="CRM" />
        </bp:sequence>
```

billing.wsdl

```
<definitions>
  <!-- Irrelevant details and namespaces are omitted -->
  <portType name="BillingRenewal">
    <operation name="initiate">
      <input message="ExpiryNotification" />
    </operation>
  </portType>

  <plnk:partnerLinkType name="BillingRenewal">
    <plnk:role name="BillingRenewal" portType="BillingRenewal" />
  </plnk:partnerLinkType>
  <plnk:partnerLinkType name="CRM">
    <plnk:role name="CRM" portType="CRM" />
  </plnk:partnerLinkType>
  <plnk:partnerLinkType name="Postal">
    <plnk:role name="Postal" portType="postal:Postal" />
  </plnk:partnerLinkType>
  <plnk:partnerLinkType name="Provision">
    <plnk:role name="Provision" portType="Provision" />
  </plnk:partnerLinkType>
</definitions>
```

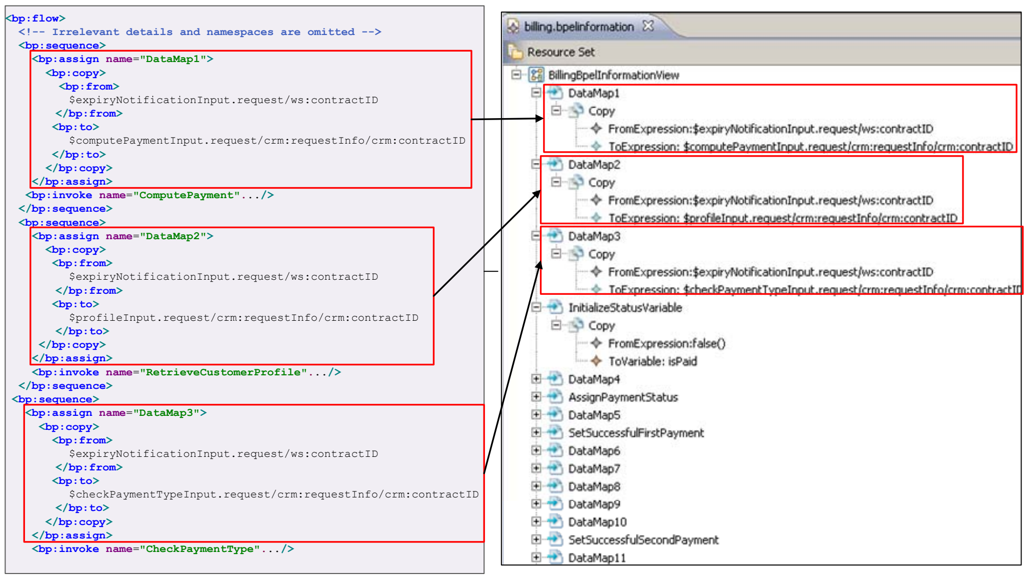
billing.bpelcollaboration

```
Resource Set
BillingBpelCollaborationView
  ReceiveExpiryNotification
  ComputePayment
  RetrieveCustomerProfile
  CheckPaymentType
  ChargePayment
  SendFirstPaymentRequest
  BlockDomain
  BlockHostingAccount
  SuspendSupportAndService
  SuspendContract
  SendSecondPaymentRequest
  SendLastPaymentRequest
  UnregisterDomain
  DeleteHostingAccount
  StopSupportAndService
  TerminateContract
  NotifyFailedBilling
  ExtendDomain
  ExtendHostingAccount
  ExtendSupportAndService
  ExtendContract
  SendInvoice
  Billing
    BillingRenewal
      initiate
  Provision
  CreditBureau
  Sales
  CRM
  Postal
  BillingRenewal
  Sales
  CRM
  Postal
  Provision
```

Properties

Property	Value
Create Instance	true
Id	1867633f-2f96-45e7-8ca1-0c7401906da4
Interface	BillingRenewal
Name	ReceiveExpiryNotification
Ns URI	http://billing.eweb.at/bpel
Operation	initiate
Partner	BillingRenewal
Type	IN
Variable	expiryNotificationInput

(a) Extracting a BpelCollaborationView from the Billing Renewal code



```
<bp:flow>
  <!-- Irrelevant details and namespaces are omitted -->
  <bp:sequence>
    <bp:assign name="DataMap1">
      <bp:copy>
        <bp:from>
          $expiryNotificationInput.request/ws:contractID
        </bp:from>
       <bp:to>
          $computePaymentInput.request/crm:requestInfo/crm:contractID
       </bp:to>
      </bp:copy>
    </bp:assign>
   <bp:invoke name="ComputePayment".../>
  </bp:sequence>
  <bp:sequence>
    <bp:assign name="DataMap2">
      <bp:copy>
       <bp:from>
          $expiryNotificationInput.request/ws:contractID
        </bp:from>
       <bp:to>
          $profileInput.request/crm:requestInfo/crm:contractID
        </bp:to>
     </bp:copy>
    </bp:assign>
    <bp:invoke name="RetrieveCustomerProfile".../>
  </bp:sequence>
 <bp:sequence>
   <bp:assign name="DataMap3">
     <bp:copy>
       <bp:from>
          $expiryNotificationInput.request/ws:contractID
        </bp:from>
       <bp:to>
          $checkPaymentTypeInput.request/crm:requestInfo/crm:contractID
        </bp:to>
      </bp:copy>
    </bp:assign>
    <bp:invoke name="CheckPaymentType".../>
```

billing.bpelinformation

```
Resource Set
BillingBpelInformationView
  DataMap1
    Copy
      FromExpression:$expiryNotificationInput.request/ws:contractID
      ToExpression: $computePaymentInput.request/crm:requestInfo/crm:contractID
  DataMap2
    Copy
      FromExpression:$expiryNotificationInput.request/ws:contractID
      ToExpression: $profileInput.request/crm:requestInfo/crm:contractID
  DataMap3
    Copy
      FromExpression:$expiryNotificationInput.request/ws:contractID
      ToExpression: $checkPaymentTypeInput.request/crm:requestInfo/crm:contractID
  InitializeStatusVariable
    Copy
      FromExpression:false()
      ToVariable: isPaid
  DataMap4
  AssignPaymentStatus
  DataMap5
  SetSuccessfulFirstPayment
  DataMap6
  DataMap7
  DataMap8
  DataMap9
  DataMap10
  SetSuccessfulSecondPayment
  DataMap11
```

(b) Extracting a BpelInformationView from the Billing Renewal code

Figure 6.7: Extracting low-level representations of the Billing Renewal process

In the previous steps, the view-based reverse engineering approach is used to recover views of the Billing Renewal process which are more relevant for stakeholders, from existing process code. However, to make this usable in practice, changes on the views should lead to corresponding changes on process code. The propagation of change is performed using VbMF code generators. The developers create a workflow, shown in Listing 6.2, which is similar to that of the CRM Fulfillment process mentioned above. In this way, any change in the process views can be automatically reflected in the process implementation.

6.2.3 Scenario 3 – Reuse

Most of companies opt for using or adapting existing development artifacts at every opportunity to do a new project. Reusability is the degree of using the existing software artifacts instead of developing new ones. The benefit of reusing artifacts is that companies can increase productivity, reduce IT cost, shorten time-to-market, and spend less time for testing and debugging. In Section 4.6, we argued that the reusability of process descriptions is hardly achievable because of the divergence of process modeling and development languages, and is mostly done by using the " *copy-and-paste*" approach. We suggested a better way for enhancing the reusability in process development that is to map process descriptions into VbMF view models and then use VbMF to develop processes (cf. Section 4.6). We demonstrate how VbMF support a better reusability of process development artifacts through a use case Order Handling process.

6.2.3.1 Understanding the current business status and requirements

In this scenario, we illustrate how VbMF can support stakehodlers in reusing existing views to develop an Order Handling process. The Order Handling process is based upon the purchase order handling system in which Internet customers can order the company's products via the Web site. Figure 6.8) depicts the core functionality of the Order Handling process in terms of a BPMN diagram.

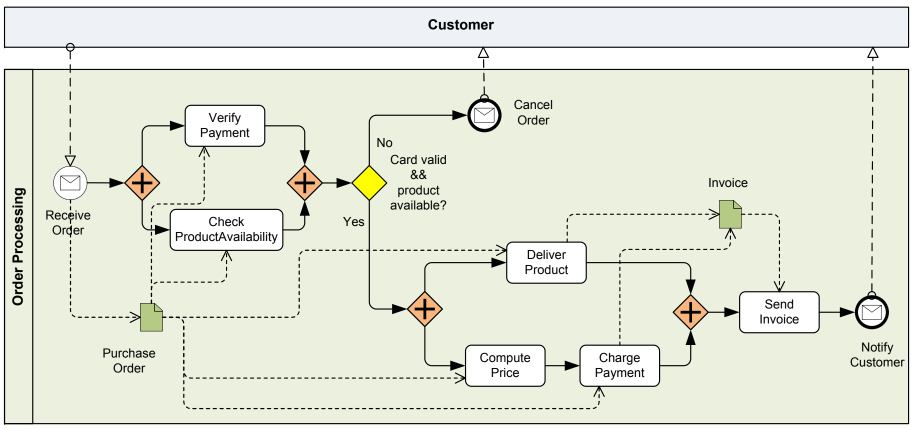


Figure 6.8: Overview of the Order Handling process

The Order Handling process starts when an Internet customer submits a purchase order. The payment account provided by the customer will be checked with regard to its completeness and correctness. In parallel, the company also checks whether the ordered product is on stock. If either the customer's payment account is invalid or the product is out of stock, the process will notify the customer and ends. Otherwise, the process continues to handle the customer's purchase order. On the one hand, the total price for the ordered product, which includes the product's price and the shipment's price, is computed, and the customer's account is accordingly charged. On the other hand, a shipment service is contacted for delivering the ordered product to the customer's address. When the charging and the shipment have been performed successfully, a postal invoice is sent and the process ends.

The company wants to reuse existing artifacts to develop the Order Handling process rather than starting from scratch. The business experts, after analyzing the business requirements, identify the functionality required by the Order Handling process and start designing the Order Handling FlowView. The IT experts, maybe together with the business experts, align the Order Handling process's functionality with the previously developed artifacts and services. In the end, they identified a number of fragments of process models and services with similar functionality exist across the enterprise. For instance, the Order Handling process contains a task that charges customer payment by invoking the services provided by the credit bureau partner which is similar to the *ChargePayment* task of the Billing Renewal process (cf. Scenario 2). Therefore, this task should be reused in the Order Handling process rather than being developed again.

6.2.3.2 Using VbMF to enhance the reusability of development artifacts

Figure 6.9 depicts an example that demonstrates how stakeholders reuse the existing *ChargePayment* in the Order Handling process modeling. For the sake of readability, the scenario is presented in terms of UML Object Diagrams[116]. On the right-hand side, we show the CollaborationView and BpelCollaborationView of the Billing Renewal process where the *ChargePayment* activity is defined at high-level and low-level of abstract, respectively. In the Billing Renewal CollaborationView, *ChargePayment:Interaction* – an instance of the *Interaction* class – has relationships with three other objects: *CreditBureau:Partner*, *CreditBureau:Interface*, and *charge:Operation*. The *ChargePayment:Interaction* object is refined in the Billing Renewal BpelCollaborationView by the *ChargePayment:Invoke* object – an instance of the *Invoke* class. The *ChargePayment:Invoke* object has two more associations with the *chargePaymentInput:VariableReference* and *chargePaymentOutput:VariableReference* objects.

In order to reuse the *ChargePayment* activity of the Billing Renewal process, the stakeholders perform two steps:

1. Create a corresponding *ChargePayment:AtomicTask* in the Order Handling FlowView as shown in the right-hand side of Figure 6.9.

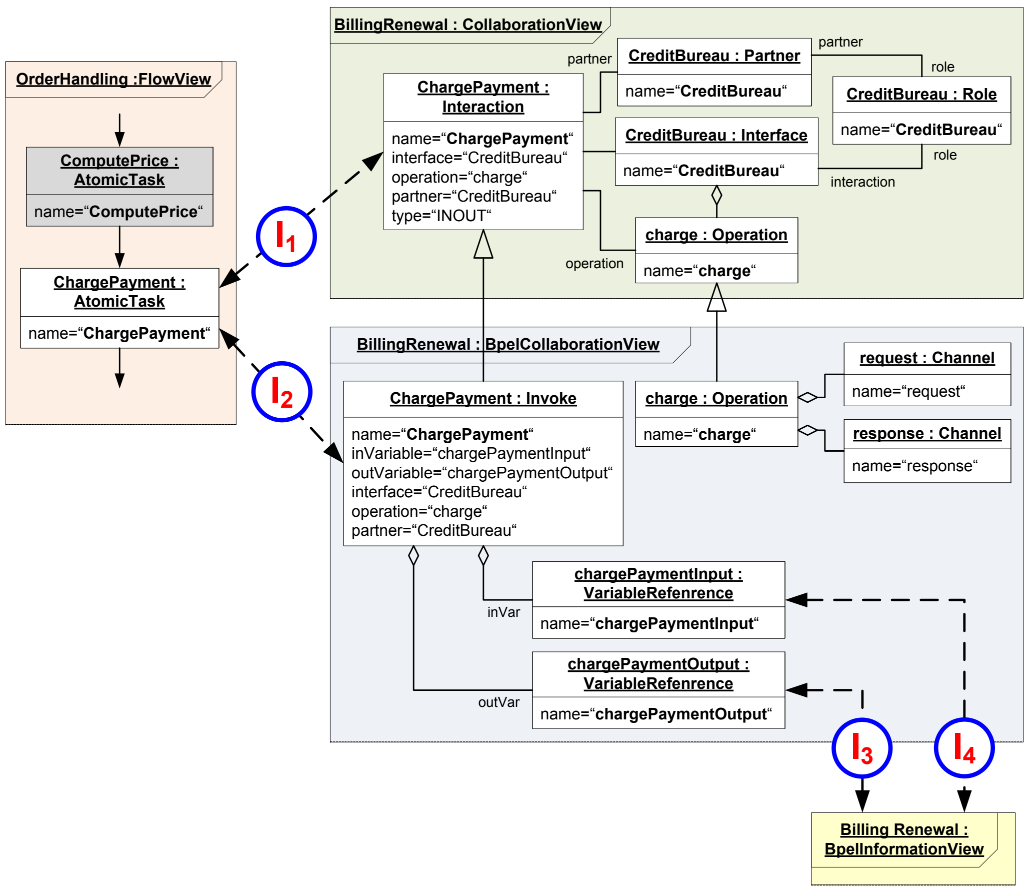


Figure 6.9: Illustration of the reusability: Reusing by referencing the *Charge Payment* element of the Billing Renewal Process in the Order Handling process development

2. Perform one of the following tasks:
   (a) Explicitly define either an integration point $\mathbf{I_1}$ between the *ChargePayment:AtomicTask* and the *ChargePayment:Interaction* or $\mathbf{I_2}$ between the *ChargePayment:AtomicTask* and the *ChargePayment:Invoke*.
   (b) Explicitly specify the CollaborationView and BpelCollaborationView of the Billing Renewal process are input views of the Order Handling process. As VbMF supports view integration by name-based matching (cf. Section 3.5), the aforementioned integration points can be implicitly resolved by VbMF tooling, for instance, the code generators.

A question might be raised at this point, that is "*how's about the relationships between either the* ChargePayment:Interaction *or* ChargePayment:Invoke *and other views of the Billing Renewal process?*". For instance, the *ChargePayment:Invoke* has assciations with *chargePaymentInput:VariableReference* and *chargePaymentOutput:VariableReference* objects which are instances of the *VariableReference* class. In the Billing Renewal process, the actual definitions of these objects belong to the BpelInformationView. Therefore, these objects are part of the integration points $\mathbf{I_3}$ and $\mathbf{I_4}$, respectively, between the BpelCollaborationView and

BpelInformationView of the Billing Renewal process. In this situation, the stakeholders can choose one of two possible approaches:

1. Reuse the existing integration points between the BpelCollaborationView and BpelInformationView of the Billing Renewal process: The stakeholders can gain more benefit of reusability but they have to analyze the subsequent dependencies of the reused objects in the BpelInformationView. In addition, these subsequent dependencies also require extra effort to maintain view synchronization when making any change in the reused views.
2. Create new objects in the Order Handling BpelInformationView and re-define $\mathbf{I_3}$ and $\mathbf{I_4}$ to reference to those objects rather than reusing the existing integration points. Although no benefit of reusability gained, there is also no binding to the Billing Renewal BpelInformationView. That is, no extra effort is needed for understanding the subsequent dependencies or maintaining view synchronization.

In summary, the separation of concerns principle realized in VbMF has isolated tangled process concerns by different domain of interests – (semi-)formalized views. The concept of integration point enables the flexibility of partially or totally reusing existing artifacts to develop new processes. As we explained during the development of the Order Handling process, each element of a certain process view is a potential reusable artifact. This way, VbMF enables the stakeholders to gain more reuse of existing process development artifacts.

6.2.4 Scenario 4 – Traceability

During the course of modeling and developing business processes in the previous scenarios, the view-based, model-driven traceability approach (VbTrace) described in Chapter 5 is utilized to (semi-)automatically establish and maintain the trace dependencies in both forward and reverse engineering toolchain. Section 5.3 explained in detail how VbTrace supported stakeholders in establishing and maintaining trace dependencies in various phases of modeling and developing the CRM Fulfillment process (cf. Scenario 1). We summarize the involvement of VbTrace in the process-driven SOA development life cycle:

Forward engineering The development life cycle starts with designing processes, for instance, by using BPMN or VbMF high-level views and ends with implementing and deploying processes, for instance, by using BPEL/WSDL. Scenario 1 described in Section 5.3.2.1 is applied for the traceability between process designs and VbMF high-level views. Then, Scenario 2 (cf. Section 5.3.2.2) is for the traceability between process views. Finally, the traceability between process views and generated code and configurations is established by applying Scenario 3 (cf. Section 5.3.2.3).

Reverse engineering The development life cycle goes on the other way around. That is, existing process artifacts, for instance, process designs and code, are interpreted to extract high-level and low-level process views. For the mapping of process artifacts into VbMF views, Scenario 1 (cf. Section 5.3.2.1) and 3 (cf. Section 5.3.2.3) are used in the beginning. During the reverse engineering, the integration points, i.e., the relationships between views, can be deducted from the relationships between corresponding elements of existing process artifacts. As a result, the traceability between process views can be established.

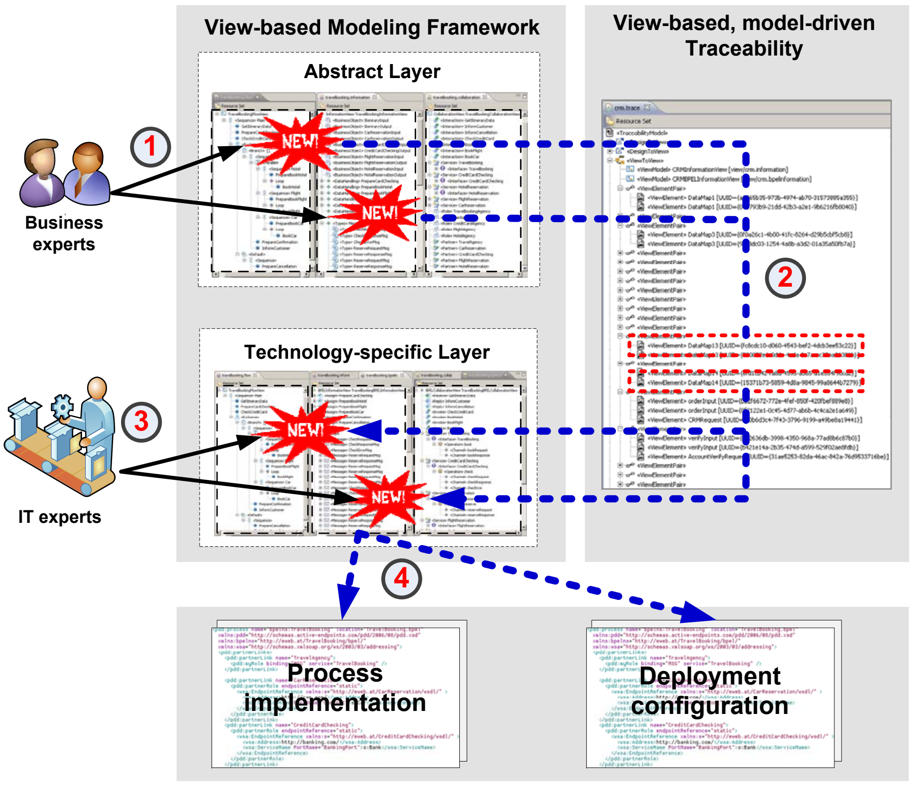


Figure 6.10: Using VbMF to quickly adapt to business requirement changes

6.2.5 Scenario 5 – Maintainability

Faced with business and regulatory turbulence, companies need to respond to change, opportunities and threats more rapidly than ever before. Companies that have a more agile and dynamic IT that can respond quickly to new regulations, customer trends, business opportunities, requirements, and threats will have a competitive advantage. Thus, the process models should enable a quicker reaction on business changes in the IT by quickly manipulating process models to develop new capabilities. Changes in process models should be made in real time without requiring further development effort of the process.

In the subsequent sections, we explain two strategies that stakeholders can utilize in order to adapt to a certain change requirement at either business level or IT level, respectively. Nonetheless, the stakeholders can adequately combine these strategies in any order to deal with the change requirements at both levels at the same time. Figure 6.10 depicts the steps the stakeholders can perform in VbMF to deal with changes at business level. These steps are compared to the industry-driven approach in Table 6.2.

Phase	**BPMN-BPEL/WSDL approach**	**VbMF approach**
Process design changing	Business experts analyze the process design in terms of BPMN diagrams in order to identify the functionality that should be changed in order to adapt to new requirements, and then, manipulate the process design. This is likely a difficult task because the process design often comprises numerous tangled concerns as shown in Figure 6.2.	The business experts analyze the "as-is" processes through the VbMF process views in the abstract layer. As we explained above, the process views in the abstract layer provide concepts that the business experts are familiar with. Thus, the business experts can quickly identify the functionality that should be changed in order to adapt to new requirements.
Implementation changing	IT experts, sometimes together with business experts, propagate the changes made in the BPMN diagram into executable process descriptions in BPEL/WSDL. This task is tedious and error-prone because there is no explicit link between the process design and implementation.	The impact of changes made by the business experts can be determined through the trace dependencies established and maintained by VbTrace (cf. Scenario 4). Based on these trace dependencies, the IT experts can accordingly modify the low-level, technology-specific views to reflect the high-level changes.
Deployment	Deployment configurations might also be, mostly manually, adjusted by IT experts in order to properly redeploy the modified process in a BPEL process engine.	Deployment configurations are automatically generated. The process then can be deployed in a BPEL process engine.

Table 6.2: Comparison of VbMF and the industry-driven approach for process adaptation

Not only changes at business level but also those in technology fields have forced companies to rethink the way to respond to changes. Sooner or later, it's likely that companies will be confronted with a challenge of implementing new technology in the software and systems. A recent survey reported by Gartner predicts that *"more than 50% of users will be dissatisfied with the slow rate of IT change in their enterprises by 2013, up from 30% in 2008"*[72]. As our approach exploited the model-driven development paradigm – a realization of the *separation of abstraction levels* principle, the technology-specific views are separated from the high-level counterparts. As a result, technological changes mostly affect the low-level, technology-specific layer of VbMF.

In Figure 6.11, we demonstrate a scenario in which the stakeholders, particularly the IT experts, deal with technological changes. The IT experts might have to modify the existing low-level view models or create new ones to represent the new technologies. In addition, the transformation templates can also be adjusted or newly created for generating process implementation and/or deployment configuration in new technologies. For instance, the first implementation of VbMF aimed at supporting BPEL version 1.1[67]. Later on, the BPEL version 2.0 is approved. We have to

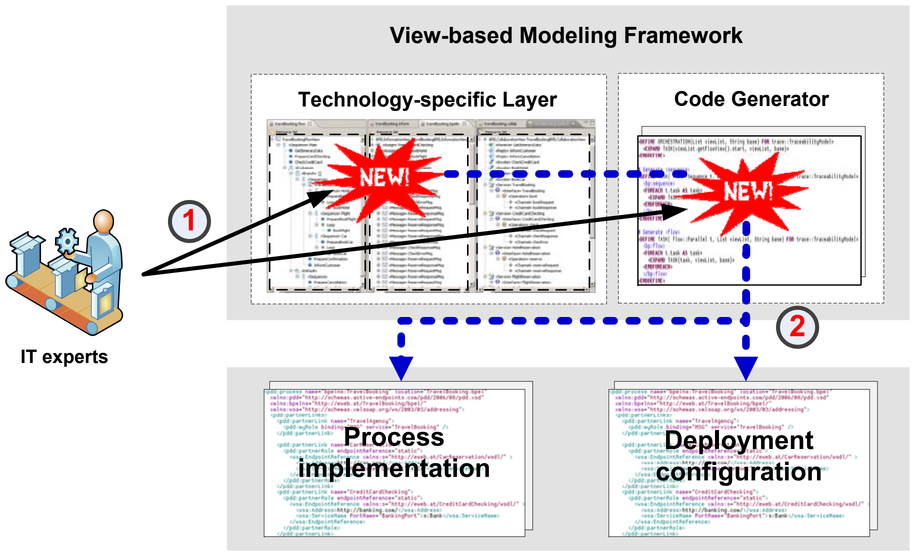
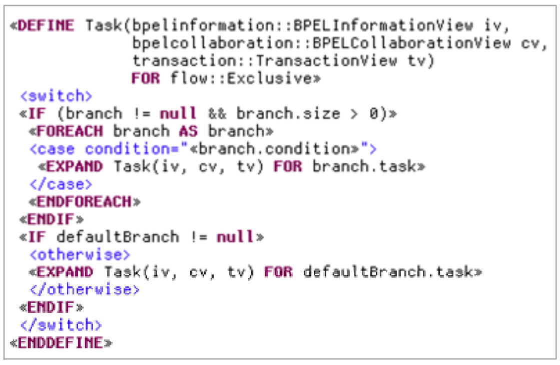
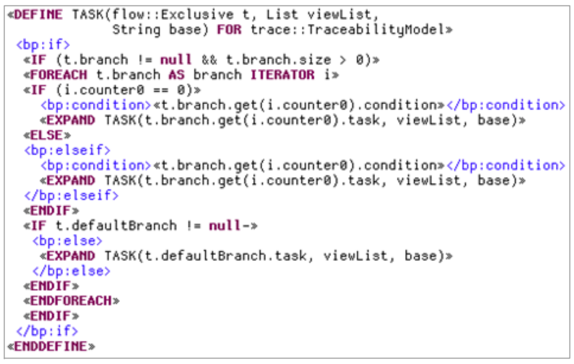


Figure 6.11: Using VbMF to adapt to technological changes

support stakeholders in quickly migrating to the new BPEL 2.0 standard. Figure 6.12 depicts the modification of the existing transformation templates in order to generate process implementation in the new standard.



```
«DEFINE Task(bpelinformation::BPELInformationView iv,
              bpelcollaboration::BPELCollaborationView cv,
              transaction::TransactionView tv)
              FOR flow::Exclusive»
 <switch>
 «IF (branch != null && branch.size > 0)»
  «FOREACH branch AS branch»
  <case condition="«branch.condition»">
   «EXPAND Task(iv, cv, tv) FOR branch.task»
  </case>
  «ENDFOREACH»
 «ENDIF»
 «IF defaultBranch != null»
  <otherwise>
  «EXPAND Task(iv, cv, tv) FOR defaultBranch.task»
  </otherwise>
 «ENDIF»
 </switch>
«ENDDEFINE»
```



```
«DEFINE TASK(flow::Exclusive t, List viewList,
             String base) FOR trace::TraceabilityModel»
 <bp:if>
  «IF (t.branch != null && t.branch.size > 0)»
  «FOREACH t.branch AS branch ITERATOR i»
  «IF (i.counter0 == 0)»
    <bp:condition>«t.branch.get(i.counter0).condition»</bp:condition>
    «EXPAND TASK(t.branch.get(i.counter0).task, viewList, base)»
  «ELSE»
  <bp:elseif>
    <bp:condition>«t.branch.get(i.counter0).condition»</bp:condition>
    «EXPAND TASK(t.branch.get(i.counter0).task, viewList, base)»
  </bp:elseif>
  «ENDIF»
  «IF t.defaultBranch != null-»
   <bp:else>
    «EXPAND TASK(t.defaultBranch.task, viewList, base)»
   </bp:else>
  «ENDIF»
  «ENDFOREACH»
  «ENDIF»
 </bp:if>
«ENDDEFINE»
```

(a) An excerpt of templates for generating BPEL 1.1 **(b)** An excerpt of templates for generating BPEL 2.0

Figure 6.12: Adjusting transformation templates for generating process code in new technology

6.2.6 Summary

The distinct characteristics of our view-based, model-driven approach for process-driven SOA development illustrated in the course of the aforementioned scenarios, such as separation of concerns, separation of abstraction levels, code generation support, adaptability, extensibility, reusability, traceability, and maintainability, are summarized in Table 6.3.

	Sep. of concerns	Sep. of abst. levels	Code generation	Adaptability	Extensibility	Reusability	Traceability	Maintainability
Scenario 1	✔	✔	✔	✔	✔			
Scenario 2	✔	✔	✔	✔				
Scenario 3	✔	✔				✔		
Scenario 4	✔	✔					✔	
Scenario 5	✔	✔		✔			✔	✔

Table 6.3: Summary of the scenario-driven evaluation

6.3 Quantitative analysis

Software metrics have been proposed and used for some time[29,45,54,145]. Essentially, software metrics deals with the measurement of the software product and the process by which it is developed[105]. In the context of our work, we mostly concentrate on the *product metrics* rather than the *process metrics*[105]. The product metrics are measures of the modeling and development artifacts, for instance, process views, process design and implementation, deployment configurations, etc., at any stage of overall development time. In this section, we present a qualitative analysis of important quality properties that support for assessment of our view-based, model-driven approach for process-driven SOA development, including the metrics of complexity, reusability, and separation of concerns.

6.3.1 Complexity

There are a number of metrics attempting to quantify the complexity of software[23,29,45,105], e.g. number of lines of code[45], McCabe complexity metric[98], Chidamber-Kemerer metrics[29], and so on. The most widely-used metric is Line of Code (LOC). LOC is typically used to measure the complexity of a software product by counting the number of lines, except blank lines and comments, in the source code[*]. This way, LOC can offer an estimation of the amount of effort for developing a software product or programming productivity. Lange pointed out the challenges in defining model size metrics that inherent to UML in particular and the MDD paradigm in general[85]. He also suggested an approach for measuring model size based on the four dimensions of Fenton and Pfleeger[45]. The complexity used in this section is a variant of Lange's model size metrics[85] extended to support specific concepts of process-driven SOA development and the MDD paradigm. According to Fenton and Pfleeger[45], Lange's metric is of cognitive complexity that rather reflects the perception and understanding of a certain model from a modeler's point of view.

Definition 6.1 (Absolute size metric). *The absolute size of a model is the number of the model's elements.*

Table 6.4 summarizes the complexity of the use cases investigated throughout our work, which are the Billing Renewal Process (BRP), CRM Fulfillment Process (CFP), Order Handling Process (OHP),

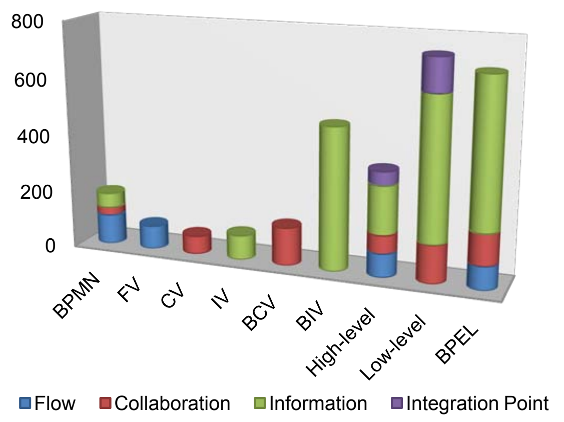
[*] Hence, LOC is somehow replaced by KNCSS – thousands of lines of non-commented source statements

Use case	Process design (BPMN)			VbMF(Hi)			VbMF(Lo)		I(x,y)		Process impl. (BPEL/WSDL)		
	Flow	Coll.	Data	FV	CV	IV	BCV	BIV	I_{high}	I_{low}	Flow	Coll.	Data
BRP	104	29	51	81	63	85	132	492	48	117	81	109	510
CFP	42	23	43	49	74	78	131	535	31	88	49	132	549
OHP	29	38	22	29	36	44	65	285	17	46	29	62	292
TBP	30	17	23	33	33	43	56	261	17	40	33	56	266

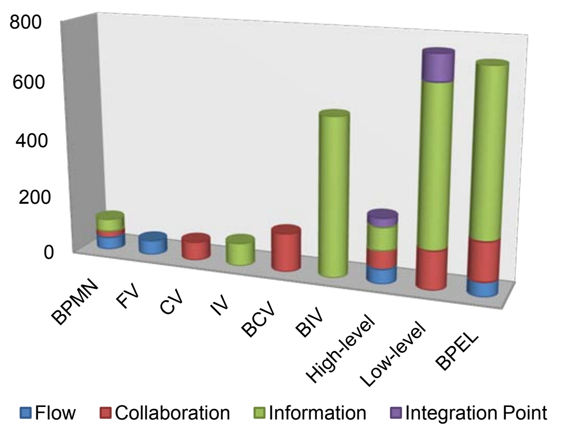
Table 6.4: The complexity of process descriptions and VbMF views

and Travel Booking Process (TBP). For each process, we measure the absolute size metric of basic concerns, including the control flow, collaboration, and data handling, of the process design (i.e., BPMN diagrams) and process implementations (i.e., BPEL/WSDL descriptions). Note that these concerns of process design and implementation are not naturally separated but rather scatted and tangled. For the sake of comparison, we measure the absolute size metric of the process design and implementation with respect to the corresponding concepts of the process concerns provided by our view-based approach. These absolute sizes are then compared to those of VbMF high-level and low-level process views, such as the FlowView, CollaborationView (CV), InformationView (IV), BpelCollaborationView (BCV), and BpelInformationView (BIV), of each use case. For the sake of readability and comparison purpose, we group the elements into corresponding process concerns which are flow, collaboration, and information concerns. $\mathbf{I(x,y)}$ denotes the number of integration points between process views in which $\mathbf{I_{high}}$ is the amount of integration points between the FlowView and other high-level process views, such as the CollaborationView and InformationView, and $\mathbf{I_{low}}$ is the amount of integration points between the FlowView and other low-level views, such as the BpelCollaborationView and BpelInformationView. Note that the number of integration points between VbMF views are considered as the cost of the separation of concerns principle realized in VbMF. The size complexity metrics are concerned with different quality properties of software and system. In general, the higher the size complexity, the harder it is to analyze and understand the system[45,141]. For instance, the absolute size metric measures the model complexity in terms of the number of model elements. The greater the number of a model's elements and relationships, the more difficult for the stakeholders to understand the model. The more model elements and relationships, the harder the stakeholders can identify the elements that must be changed in the course of evolution tasks or understand relevant functionalities during the course of reuse tasks[141]. Figure 6.13 visualizes the comparisons of the complexity of process designs, implementations, and VbMF process views with respect to the basic concerns: the control-flow, collaboration, and information concern.

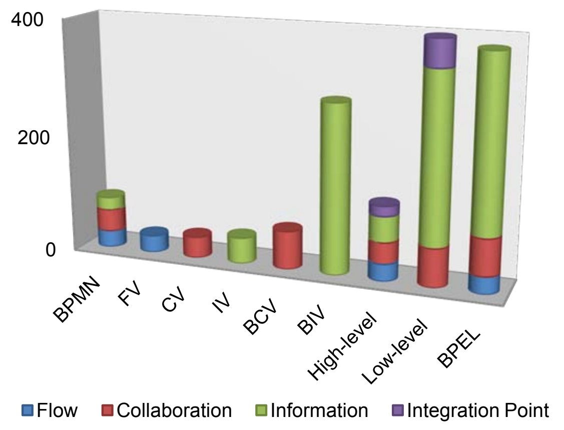
The results show that the complexity of each of VbMF high-level views is lower than that of the process design and the complexity of each of VbMF low-level views is lower than that of the process implementation. Those results prove that our approach has reduced the complexity of process descriptions by the notion of (semi-)formalized views. We also measure a high-level representation of process by using an integration of VbMF abstract views and a low-level representation of process by using an integration of VbMF technology-specific views. The numbers say that the complexity of



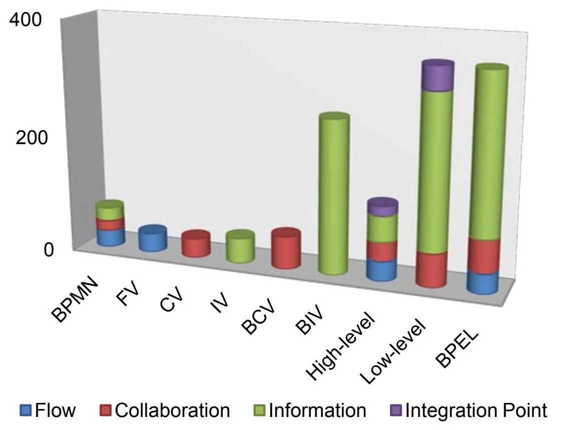
(a) Billing Renewal Process



(b) CRM Fulfillment Process



(c) Order Handling Process



(d) Travel Booking Process

Figure 6.13: Comparison of the complexity of processes and VbMF views

the high-level representation is comparable to that of process design while the complexity of the low-level representation is comparable to that of the process implementation. Because we use view integration mechanisms for combining views, the overhead of integration points occurs in both aforementioned integrated representations.

6.3.2 Reusability

The IEEE Standard Glossary of Software Engineering Terminology[68] defines reusability as:

Definition 6.2 (Reusability)**.** *The degree to which a software module or other work product can be used in more than one computer program or software system.*

In the context of an MDD paradigm, we refine this definition as *the degree to which a model can be used in more than one software or system*. As such, the reusability of a model can be measured by *the amount of reusable model elements and the model itself which can be used in more than one software or system without any changes or with small adaptations.*

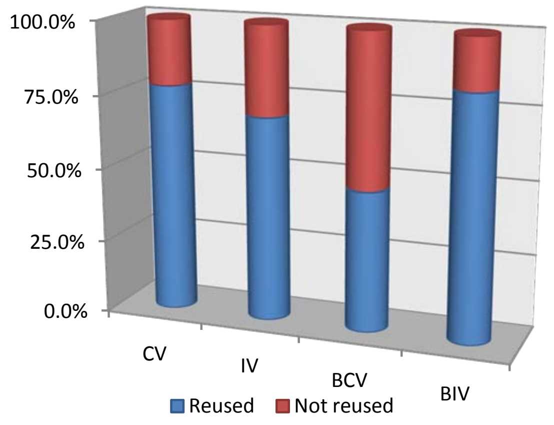
We illustrated in Scenario 3 (cf. Section 6.2.3) that each element of VbMF process views, except the Core Model and the FlowView, can be a potentially reusable artifact. Reusing a Core Model of a process is likely unreasonable because this model mostly contains process-specific meta-data, such as meta-information of the process itself, of services required or provided by the process, and of the views that model the process. A FlowView purely contains a control flow or a fragment of a control flow that defines the business logic, i.e., the execution order of process tasks, in order to achieve a particular business goal. Note that detailed descriptions of process tasks, for instance, invoking a service, transforming data objects, are not included in the FlowView but others such as (Bpel)CollaborationView and (Bpel)InformationView. Therefore, reusing an existing FlowView to develop a new process is possible but inefficient. Nonetheless, a FlowView can be reused as the documentation of an "*as-is*" process that can be referenced, or even used as a skeleton, for developing new processes. For this reason, we do not measure the reusability of the Core Models and FlowViews of the use cases.

The reusability of VbMF in process-driven SOA development is denoted by how VbMF views of a process can be potentially reused to develop other processes. We measure the reusability of VbMF views in each use case by analyzing and estimating the amount of *potentially reusable elements* (E_R) of each views. R_R denotes effective ratio of reuse, i.e., $R_R = E_R/E$, where E is the total number of model elements of the corresponding view. We present this metric of VbMF process views in Table 6.5. Figure 6.14 visualizes the degree of "*reuse*" and "*not reused*" of VbMF views for each use case.

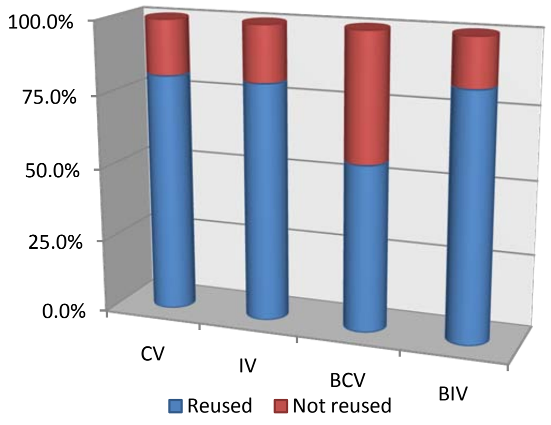
Use case	CV			IV			BCV			BIV		
	E_R	E	RR(%)	E_R	E	RR(%)	E_R	E	RR(%)	E_R	E	RR(%)
BRP	49	63	77.78	59	85	69.41	63	132	47.73	407	494	82.39
CRM	60	74	81.08	63	78	80.77	74	131	56.49	448	537	83.43
OHP	29	36	80.56	36	44	81.82	36	65	55.38	238	286	83.22
TBP	27	33	81.82	33	43	76.74	33	56	58.93	219	260	84.23
Average			**80.31**			**77.19**			**54.63**			**83.32**

Table 6.5: Measures of the reusability of process models in VbMF

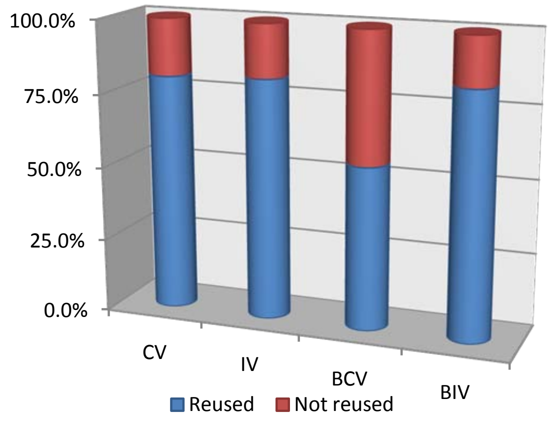
The ratio of reuse also reflects the tendency of integration of VbMF views. That is, *AtomicTasks* of the FlowView are integrated with the corresponding elements of the CollaborationView and InformationView such as *Interaction* and *Data Handling*, or elements of the BpelCollaborationView and BpelInformationView, such as *Receive*, *Reply*, *Invoke*, and *Assign*. In addition, a number of elements of the (Bpel)CollaborationView have references to corresponding elements of (Bpel)InformationView whilst none of the (Bpel)InformationView's element depends on other views' elements. As a result, the ratio of reuse of the (Bpel)InformationView is much higher than that of the (Bpel)InformationView. The ratios of reuse of high-level views are higher than that of low-level ones because the abstract concepts are more reusable than the technology-specific counterparts. The average degrees of reuse



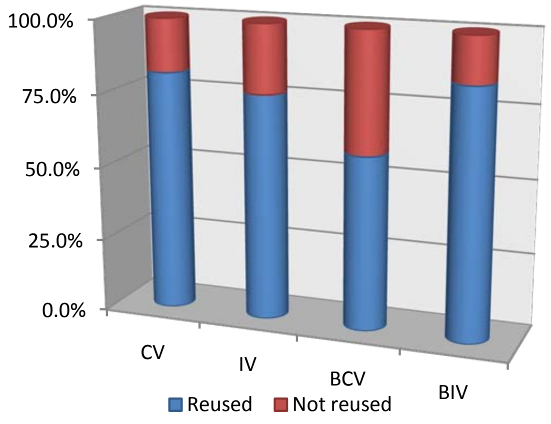
(a) Billing Renewal Process



(b) CRM Fulfillment Process



(c) Order Handling Process



(d) Travel Booking Process

Figure 6.14: The potential ratio of reuse of process views

over four use cases are very promising: **80.31**% for the CollaborationView (CV), **77.19**% for the InformationView (IV), **54.63**% for the BpelCollaborationView (BCV), and **83.32**% for the BpelInformationView (BIV).

6.3.3 Separation of concerns

Separation of concerns is a well-established principle in software engineering in which software or system is separated into distinct features in order to minimize the overlap in functionality[35,53,152]. A concern is any piece of interest or focus in a software or system. Concerns are realized in software development using various abstractions provided by languages, methods, and tools. For instance, aspect-oriented software development (AOSD) paradigm introduces the notion of aspects as a new abstraction and offers a new methodology for populating aspects and related components (classes, methods, etc.). Sant'Ana et al.[141] propose a number of metrics for assessing the separation of concerns in aspect-oriented software development[78] such as *Concern Diffusion over Components* (CDC), *Concern Diffusion over Operations* (CDO), and *Concern Diffusion over LOC* (CDLOC).

Hoffman and Eugster exploit the notion of explicit joint points (EJPs) to reduce obliviousness and report an empirical analysis on modularity and obliviousness[60]. Sant'Anna et al. note that the metrics CDC, CDO and CDLOC represent *the degree to which a single concern maps to software elements*. A lower value of these metrics will make software engineers easier to understand and maintain software designs[141]. Moreover, the separation of concerns is also a predictor of understandability, maintainability, reusability because localized, isolated concerns are better understood, manipulated, and reused than scattered, tangled ones[141].

The notion of views exploited in our approach is a realization of the separation of concerns principle. The biggest difference between our approach and AOSD approaches, as explained in Section 3.8, is the level of abstraction considered in each approach. Most of AOSD approaches for process-driven SOA development and other fields focus at a low level of abstraction, e.g., code level. Nonetheless, we share a common conceptual foundation – the separation of concerns principle. Thus, we use a variant of these metrics extended to support concepts of process-driven SOAs and MDD.

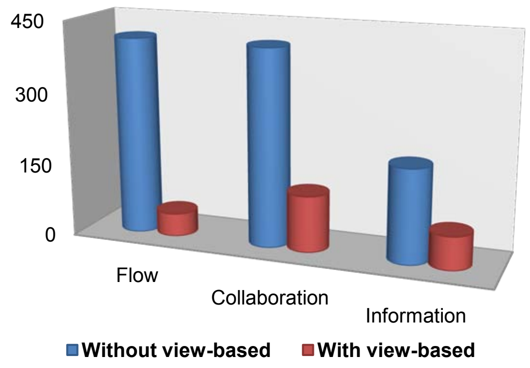
Definition 6.3 (Process-driven Concern Diffusion – PCD)**.** *PCD of a process concern is a metric that counts the number of elements of other concerns which are either tangled in that concern or directly referenced by elements of that concern.*

Use case	Flow			Collaboration			Information		
	Process	VbMF	Reduce (%)	Process	VbMF	Reduce (%)	Process	VbMF	Reduce (%)
BRP	411	48	88.32	409	117	71.39	195	69	64.62
CFP	398	31	92.21	407	88	78.38	176	57	67.61
OHP	212	17	91.98	221	46	79.19	93	29	68.82
TBP	175	17	90.29	186	40	78.49	85	23	72.94

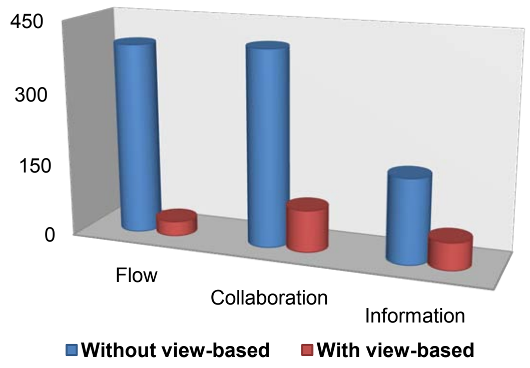
Table 6.6: Measures of process-driven concern diffusion

Process-driven Concern Diffusion (PCD) metric of a concern estimates the degree that a certain process concern scattered and tangled with the others. The higher PCD metric of a concern, the more difficult it is for the stakeholders to understand and manipulate the concern. Table 6.6 gives the results for the PCD metric of the process descriptions and VbMF views in the aforementioned use cases. We measure the PCD metric of the process descriptions represented in BPEL and WSDL with respect to different process concerns such as the control-flow, collaboration, and information. Figure 6.15 visualizes the comparison of PDC metric of process descriptions and VbMF view models for each use case.

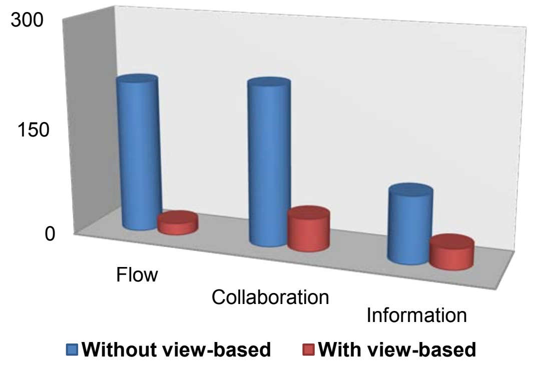
A process description often comprises numerous tangled process concerns. VbMF, by contrast, enables the stakeholders to deal with the process through separate view models and integration points. For instance, a process control-flow is described by a BPEL description that often includes many other concerns such as service interactions, data processing, transactions, and so on. As a result, the diffusion of the control-flow concern of the process description is higher than that



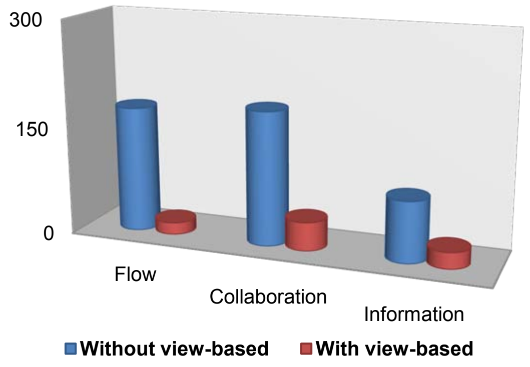
(a) Billing Renewal Process



(b) CRM Fulfillment Process



(c) Order Handling Process



(d) Travel Booking Process

Figure 6.15: Comparison of the Process-driven Concern Diffusion metric (lower is better)

of the VbMF FlowView. The results show that the separation of concerns principle exploited in our view-based, model-driven approach has significantly reduced the scatter and tanglement of process concerns. Note that the control flow is the central notion of process-driven SOAs which specifies the core business logic of processes. Thus, it is more important that we achieved a significant decrement of the diffusion of the control-flow approximately **88.32**%–**92.21** %, which denotes a better understandability and maintainability of the core functionality of processes. For other concerns, our approach is also shown to significantly reduce concern diffusions approximately **71.39**%–**79.19**% for the collaboration concern, and **64.62**%–**72.94**% for the information concern (see Table 6.6), and therefore, improve the understandability, reusability, and maintainability of business processes.

Chapter 7

Conclusion

It is good to have an end to journey toward; but it is the journey that matters, in the end.

— *Ursula K. Le Guin*

In this chapter, we summarize the work reported in this dissertation by revisiting the research problems presented in Chapter1 and consolidating the key contributions achieved. The extent to which contributions address the research problems is reviewed and discussed. Finally, we present our visions about future research directions.

7.1 Contribution summary

The initial goal in conducting this work was to investigate two major issues that have not been solved yet in existing approaches for process-driven SOAs. *First*, the process descriptions consist of various tangled concerns, such as the control flow, data handling, service invocations, transactions, event handling, etc. As the number of services and processes involving in a business process increase, the complexity of developing and maintaining the process is quickly multiplied. This complexity inheres in different phases of process development ranging from design/modeling to implementation, deployment, and maintenance. *Second*, there exists a huge divergence of process modeling and development languages due to the difference of language syntaxes and semantics, the difference of granularity at different abstraction levels, and the lack of explicit links between high-level, e.g., process designs) and low-level representations, e.g., process implementations.

In this dissertation, we presented a view-based, model-driven approach for process-driven SOAs for addressing aforementioned challenges. In general, the novelty of our approach is twofold. *Firstly*, we exploit the notion of views – a realization of the separation of concerns principle – to separate process descriptions into different (semi-)formalized representations. This way, we can aid the stakeholders mastering the *horizontal dimension*, i.e., dealing with the complexity of different process concerns. *Secondly*, we utilize the MDD paradigm – a realization of the separation of abstraction levels principle – to align process views into different abstraction levels including a high-level, platform-neutral layer and a low-level, technology-specific counterpart. This way, our approach provides stakeholders the ability of working with suitable levels of abstractions according to their specific expertise and interests. Moreover, the artifacts developed by different stakeholders can be brought together by view integration mechanisms to form a richer view or a more thorough view of processes. On other words, our approach supports the stakeholders in mastering the *vertical dimension*, i.e., bridging the gap between abstraction levels. In the subsequent sections, we discuss particular aspects of our contributions.

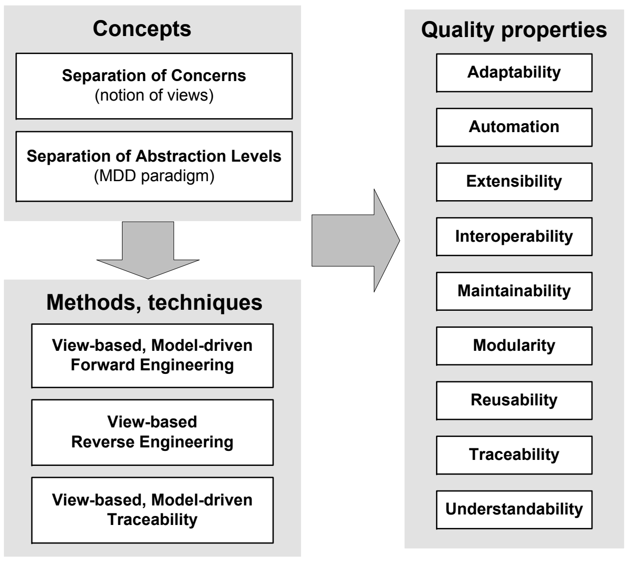

Concepts
Separation of Concerns (notion of views)
Separation of Abstraction Levels (MDD paradigm)
Methods, techniques
View-based, Model-driven Forward Engineering
View-based Reverse Engineering
View-based, Model-driven Traceability
Quality properties
Adaptability
Automation
Extensibility
Interoperability
Maintainability
Modularity
Reusability
Traceability
Understandability

Figure 7.1: Summarization of our key contributions

7.1.1 Methodology

Though this dissertation work, we contribute a new software development methodology in which the principle of separation of concerns and the principle of separation of abstraction levels are combined to shape a modeling and development framework for process-driven SOAs. The important outcomes of this combination are the various (semi-)formalized views tailoring different perspectives of process models and different abstraction levels for accommodating the difference of expertise, knowledge, and interests of the stakeholders involving in process development. The resulting framework, including a number of tools and techniques such as view integration, view extension, view-based reverse engineering, and view-based traceability, supports stakeholders in modeling and developing software and systems, in terms of business processes, in an agile, extensible, and flexible manner in both top-down and bottom-up approaches. Our approach have been realized and instantiated via different scenarios that investigates a number of industrial use cases. A qualitative analysis conducted in these use cases proves that our approach achieves many pragmatic and promising results.

7.1.2 Quality properties of process-driven SOA development

In the existing approaches to process-driven SOA development, the tanglement of concerns and the divergence of process modeling and development languages impair many important quality properties of business process development. Our view-based, model-driven approach aims at addressing these challenges in order to reconcile these quality properties, which are:

- ***Adaptability***: Appropriate representations of process models adapted to particular stakeholders' interests and expertise can be achieved by choosing relevant abstraction levels and/or integrating relevant views to produce a richer or a more thorough view of processes.
- ***Automation***: Our approach enhances the automation of process development at different points: generating executable code from process views, extracting high-level and low-level representations from existing process descriptions, and (semi)-automatically establishing and maintaining the trace dependencies between development artifacts.
- ***Extensibility***: Our approach can be expanded to represent additional process concerns by adding a *New-Concern-View* model that might inherit basic concepts of the Core model or existing views and provide new concepts for capturing the new concern. We call these *horizontal extensions*, i.e., extensions along the horizontal dimension (cf. Section 3.5). On the other hand, the vertical dimension can also be extended. An existing view at a certain level of abstraction can be refined down to a lower level by using the view extension mechanism (cf. Section 3.5). Thus, we call these *vertical extensions*.
- ***Interoperability***: The huge divergence of process modeling languages deteriorates the interoperability of different process representations in these languages. We potentially reconcile the interoperability by supporting the mapping of different process descriptions into VbMF using the reverse engineering approach and maintaining the trace dependencies occurring during the mapping.
- ***Maintainability***: Software maintenance is the modification of software in order to correct faults, to improve some qualities, or to adapt to a new requirement. The key factors that might affect software maintenance are: high complexity, limited understanding, and limited support for traceability and impact analysis. Our approach helps overcoming these factors by using separate (semi-)formalized views to reduce complexity and concern diffusions, using adequate levels of abstraction and tailored or integrated views to enhance understanding, and using traceability approach to support traceability and change impact analysis.
- ***Modularity***: The separation of concerns principle exploited in our approach, as explained in Chapter 3 and qualitatively proved in Chapter 6, enhances the modularity of process descriptions, and therefore, reduces the complexity and the diffusion of process concerns.

- ***Reusability***: In our approach process descriptions are separated into different views representing distinct process concerns. The integration of views are performed via name-based matching. Therefore, stakeholders can better reuse existing VbMF elements or views by explicitly establishing relevant integration points rather than the tedious, error-prone "copy-paste" method.

- ***Traceability***: Dependency relationships between artifacts are crucial in software development because they strongly support artifact tracing, change impact analysis, evolution management, and other activities. Our view-based, model-driven traceability approach supports stakeholders in (semi-)automatically establishing and maintaining different kinds of trace dependencies at different granularity acquired during the process development life cycle. As a result, our traceability approach plays the role of an intermediate bridge that offers stakeholders the ability of leveraging the traceability meta-data for tracing artifacts, analyzing the impact of a certain change in process models, or managing the evolution of the process development.

- ***Understandability***: Understanding the functionality of products, for instance, software or systems, is essentially prerequisite to the subsequent steps such as product development, maintenance and evolution. Thus, the understandability of a product affects some other quality properties, such as the reusability and maintainability, of that product. The understandability, in turns, is influenced by other properties such as the complexity, adaptability, and traceability. As we explained so far, our approach has provided appropriate supports for stakeholders to tackle the root of these causal relationships, i.e., to reduce the complexity, and enhance the adaptability and traceability of process-driven SOAs.

7.2 Future work

So far our view-based, model-driven approach has produced many encouraging results in addressing the major issues in process-driven SOA development. Besides, there are some interesting ideas raised during the course of our research but have not exploited further in the scope of this dissertation for some reasons.

7.2.1 Enhancing view integration mechanisms

In order to support the stakeholders in achieving a richer or a more thorough view from the existing ones, we proposed the view integration mechanism. In VbMF, view integration mechanism is realized by using a name-based matching algorithm (see Section 3.5). This algorithm is based on an assumption that the modelers shall use the same name for different occurrences of a concept or an artifact. For instance, if there is a process task appearing in the FlowView under the name of *Receive Customer Order*, then its occurrence, or in other words, its definition, in the CollaborationView should be accordingly named *Receive Customer Order* such that they can be properly integrated. Although the algorithm is simple, it is effectively used at the view level (or model level) because from a modeler's point of view in reality, it makes sense, and is reasonable, to assign the same name to the modeling entities that pose or share the same functionality and semantics. Nevertheless, improving the view integration mechanism toward further exploiting class hierarchical structures or ontology-based structures along with the existing algorithm will help stakeholders gaining more preciseness and flexibility. For instance, utilizing an ontology-based approach can raise the view integration to the conceptual level in which views can be integrated based on the similarity of concepts or relevant relationships between concepts.

7.2.2 Integration with model-driven repositories

For better modeling, developing, reusing, and maintaining business processes using our approach, a model repository is also important. A model repository is a single point of access where development artifacts can be stored, retrieved, managed, and shared among different projects and stakeholders easily and efficiently. There are a number of existing efforts both in research and industry for software development repositories such as revision control systems[71,153]. These approaches can support stakeholders in storing, sharing, and tracking changes, and controlling over changes to software source code. Unfortunately, they are not model-aware. This is, the first-class element in these approaches is merely a source document. In the context of the MDD paradigm, a model is the first-class citizen. A model contains a number of elements along with the relationships between those elements such as generalize, depend, use, etc. One or many models can be persisted in particular serialization format. For instance, a UML model[116] can be serialized to a file in XMI format[119]. The exiting revision control systems merely treat the XMI serialization of that UML model as a normal document rather than a UML model. Thus, a repository should support storing, retrieving, versioning, and sharing models and relevant meta information of models. In some recent works, we observed that this direction has gained many attentions in both research and industry. France et al. proposed a repository for MDD (ReMoDD) that will contain artifacts that support research and education in model-driven development (MDD)[50]. The NetBeans Metadata Repository is an industry effort which supports storing and sharing models based on MOF[117] and utilizes XMI[119] as the common interchange format. Oliveira et al.[111] introduced a version control system, namely, Odyssey-VCS, that deals with the complex models used by UML-based CASE tools. Odyssey-VCS also enables the configuration of both the unit of versioning and unit of comparison for each specific project, respecting the different needs of the diverse development scenarios[111]. Recently, Holmes et al. propose a promising approach, Model-Aware Service Environment (MORSE), that supports managing MDD projects and identifying and reflecting on models, model elements and their relationships[63]. In our prototype, we utilized a simple model repository in Eclipse Modeling Framework[38]. Better integration our framework with model-driven repositories mentioned above is a significant step towards a collaborative model-driven development environment.

7.2.3 Tool support

Due to the limitation of resources, we have just developed a prototypical view-based modeling framework to illustrate our approach rather than implementing a full-fledged development tooling. Nonetheless, our implementation is based on Eclipse – a decent and popular integrated software development environment. Our concepts have been realized on top of the Eclipse Modeling Framework[38] – a MOF-compliant modeling framework and code generation facility for building tools and other applications – and openArchitectureWare[122] – one of the leading open source tools for model-driven software development. As a result, our view-based modeling framework supports current (de facto) standards that support software modeling and interoperability of models and tools, such as MOF/EMOF[117] and XMI[119]. Nonetheless, there are still many rooms for improvement, for instance, enhancing tool support for view integration mechanisms, traceability, repository integration, and a friendly, unified user-interface providing adequate workbench for process-driven SOA development stakeholders. These tool supports enable stakeholders fully employed the key concepts and contributions of our approach.

Bibliography

[1] ActiveEndpoints. ActiveBPEL Engine. http://www.activevos.com/community-open-source.php, 2008. (accessed 2007/01/02).

[2] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, and Y. Shaham-Gafni. Model traceability. *IBM System Journal: Model-Driven Software Development*, 45(3), 2006.

[3] M. Aleksy, T. Hildenbrand, C. Obergfell, and M. Schwind. A Pragmatic Approach to Traceability in Model-Driven Development. In *PRIMIUM*, 2008.

[4] I. Alexander. SemiAutomatic Tracing of Requirement Versions to Use Cases - Experience and Challenges. In *TEFSE'03: 2nd International Workshop on Traceability in Emerging Forms of Software Engineering*, 2003.

[5] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer, 2004.

[6] 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.

[7] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo. Recovering Traceability Links between Code and Documentation. *IEEE Trans. Softw. Eng.*, 28(10):970–983, 2002.

[8] G. Antoniol, G. Canfora, A. de Lucia, and G. Casazza. Information Retrieval Models for Recovering Traceability Links between Code and Documentation. In *ICSM '00: Proceedings of the International Conference on Software Maintenance (ICSM'00)*, page 40, Washington, DC, USA, 2000. IEEE Computer Society.

[9] Apache. Apache ODE (Orchestration Director Engine). http://ode.apache.org, 2008. (accessed 2007/02/01).

[10] B. Axenath, E. Kindler, and V. Rubin. An Open and Formalism Independent Meta-Model for Business Processes. In *Proc. of the Workshop on Business Process Reference Models*, pages 45–59, 2005.

[11] B. Axenath, E. Kindler, and V. Rubin. AMFIBIA: A Meta-Model for the Integration of Business Process Modelling Aspects. *International Journal of Business Process Integration and Management*, 2(2):120–131, 2007.

[12] H. Bär, M. Bauer, O. Ciupke, S. Demeyer, S. Ducasse, M. Lanza, R. Marinescu, R. Nebbe, O. Nierstrasz, M. Przybilski, T. Richner, M. Rieger, C. Riva, A.-M. Sassen, B. Schulz, P. Steyaert, S. Tichelaar, and J. Weisbrod. *The FAMOOS Object-Oriented Reengineering Handbook*. SCG FAMOOS, Oct. 1999.

[13] M. Beisiegel, H. Blohm, D. Booz, M. Edwards, O. Hurley, S. Ielceanu, A. Miller, A. Karmarkar, A. Malhotra, J. Marino, M. Nally, E. Newcomer, S. Patil, G. Pavlik, M. Raepple, M. Rowley, K. Tam, S. Vorthmann, P. Walker, and L. Waterman. SCA Service Component Architecture Assembly Model Specification V1.00. http://www.osoa.org/download/attachments/35/SCA_AssemblyModel_V100.pdf?version=1, Mar. 2007.

[14] T. J. Biggerstaff. Design Recovery for Maintenance and Reuse. *IEEE Computer*, 22(7):36–49, 1989.

[15] K. Blinco, T. Grisby, A. Laird, O. O'Neill, V. Srikanth, and C. Smythe. Adoption of service oriented architecture (SOA) for enterprise systems in education: Recommended practices. Technical report, IMS Global Learning Consortium, May 2009.

[16] BMPI. Business Process Modeling Language. http://www.bpmi.org/downloads/BPML1.0.zip, Nov. 2002. (accessed 2007/02/08).

[17] R. Bobrik, M. Reichert, and T. Bauer. View-Based Process Visualization. In *BPM*, volume 4714/2007, pages 88–95. Springer, 2007.

[18] L. Bondé, P. Boulet, and J.-L. Dekeyser. *Traceability and Interoperability at Different Levels of Abstraction in Model-Driven Engineering*, pages 263–273. Applications of Specification and Design Languages for SoCs. Springer Netherlands, 2006.

[19] T. L. Booth. *Sequential Machines and Automata Theory*. John Wiley & Sons, Inc., New York, Jan. 1967.

[20] A. Brown. An Introduction to Model Driven Architecture - Part I: MDA and today's systems. http://www.ibm.com/developerworks/rational/library/3100.html, Feb. 17 2004. (accessed 2009/02/17).

[21] CA Wily. Techweb SOA study results. http://www.ca.com/files/SupportingPieces/cmp-global-survey_196383.pdf, Dec. 2009.

[22] C. Cappelli, J. C. S. P. Leite, T. Batista, and L. Silva. An aspect-oriented approach to business process modeling. In *EA '09: Proceedings of the 15th workshop on Early aspects*, pages 7–12, New York, NY, USA, 2009. ACM.

[23] J. Cardoso, J. Mendling, G. Neumann, and H. A. Reijers. A discourse on complexity of process models. In *Business Process Management Workshops, BPM 2006 International Workshops, BPD, BPI, ENEI, GPWW, DPM, semantics4ws, Vienna, Austria, September 4-7, 2006, Proceedings*, pages 117–128, 2006.

[24] A. Charfi and M. Mezini. Aspect-Oriented Web Service Composition with AO4BPEL. In *Proc. of the 2nd European Conference on Web Services (ECOWS)*, volume 3250 of *LNCS*, pages 168–182. Springer, Sept. 2004.

[25] A. Charfi and M. Mezini. An aspect-based process container for BPEL. In *AOMD '05: Proc. of the 1st workshop on Aspect oriented middleware development*, New York, NY, USA, 2005. ACM Press.

[26] A. Charfi and M. Mezini. AO4BPEL: An Aspect-oriented Extension to BPEL. *World Wide Web*, 10(3):309–344, 2007.

[27] A. Charfi, B. Schmeling, A. Heizenreder, and M. Mezini. Reliable, Secure, and Transacted Web Service Compositions with AO4BPEL. In *ECOWS '06: Proceedings of the European Conference on Web Services*, pages 23–34, Washington, DC, USA, 2006. IEEE Computer Society.

[28] 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.

[29] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, 20(6):476–493, 1994.

[30] E. J. Chikofsky and J. H. I. Cross. Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Software*, 7(1):13–17, 1990.

[31] 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.

[32] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley Professional, New York, NY, USA, Sept. 2002.

[33] P. Constantopoulos, M. Jarke, J. Mylopoulos, and Y. Vassiliou. The software information base: a server for reuse. *The VLDB Journal*, 4(1):1–43, 1995.

[34] R. Davis. *Business Process Modelling with ARIS: a Practical Guide*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[35] E. W. Dijkstra. EWD 447: On the role of scientific thought. *Selected Writings on Computing: A Personal Perspective*, pages 60–66, 1982.

[36] D. Drusinsky. *Modelling and verification using UML Statecharts*. Elservier, Apr. 2006.

[37] S. Dustdar and B. J. Krämer. Introduction to special issue on service oriented computing (SOC). *ACM Trans. Web*, 2(2):1–2, 2008.

[38] Eclipse. Eclipse Modeling Framework. `http://www.eclipse.org/emf`, 2004. (accessed 2007/02/01).

[39] Eclipse. Eclipse BPMN Modeler. `http://www.eclipse.org/bpmn`, 2006. (accessed 2008/02/01).

[40] A. Egyed. A Scenario-Driven Approach to Trace Dependency Analysis. *IEEE Trans. Softw. Eng.*, 29(2):116–132, 2003.

[41] T. Erl. *Service-Oriented Architecture: Concepts, Technology and Design*. Prentice Hall, 2005.

[42] EU Framework 7 STREP. Compliance-driven Models, Languages, and Architectures for Services (COMPAS). `http://www.compas-ict.eu`, 2008.

[43] M. Evenson and B. Schreder. SemBiz Deliverable: D4.1 Use Case Definition and Functional Requirements Analysis. `http://sembiz.org/attach/D4.1.pdf`, Aug. 2007.

[44] R. Farahbod, U. Glässer, and M. Vajihollahi. Specification and Validation of the Business Process Execution Language for Web Services. In *Abstract State Machines*, pages 78–94, 2004.

[45] N. Fenton and S. L. Pfleeger. *Software metrics (2nd ed.): a rigorous and practical approach*. PWS Publishing Co., Boston, MA, USA, 1997.

[46] D. F. Ferraiolo, J. F. Barkley, and D. R. Kuhn. A role-based access control model and reference implementation within a corporate intranet. *ACM Trans. Inf. Syst. Secur.*, 2(1):34–64, 1999.

[47] R. T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[48] R. E. Filman, T. Elrad, S. Clarke, and M. Aksit. *Aspect-Oriented Software Development*. Addison-Wesley Professional, Oct. 2004.

[49] FIT Project. Semantic Business Process Management for Flexible Dynamic Value Chains (SemBiz). `http://sembiz.org`, 2006.

[50] R. B. France, J. M. Bieman, and B. H. C. Cheng. Repository for model driven development (remoDD). In *MoDELS Workshops*, Lecture Notes in Computer Science, pages 311–317. Springer, 2006.

[51] D. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[52] I. Galvão and A. Goknil. Survey of Traceability Approaches in Model-Driven Engineering. In *EDOC*, pages 313–326, 2007.

[53] C. Ghezzi, M. Jazayeri, and D. Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, 1991.

[54] P. Goodman. *The Practical Implementation of Software Metrics*. McGraw-Hill, Inc., New York, NY, USA, 1993.

[55] O. Gotel and A. Finkelstein. Contribution structures [Requirements artifacts]. In *Proceedings of 1995 IEEE International Symposium on Requirements Engineering (RE'95)*, pages 100–107, 1995.

[56] F. Gottschalk, W. M. P. van der Aalst, M. H. Jansen-Vullers, and H. M. W. Verbeek. Protos2CPN: using colored Petri nets for configuring and testing business processes. *STTT*, 10(1):95–110, 2008.

[57] J. Greenfield, K. Short, S. Cook, and S. Kent. *Software Factories: Assembling Applications with Patterns, Frameworks, Models & Tools*. J. Wiley and Sons Ltd., 2004.

[58] J. H. Hayes, A. Dekhtyar, and J. Osborne. Improving requirements tracing via information retrieval. In *Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International*, pages 138–147, Sept. 2003.

[59] C. Hentrich and U. Zdun. Patterns for Process-Oriented Integration in Service-Oriented Architectures. In *Proc. of 11th European Conference on Pattern Languages of Programs (EuroPLoP 2006)*, pages 1–45, Irsee, Germany, July 2006.

[60] K. Hoffman and P. Eugster. Towards reusable components with aspects: an empirical study on modularity and obliviousness. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 91–100, New York, NY, USA, 2008. ACM.

[61] C. Hofmeister, R. Nord, and D. Soni. *Applied Software Architecture*. Addison-Wesley Professional, 1999.

[62] T. Holmes, H. Tran, U. Zdun, and S. Dustdar. Modeling Human Aspects of Business Processes - A View-Based, Model-Driven Approach. In I. Schieferdecker and A. Hartman, editors, *4th European Conference on Model Driven Architecture Foundations and Applications (ECMDA-FA) 2008*, volume 5095 of *LNCS*, pages 246–261. Springer, 2008.

[63] T. Holmes, U. Zdun, and S. Dustdar. MORSE: A Model-Aware Service Environment. In *Proceedings of the 4th IEEE Asia-Pacific Services Computing Conference (APSCC)*, page 0, Dec. 2009.

[64] IBM. Travel Booking Process. http://publib.boulder.ibm.com/bpcsamp/scenarios/travelBooking.html, 2006. (accessed 2008/01/05).

[65] IBM. WebSphere Process Server. http://www-01.ibm.com/software/integration/wps, 2008. (accessed 2008/02/01).

[66] IBM. WebSphere© MQ Workflow FlowMark© Definition Language (FDL). http://www-01.ibm.com/software/integration/wps, 2008. (accessed 2008/02/01).

[67] 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. (accessed 2007/02/01).

[68] IEEE. IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*, Dec. 1990. (accessed on 2007/06/08).

[69] IEEE. IEEE Std 1471-2000 IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. http://standards.ieee.org/reading/ieee/std_public/description/se/1471-2000_desc.html, 2007.

[70] IETF. RPC 1831: Remote Procedure Call Protocol Specification Version 2. http://tools.ietf.org/html/rfc1831, Aug. 1995. (accessed 2007/03/12).

[71] C. Inc. Subversion (SVN). http://subversion.tigris.org, Oct. 2000.

[72] G. Inc. Universal enterprise-wide deployment strategies may be counterproductive. http://www.gartner.com/it/page.jsp?id=717008, July 2008.

[73] Intalio. Intalio Server. http://www.intalio.com/products/server, 2008. (accessed 2008/02/01).

[74] ISO/IEC. ISO/IEC 10746-3 Open Distributed Processing – Reference Model: Architecture. http://standards.iso.org/ittf/PubliclyAvailableStandards/s020697_ISO_IEC_10746-3_1996(E).zip, Sept. 1996. (accessed 2008/02/22).

[75] JBoss. jBPM Process Definition Language (jPDL). http://jboss.org/jbossjbpm/jpdl. (accessed 2008/02/01).

[76] Jean-Jacques Dubray. WSPER - an abstract SOA framework. http://www.wsper.org, 2007. (accessed 2009/01/02).

[77] 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.

[78] G. Kiczales. Aspect-Oriented Programming. *ACM Comput. Surv.*, 28(4es):154, 1996.

[79] B. Kiepuszewski, A. H. M. ter Hofstede, and W. M. P. van der Aalst. Fundamentals of control flow in workflows. *Acta Informatica*, 39(3):143–209, 2003.

[80] E. Kindler. On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. In *Business Process Management*, pages 82–97, 2004.

[81] A. Kozlenkov and A. Zisman. Are their Design Specifications Consistent with our Requirements? In *RE '02: Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering*, pages 145–156, Washington, DC, USA, 2002. IEEE Computer Society.

[82] P. Kruchten. The 4+1 View Model of Architecture. *IEEE Softw.*, 12(6):42–50, 1995.

[83] P. Kruchten. *The Rational Unified Process: An Introduction (3rd Edition)*. Addison-Wesley Professional, Dec. 2003.

[84] I. Kurtev, J. Bézivin, F. Jouault, and P. Valduriez. Model-based DSL frameworks. In *OOPSLA '06: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 602–616, New York, NY, USA, 2006. ACM.

[85] C. F. J. Lange. Model size matters. In *Models in Software Engineering, Workshops and Symposia at MoDELS 2006, Genoa, Italy, October 1-6, 2006, Reports and Revised Selected Papers*, Lecture Notes in Computer Science, pages 211–216. Springer, 2006.

[86] P. Letelier. A Framework for Requirements Traceability in UML-based Projects. In *Proc. of 1st International Workshop on Traceability in Emerging Forms of Software Engineering - 17th IEEE International Conference on Automated Software Engineering*, pages 32–41, 2002.

[87] M. Lindvall and K. Sandahl. Practical implications of traceability. *Softw. Pract. Exper.*, 26(10):1161–1180, 1996.

[88] B. List and B. Korherr. A UML 2 Profile for Business Process Modelling. In *1st Int. Workshop on Best Practices of UML (BP-UML'05)*, Austria, 2005, 2005.

[89] A. D. Lucia, F. Fasano, R. Oliveto, and G. Tortora. Recovering traceability links in software artifact management systems using information retrieval methods. *ACM Trans. Softw. Eng. Methodol.*, 16(4):13, 2007.

[90] A. D. Lucia, R. Oliveto, and G. Tortora. Adams re-trace: traceability link recovery via latent semantic indexing. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 839–842, New York, NY, USA, 2008. ACM.

[91] S. Ma, L. Zhang, and J. He. Towards Formalization and Verification of Unified Business Process Model Based on Pi Calculus. In *SERA '08: Proceedings of the 2008 Sixth International*

Conference on Software Engineering Research, Management and Applications, pages 93–101, Washington, DC, USA, 2008. IEEE Computer Society.

[92] P. Mader, O. Gotel, and I. Philippow. Rule-Based Maintenance of Post-Requirements Traceability Relations. In *International Requirements Engineering, 2008. RE '08. 16th IEEE*, pages 23–32, Sept. 2008.

[93] P. Mäder, I. Philippow, and M. Riebisch. A Traceability Link Model for the Unified Process. In *SNPD (3)*, pages 700–705, 2007.

[94] J. I. Maletic, E. V. Munson, A. Marcus, and T. N. Nguyen. Using a Hypertext Model for Traceability Link Conformance Analysis. In *TEFSE'03: 2nd International Workshop on Traceability in Emerging Forms of Software Engineering*, 2003.

[95] A. Marcus and J. I. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 125–135, Washington, DC, USA, 2003. IEEE Computer Society.

[96] R. J. Mayer, C. P. Menzel, M. K. Painter, P. S. de Witte, T. Blinn, and B. Perakath. Integrated DEFinition for Process Description Capture Method Report. http://www.idef.com/pdf/Idef3_fn.pdf, Sept. 1995. (accessed 2008/02/01).

[97] C. Mayr, U. Zdun, and S. Dustdar. Model-Driven Integration and Management of Data Access Objects in Process-Driven SOAs. In *ServiceWave*, pages 62–73, 2008.

[98] T. J. McCabe. A complexity measure. *IEEE Trans. Software Eng.*, 2(4):308–320, 1976.

[99] S. J. Mellor, A. N. Clark, and T. Futagami. Guest editors' introduction: Model-driven development. *IEEE Software*, 20(5):14–18, 2003.

[100] J. Mendling and M. Hafner. From Inter-organizational Workflows to Process Execution: Generating BPEL from WS-CDL. In *OTM Workshops*, pages 506–515, 2005.

[101] 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.

[102] 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.

[103] 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.

[104] Microsoft. Windows Communication Foundation (WCF). http://msdn.microsoft.com/en-us/netframework/aa663324.aspx, 2008. (accessed 2008/03/06).

[105] E. E. Mills. Software metrics. http://www.sei.cmu.edu/reports/88cm012.pdf, Dec. 1998.

[106] R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes Pt.1. *Information and Computation*, 100:1–40, Sept. 1992.

[107] L. Naslavsky, H. Ziv, and D. J. Richardson. Towards traceability of model-based testing artifacts. In *A-MOST '07: 3rd International Workshop on Advances in Model-based Testing*, pages 105–114, New York, NY, USA, 2007. ACM.

[108] OASIS. Universal Description, Discovery and Integration(UDDI) 3.02. http://uddi.org/pubs/uddi-v3.0.2-20041019.pdf, Oct. 2004. (accessed 2006/04/20).

[109] OASIS. Business Process Execution Language (WSBPEL) 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf, May 2007.

[110] J. Oldevik and T. Neple. Traceability in Model to Text Transformations. In *2nd ECMDA Traceability Workshop (ECMDA-TW)*, pages 17–26, June 2006.

[111] H. Oliveira, L. Murta, and C. Werner. Odyssey-VCS: a flexible version control system for UML model elements. In *SCM '05: Proceedings of the 12th international workshop on Software configuration management*, pages 1–16, New York, NY, USA, 2005. ACM.

[112] OMG. UML 1.4 Specification - UML Profile for Business Modeling. http://www.omg.org/docs/formal/01-09-75.pdf, Sept. 2001. (accessed 2008/02/05).

[113] OMG. Model-Driven Architecture. http://www.omg.org/mda, 2003. (accessed 2006/03/02).

[114] OMG. Common Object Request Broker Architecture (CORBA). http://www.omg.org/docs/formal/04-03-12.pdf, Mar. 2004.

[115] OMG. Second revised submission to the MOF Model to Text Transformation RFP. 2005, Object Management Group. http://www.omg.org/cgi-bin/apps/doc?ad/05-11-03.pdf, 2005.

[116] OMG. Unified Modelling Language (UML) 2.0. http://www.omg.org/spec/UML/2.0, July 2005.

[117] OMG. Meta Object Facility (MOF ™) 2.0. http://www.omg.org/spec/MOF/2.0/HTML, Jan. 2006.

[118] OMG. Object Constraint Language(OCL) 2.0. http://www.omg.org/spec/OCL/2.0, May 2006.

[119] OMG. XML Metadata Interchange (XMI) 2.1.1. http://www.omg.org/technology/documents/formal/xmi.htm, Dec. 2007.

[120] OMG. Business Process Definition Metamodel 1.0. http://www.omg.org/spec/BPDM/1.0, Nov. 2008. (accessed 2009/01/03).

[121] OMG. Business Process Modeling Notation (BPMN) 1.1. http://www.omg.org/spec/BPMN/1.1, Jan. 2008.

[122] openArchitectureWare. A modular MDA/MDD generator framework. http://www.openarchitectureware.org, Aug. 2002. (accessed 2007/04/09).

[123] Oracle. Oracle Process Manager. http://www.oracle.com/technology/products/ias/bpel, 2004. (accessed 2009/02/01).

[124] Orchestra. Nova Orchestra. http://orchestra.ow2.org, 2008. (accessed 2009/02/01).

[125] 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 *IEEE International Conference on Web Services*, pages 285–292, 2006.

[126] M. Papazoglou. *Web Services:Principles and Technology*. Prentice Hall, 2007.

[127] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: a research roadmap. *Int. J. Cooperative Inf. Syst.*, 17(2):223–255, 2008.

[128] C. Pautasso, O. Zimmermann, and F. Leymann. Restful web services vs. "big" web services: making the right architectural decision. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 805–814, New York, NY, USA, 2008. ACM.

[129] K. Pohl. PRO-ART: Enabling Requirements Pre-Traceability. In *ICRE*, pages 76–85, 1996.

[130] F. Puhlmann. Soundness Verification of Business Processes Specified in the Pi-Calculus. In *OTM Conferences (1)*, pages 6–23, 2007.

[131] G. S. Raj, B. P. G., K. Babo, and R. Palkovic. Implementing Service-Oriented Architectures (SOA) with the Java EE 5 SDK. http://java.sun.com/developer/technicalArticles/WebServices/soa3/ImplementingSOA.pdf, May 2006. (accessed 2007/03/06).

[132] B. Ramesh and V. Dhar. Supporting Systems Development by Capturing Deliberations During Requirements Engineering. *IEEE Trans. Softw. Eng.*, 18(6):498–510, 1992.

[133] B. Ramesh and M. Jarke. Toward Reference Models for Requirements Traceability. *IEEE Trans. Softw. Eng.*, 27(1):58–93, 2001.

[134] 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, June 2006.

[135] M. Rekoff. On reverse engineering. *IEEE Transactions on Systems, Man and Cybernetics*, 15(2):244–252, 1985.

[136] Research and Markets. Services oriented architecture (SOA) infrastructure market shares, strategies, and forecasts, 2006 to 2012. http://www.researchandmarkets.com/reportinfo.asp?report_id=344145, July 2006.

[137] Research and Markets. Services oriented architecture (SOA) infrastructure market shares, strategies, and forecasts, 2006 to 2012. http://www.researchandmarkets.com/reportinfo.asp?report_id=471334, May 2007.

[138] Research and Markets. Services oriented architecture (SOA) infrastructure market shares, strategies, and forecasts, 2007 to 2013. http://www.researchandmarkets.com/reportinfo.asp?report_id=604023, Apr. 2008.

[139] Research and Markets. Worldwide SOA component services market shares, strategy, and forecasts, 2009 to 2015. http://www.researchandmarkets.com/reportinfo.asp?report_id=838370, Apr. 2009.

[140] K. Salimifard and M. Wright. Modelling and Performance Analysis of Workflow Management Systems Using Timed Hierarchical Coloured Petri Nets. In *ICEIS*, pages 843–846, 2002.

[141] C. Sant'Anna, A. Garcia, C. Chavez, C. Lucena, and A. v. von Staa. On the reuse and maintenance of aspect-oriented software: An assessment framework. In *Proceedings XVII Brazilian Symposium on Software Engineering*, 2003.

[142] A.-W. Scheer. *ARIS - Business Process Modeling*. Springer, 2000.

[143] B. A. Schmit and S. Dustdar. Model-driven Development of Web Service Transactions. *International Journal Enterprise Modelling and Information Systems Architectures*, 1(1):46–55, Oct. 2005.

[144] K. A. Schulz and M. E. Orlowska. Facilitating cross-organisational workflows with a workflow view approach. *Data Knowl. Eng.*, 51(1):109–147, 2004.

[145] M. Shepperd, editor. *Software engineering metrics I: measures and validations*. McGraw-Hill, Inc., New York, NY, USA, 1993.

[146] P. Sinogas, A. Vasconcelos, A. Caetano, J. Neves, R. Mendes, and J. Tribolet. Business processes extensions to UML profile for business modeling. In *In proceedings of the 3rd International Conference on Enterprise Information Systems (ICEIS 2001), Setubal*, pages 673–678, 2001.

[147] D. Skogan, R. Grønmo, and I. Solheim. Web service Composition in UML. In *Enterprise Distributed Object Computing Conference, 2004*, pages 47–57, 2004.

[148] G. Spanoudakis and A. Zisman. *Software Traceability: A Roadmap*, volume 3, pages 395–428. World Scientific Publishing, Handbook of Software Engineering and Knowledge Engineering: Recent Advances edition, 2005.

[149] G. Spanoudakis, A. Zisman, E. Prez-Miana, and P. Krause. Rule-based generation of requirements traceability relations. *Journal of Systems and Software*, 72(2):105–127, 2004.

[150] T. Stahl and M. Völter. *Model-Driven Software Development: Technology, Engineering, Management*. Wiley, 2006.

[151] Sun Microsystems. Java Platform Enterprise Edition. http://java.sun.com/javaee, 2008.

[152] P. Tarr, H. Ossher, W. Harrison, and S. M. Sutton,Jr. N degrees of separation: multi-dimensional separation of concerns. In *ICSE '99: Proceedings of the 21st International Conference on Software Engineering*, pages 107–119, New York, NY, USA, May 1999. ACM.

[153] T. C. Team. Concurrent versions system (CVS). http://www.nongnu.org/cvs, Nov. 1990.

[154] H. Tran, T. Holmes, U. Zdun, and S. Dustdar. *Modeling Process-Driven SOAs – a View-Based Approach*, chapter 2. Information Science Reference, Handbook of Research on Business Process Modeling edition, Apr. 2009.

[155] H. Tran, U. Zdun, and S. Dustdar. View-based and Model-driven Approach for Reducing the Development Complexity in Process-Driven SOA. In *Intl. Conf. on Business Process and Services Computing (BPSC)*, volume 116 of *LNI*, pages 105–124. GI, 2007.

[156] H. Tran, U. Zdun, and S. Dustdar. View-based Integration of Process-driven SOA Models At Various Abstraction Levels. In *R.-D. Kutsche and N. Milanovic, Editors, Proceedings of First International Workshop on Model-Based Software and Data Integration MBSDI 2008*, pages 55–66. Springer, Apr. 2008.

[157] H. Tran, U. Zdun, and S. Dustdar. View-Based Reverse Engineering Approach for Enhancing Model Interoperability and Reusability in Process-Driven SOAs. In H. Mei, editor, *10th Intl. Conf. on Software Reuse, ICSR 2008*, volume 5030 of *LNCS*, pages 233–244. Springer, 2008.

[158] H. Tran, U. Zdun, and S. Dustdar. VbTrace: Using View-based and Model-driven Development to Support Traceability in Process-driven SOAs. *Journal on Software & Systems Modeling – Special Issue on Traceability in Model-Driven Engineering*, 2009. (forthcomming).

[159] 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 Proc., 2006.

[160] 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.

[161] W. van der Aalst, M. Dumas, A. H. M. ter Hofstede, and P. Wohed. Pattern Based Analysis of BPMN (and WSCI). Technical report, FIT-TR-2002-04, Queensland University of Technology, Brisbane, 2002.

[162] W. 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.

[163] W. M. P. van der Aalst, M. Dumas, and A. H. M. ter Hofstede. Web Service Composition Languages: Old Wine in New Bottles? In *Proceedings 29th EUROMICRO Conference, Track on Software Process and Product Improvement*, pages 298–307, 2003.

[164] W. M. P. van der Aalst and A. H. M. ter Hofstede. YAWL: yet another workflow language. *Inf. Syst.*, 30(4):245–275, 2005.

[165] W. M. P. van der Aalst and K. van Hee. *Workflow Management. Models, Methods, and Systems.* MIT Press, Cambridge, MA, 2002.

[166] A. van Deursen, P. Klint, and J. Visser. Domain-Specific Languages: An Annotated Bibliography. http://homepages.cwi.nl/~arie/papers/dslbib, Mar. 1998. (accessed 2009/02/18).

[167] A. von Knethen, B. Paech, F. Kiedaisch, and F. Houdek. Systematic requirements recycling through abstraction and traceability. In *Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on*, pages 273–281, 2002.

[168] W3C. XML Path Language (XPath) 1.0. http://www.w3.org/TR/xpath, Nov. 1999. (accessed 2008/02/01).

[169] W3C. Simple Object Access Protocol (SOAP) 1.1 . http://www.w3.org/TR/2000/NOTE-SOAP-20000508, May 2000. (accessed 2006/02/20).

[170] W3C. Web Services Description Language 1.1, Mar. 2001.

[171] W3C. Web Services Architecture. http://www.w3.org/TR/ws-arch, Feb. 2004. (accessed 2008/07/12).

[172] W3C. Web Services Glossary. http://www.w3.org/TR/ws-gloss, Feb. 2004. (accessed 2009/02/20).

[173] W3C. Web Services Choreography Description Language (WS-CDL). http://www.w3.org/TR/ws-cdl-10, Nov. 2005. (accessed 2007/03/05).

[174] W3C. SOAP Version 1.2 Part 0: Primer (Second Edition). http://www.w3.org/TR/soap12-part0, Apr. 2007. (accessed 2008/06/11).

[175] W3C. Web Services Description Language 2.0. http://www.w3.org/TR/wsdl20, June 2007. (accessed 2008/09/11).

[176] W3C. Web Services Policy 1.5 - Framework. http://www.w3.org/TR/ws-policy, Sept. 2007. (accessed 2009/02/01).

[177] W3C. XML Schema Definition Language (XSD) 1.1 Part 1: Structures. http://www.w3.org/TR/xmlschema11-1, Apr. 2009.

[178] S. Walderhaug, E. Stav, U. Johansen, and G. K. Olsen. *Traceability Model-Driven Software Development*, pages 133–160. Designing Software-Intensive Systems - Methods and Principles. Information Science Reference, 2008.

[179] L. A. Walton. Software Design For Reliability and Reuse. http://www.spatial.maine.edu/~lisa.walton/dsl.html, 1996. (accessed 2007/02/18).

[180] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[181] WfMC. XML Process Definition Language (XPDL). http://www.wfmc.org/standards/XPDL.htm, Apr. 2005. (accessed 2008/02/01).

[182] S. A. White. Using BPMN to Model a BPEL Process. http://www.bpmn.org/Documents/Mapping%20BPMN%20to%20BPEL%20Example.pdf, Apr. 2005.

[183] D. S. Wile and J. C. Ramming. Guest Editorial: Introduction to the Special Section "Domain-Specfic Languages (DSL)". *IEEE Trans. Software Eng.*, 25(3):289–290, 1999.

[184] 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, June 2003.

[185] U. Zdun. Concepts for Model-Driven Design and Evolution of Domain-Specific Languages. In *International Workshop on Software Factories - OOPSLA 2005*. Software Factories, 2005.

[186] U. Zdun and S. Dustdar. Model-Driven Integration of Process-Driven SOA Models. *International Journal of Business Process Integration and Management (IJBPIM)*, 2(2):109–119, 2007.

[187] U. Zdun, H. Tran, T. Holmes, E. Oberortner, E. Mulo, and S. Dustdar. Compliance in Service-oriented Architectures: A Model-driven and View-based Approach. *Information Systems Journal*, 2009. (submitted).

[188] 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.

[189] A. Zisman and A. Kozlenkov. Managing Inconsistencies in UML Specifications. In *Proceedings of the ACIS Fourth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD'03), October 16-18, 2003, Lübeck, Germany*, pages 128–138. ACIS, 2003.

[190] A. Zisman, G. Spanoudakis, E. Pérez-Miñana, and P. Krause. Tracing Software Requirements Artifacts. In *Proceedings of the International Conference on Software Engineering Research and Practice, SERP '03, June 23 - 26, 2003, Las Vegas, Nevada, USA*, pages 448–455. CSREA Press, 2003.

[191] 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.

[192] M. zur Muehlen. *Workflow-based Process Controlling: Foundation, Design, and Application of Workflow-driven Process Information Systems.* Logos Verlag, Berlin, Germany, 2004.

[193] M. zur Muehlen and J. Recker. How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation. In *CAiSE '08: Proceedings of the 20th international conference on Advanced Information Systems Engineering*, pages 465–479, Berlin, Heidelberg, June 2008. Springer-Verlag.

Index