FAKULTÄT FÜR !NFORMATIK

Integrating Semantic Business Process Management and View Based Modeling

MAGISTERARBEIT

zur Erlangung des akademischen Grades

Magister der Sozial- und Wirtschaftswissenschaften (Mag. rer. soc. oec.)

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Clemens Blamauer, Bakk. rer. soc. oec.
Matrikelnummer 9835154

Mitverfasser:
Daniel Lintner, Bakk. rer. soc. oec.
Matrikelnummer 0103160

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer/Betreuerin: PD Dr. Uwe Zdun
Mitwirkung: Dipl.-Ing. Ta'id Holmes, DEA Bakk. techn.

Wien, _____ _____
(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

Clemens Blamauer
Ottakringerstraße 94/7
1170 Wien

Daniel Lintner
Westbahnstraße 5/2/24
1070 Wien

Hiermit erklären wir, dass wir diese Arbeit selbständig verfasst haben, dass wir die verwendeten Quellen und Hilfsmittel vollständig angegeben haben und dass wir die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht haben.

Wir haben uns bemüht, sämtliche Inhaber der Bildrechte ausfindig zu machen und ihre Zustimmung zur Verwendung der Bilder in dieser Arbeit eingeholt. Sollte dennoch eine Urheberrechtsverletzung bekannt werden, ersuchen wir um Meldung bei uns.

Wien, am _____ _____ _____
Clemens Blamauer Daniel Lintner

Acknowledgements

We would like to thank our advisors PD Dr. Uwe Zdun and Dipl. Ing. Ta'id Holmes, DEA from the Distributed System Group of the Institute of Information Systems, Vienna University of Technology for their help, their suggestions, and their support during the SemBiz project and during writing this thesis.

We additionally thank eTel Telekom Austria and Hanival Internet Services, where we commenced and carried out our practical work. Particularly, we want to thank Mark Evenson for his mentorship and his support as well as Bernhard Schreder for sharing his knowledge and profound feedback.

Finally we would like to thank all the SemBiz project's team members, it was a pleasure to work with you!

Clemens & Daniel

Especially, I want to thank my family: my father, my mother and my sister, whose love and support during my whole life made it possible to write this thesis in the first place.

Finally, I dedicate this work to Elisabeth, whose patient love and encouragement made it possible to complete this thesis.

Clemens

Mostly, I need to thank Karin and Yasmin for their absolute backing, love and staying the course, this work is dedicated to both of you.

I want to thank my family for making all this possible. Concretely I want to thank my Grandmother Margaretha and my Aunt Michaela, who convinced me to start my studies and always supported me during this time.

Daniel

Abstract

With the increasing interest in Service Oriented Architectures and related technologies, such as Web Services and the Business Process Execution Language (BPEL), Business Process Management (BPM) has become more and more important in recent years. However, there still exists a gap between Business Process Modeling, as it is done by business experts, and Business Process Deployment and Execution, as it is maintained by IT Experts. Currently, a lot of research is going on in the field of semantic technologies which promise to enable a high level of automation to narrow or even close this gap. This should be achieved through a well-defined knowledge representation which allows reasoning on the one hand and generation of executable code on the other hand.

In this thesis, we analyze the options to transfer ontologized knowledge representations into executable code and suggest a generic engineering process model using the facilities of Model Driven Software Development (MDSD) to fulfill this goal. This generic process is evaluated by introducing a concrete implementation done for the SemBiz project, where a semantic layer based on the Web Service Modeling Ontology (WSMO) is used for querying and reasoning over the process space, and a MDSD layer based on Eclipse Modeling Framework (EMF) is used for process abstraction and code generation.

Abstract

Mit dem zunehmenden Interesse an Service Orientierten Architekturen und den damit verbundenen Technologien, wie zum Beispiel Web Services und der Business Process Execution Language (BPEL), hat Geschäftsprozessmanagement in den letzten Jahren an Bedeutung gewonnen. Dennoch besteht eine Kluft zwischen der Gestaltung von Geschäftsprozessen, wie sie von ExpertInnen der Geschäftswelt durchgeführt wird, und deren Ausführung, die von IT-ExpertInnen umgesetzt und gewartet wird. Derzeit gibt es intensive Forschungsansätze im Bereich semantischer Technologien, die hohe Automatisierbarkeit versprechen, sodass diese Kluft weitgehend oder sogar vollständig überwunden werden soll. Erreicht werden soll dies durch eine wohldefinierte Wissensrepräsentation, die logisches Schließen erlaubt, auf der einen Seite sowie Generation von ausführbarem Code auf der anderen Seite.

In dieser Magisterarbeit analyisieren wir die Möglichkeiten, ontologisiertes Wissen in ausführbaren Code zu übersetzen und präsentieren einen allgemeinen Entwicklungsprozess, der die Möglichkeiten von Modellgetriebener Softwareentwicklung (Model Driven Software Development, MDSD) nützt, um dieses Ziel zu erreichen. Dieser allgemeine Prozess wird dann an Hand einer konkreten Implementierung im Rahmen des SemBiz Projekts evaluiert. Im Rahmen des Projektes gibt es eine Semantische Schicht, basierend auf der Web Service Modeling Ontology (WSMO), die Abfrage und Schließen über die Prozesse der Wissensbasis ermöglicht, und eine Modellgetriebene Schicht basierend auf dem Eclipse Modeling Framework (EMF), die für Prozessabstraktion und Codegenerierung verwendet wird.

Contents

1 Introduction (by Clemens Blamauer and Daniel Lintner) 1
- 1.1 Problem definition . . . 2
- 1.2 Structure of this work . . . 3

2 The Workflow Technological Space (by Daniel Lintner) 4
- 2.1 Business Process Modeling . . . 4
- 2.2 SOA and Web Services . . . 7

3 The Ontology Technological Space (by Daniel Lintner) 8
- 3.1 Ontologies in the field of Computer Science . . . 9
- 3.2 Ontologies in the field of Semantic Web . . . 11
- 3.3 Ontologies in the field of Semantic Web Services . . . 12

4 The Model Driven Engineering (MDE) Technological Space (by Clemens Blamauer) 20
- 4.1 Model Driven Architecture (MDA) . . . 22
- 4.2 Eclipse Modeling Framework (EMF) . . . 25
- 4.3 Viewbased Modeling Framework (VbMF) . . . 27
  - 4.3.1 Core View and extension mechanisms . . . 28
  - 4.3.2 VbMF ControlFlow View . . . 29
  - 4.3.3 VbMF Collaboration View . . . 30
  - 4.3.4 VbMF Information View . . . 31

5 View-based Ontology Integration Process (by Clemens Blamauer and Daniel Lintner) 33

6 View Creation (by Clemens Blamauer) 34
- 6.1 Related Work . . . 35
  - 6.1.1 Ontology Definition Metamodel (ODM) . . . 35
  - 6.1.2 EMF-based Ontology Definition Metamodel . . . 36
  - 6.1.3 ModelCVS . . . 37
  - 6.1.4 Differences between Ontologies and MDE . . . 38
- 6.2 Create Ontology Metamodel . . . 40
- 6.3 Create Ontology Import . . . 43
- 6.4 Create View Metamodel . . . 43
- 6.5 View Import . . . 51

6.6 Manual View and Import Completion 53

7 Create Transformation Rules (by Daniel Lintner) 57
7.1 Related Work 58
7.1.1 Modelware 58
7.1.2 Model Transformations By-Example 58
7.2 Create Examples 60
7.3 Deriving Model Correspondences 61
7.3.1 Classification of source elements 62
7.3.2 Identification of element correspondences 64
7.3.3 Conclusion 67
7.4 Design Transformation Rules 68
7.4.1 Incomplete 70
7.4.2 Parameters 72
7.4.3 Precomputed 73
7.4.4 Model Navigator 74
7.4.5 Path Reminder 75
7.4.6 Metamodel Polymorphic Rules 75
7.4.7 Test and Evaluation 76

8 Integration and Deployment (by Clemens Blamauer) 78
8.1 Integration of single Model Transformation Steps 78
8.1.1 Transformation Workflow 79
8.1.2 Workflow Component Adapters 80
8.2 Transformation Deployment 81
8.2.1 Web Service Deployment 81
8.2.2 Sembiz Process Deployment process 81

9 Evaluation (by Clemens Blamauer and Daniel Lintner) 84
9.1 Evaluation of the practical transformation task 84
9.1.1 Mapping the Process Setup 84
9.1.2 Mapping the Process Start and End 86
9.1.3 Mapping Atomic Processes 89
9.1.4 Mapping Atomic Process Invariants 93
9.1.5 Mapping Composite Processes 93
9.1.6 Mapping Conditional Branches 97
9.1.7 Mapping Fault- and Compensationhandling 100
9.2 Open Issues and future Work 101

10 Conclusion
(by Clemens Blamauer and Daniel Linter) **105**

A Zusammenfassung auf Deutsch **106**

B Lebenslauf Clemens Blamauer **107**

C Lebenslauf Daniel Lintner **107**

List of Figures

1 The Critical IT - Process Divide . . . 2
2 Simple EPC diagram . . . 5
3 Simple BPMN diagram . . . 6
4 Selected classes and properties of the OWL-S Profile . . . 14
5 Top level OWL-S process ontology . . . 15
6 WSML variant space . . . 17
7 The MOF Metadata Architecture . . . 23
8 CIM, PIM, and PSM . . . 24
9 Ecore key types . . . 26
10 Meta levels of the View-based Modeling Framework . . . 28
11 VbMF Core View . . . 28
12 VbMF ControlFlow View . . . 30
13 VbMF Collaboration View . . . 31
14 VbMF Information View . . . 31
15 High-level View-based Integration Process . . . 33
16 Create View subprocess . . . 34
17 Common features of UML and OWL . . . 36
18 Transformation between Ecore and EODM OWL . . . 37
19 ModelCVS Ecore to OWL mapping . . . 38
20 The WSMO Metamodel in MOF . . . 41
21 Import of XML Schema into an EMF Project . . . 41
22 The WSMO Metamodel in Ecore . . . 42
23 Transformation setup of the Create View Metamodel activity . . . 43
24 The BPMO Ontology as an EPackage . . . 45
25 Mapping concepts to EClasses . . . 47
26 Mapping relations to EClasses and EReferences . . . 48
27 Mapping enumeration-like axioms to EEnums . . . 51
28 Transformation setup of the Create View Import activity . . . 53
29 Transformation setup of the View Import activity . . . 53
30 Create Transformation subprocess . . . 57
31 Overview of the Modelware work packages . . . 59
32 Overview of the MTBE framework . . . 59
33 The OrderProvisioning process ontology . . . 60
34 The OrderProvisioning BPEL Transaction View . . . 61
35 Example for irrelevant elements in the BPMOView . . . 62
36 Example for hidden elements in the VbMF CollaborationView . . . 63
37 Example of missing information for the VbMF InformationView . . . 63
38 1:1 mappings between BPMO views and the VbMF Collaboration view . . . 64

39 Example of a source class having multiple 1:1 mappings 65
40 Example of a 1:N mapping between BusinessProcess and Activity 66
41 Example of a N:1 correspondance 67
42 Example of model correspondences on the class level 68
43 Example model correspondences on the class and attribute level . 69
44 BPMO View to BPEL Transaction View transformation setup . . . 69
45 Simple Guidance Metamodel 71
46 Guidance Example 71
47 Parameter Metamodel 72
48 Parameter Example 73
49 OrderProvisioning BPEL validation 77
50 Deploy subprocess 78
51 OrderProvisioning - BPEL Collaboration View 85
52 OrderProvisioning - BPEL Transaction View 87
53 Process Start in the BPEL Collaboration View 88
54 Process Start in the BPEL Information View 88
55 PleskUserLookup - BPMO Business Process View 91
56 PleskUserLookup - BPEL Collaboration View 92
57 PleskUserLookup - BPEL Information View 93
58 Mapping XOR and CASE SelectProcesses 97
59 Mapping OR SelectProcesses 98
60 Generated OrderProvisioning BPEL 102

1 Introduction

In the last two decades, more and more companies began to re-define themselves through their business processes. The initial idea behind was to become aware of processes which generate value for either customers or the companies themselves, to document the outcome and be able to re-engineer business processes according to changes in market demand [32].

The forming of the service oriented architecture (SOA) pattern, and the development of web technologies and standards lead to a couple of enabling technologies which in combination are able to integrate business processes and the IT infrastructure properly.

However business processes, as defined by business experts and executable processes designed by IT personnel are hardly linked and additionally do operate on different levels of detail. As shown in Figure 1, the so called *Critical IT - Process Divide* leaves a gap between business experts (real persons or teams), which are not necessarily familiar with technical details, and developers, which may in turn not be familiar or in touch with the development of high level business processes.

Business process management (BPM) aims at reducing this gap, in managing business processes from a business view:

> ”BPM technology provides not only the tools and infrastructure to define, simulate, and analyze business process models, but also the tools to implement business processes in such a way that the execution of the resulting software artifacts can be managed from a business process perspective.” [41]

Further, the creation and maintenance of process oriented IT infrastructures is a non-trivial and labor intensive task.

However, there is still a gap between the business and the technical view: business experts (real persons or teams) are not necessarily familiar with technical details. On the other hand, business process models somehow have to be transformed to executables, deployed to the infrastructure and maintained by technical people.

One example and the scope of this work - is the SemBiz project, which claims to develop a prototype (toolchain) that supports the whole business process management lifecycle through

> ”[...] an exhaustive semantic description framework that allows managing business processes on the business level as well as support for automated execution of business processes on the technical level.” [15]

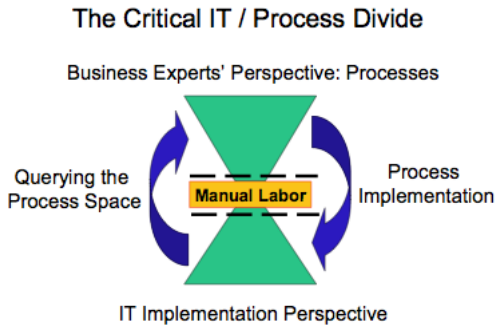


Figure 1: The Critical IT - Process Divide
(source: [35])

It uses semantics for discovery, querying and functional composition, and a Model-driven approach for syntactical composition and domain specific languages (DSL) for generation of executables [59].

In order to develop and test the SemBiz toolchain, real world use cases are provided by use case partners, namley eTel Telekom Austria and Hanival Internet Services. Both companies are working in the field of internet service providing (ISP)/telecommunication. As scientific partners in the project, the Digital Enterprise Research Institute (DERI) Innsbruck contribute the functional composition parts like semantic annotation, modeling, process reasoning and discovery, the Information Systems Institute (Infosys) Vienna is concerned with the syntactical composition which includes model-driven composition and validation and provides code generation capabilities in order to generate executable processes (BPEL).

1.1 Problem definition

In order to generate non-semantic, executable code from ontologies, it is necessary to translate between different technological spaces. A part of the problem is mitigated by the VbMF framework [36, 59–61], which reduces complexity through a Separation of Concerns approach, in providing different architectural views [38]. In order to make use of the VbMF framework, which is built on MDSD (model-driven software development) technologies, it is necessary to populate VbMF views out of business process ontologies, which is the main focus of our work.

1.2 Structure of this work

The further work is structured as follows: In Section 2, will give a brief outline of both BPM and SOA. We will then give a general overview of ontologies in Section 3, and discuss, how they can improve BPM and SOA through the use of semantic technologies. In Section 4, we will then introduce the theoretical concepts of Model-Driven Engineering and technologies to realizing them.

In Section 5 we will present an integration process in order to bridge between ontologies and model-driven technologies. The first integration step, described in Section 6 is to automatically create views on ontologies in terms of models and facilities to populate these views from ontologies. In Section 7 we describe the next integration step, translating between the ontology views and the VbMF views. Finally, in Section¡8, we will show how single transformation steps are combined to provide a Web service that is orchestrated by the Sembiz Deployment process.

In the evaluation in Section 9 we will analyze the Use Case Partner's use cases to evaluate the transformation and outline unaddressed issues and future work.

2 The Workflow Technological Space

The foundation of modern workflow technology dates back on work in the late 1970's and the early 1980's. The computer science field of *office automation* was researched by industry and universities including AT&T, IBM, Xerox, M.I.T or the Harvard Business School [29]. Later movements like Business Process Reengineering made the idea of a process-driven organization more and more common. Today almost every company is aware of its processes and documented them in some kind of way but few of them have them actually in an executable form, so there exists a huge amount of business knowledge in the form of diagrams but no corresponding executable code.

The following section gives a brief introduction into process modeling notations. Section 2.2 discusses Service Oriented Architectures (SOA) and Web Services as todays enabling technologies.

2.1 Business Process Modeling

In order to describe business processes adequately a number of Business Process Modeling Languages (BPMLs) have been developed and used in a variety of domains and application areas. Their aim is to describe process models including information about what a process is going to solve, how a process is going to solve its task, what kind of business data or objects are going to be processed, who is participating in the process and what kind of responsibilities are associated with the process. Widely-used BPMLs are Event-driven Process Chains (EPCs) and the Business Process Modelling Notation (BPMN) which are briefly introduced in the following paragraphs.

Event-driven Process Chains EPCs provide a simple and easy understandable notation. The goal of EPCs is to support the modeling of correlations between businesses and information systems. The notation is based on the elements *Event* and *Function* as well as *connectors* between them.

Events are said to be passive elements which cause a function or are produced by a function. An event is represented as a rectangle. Functions are active elements and model concrete tasks or activities carried out in a company. Functions are transformations between an initial and a resulting state. Furthermore, a complex function again can be modeled in a separate EPC and is then called a hierarchical function. A Function is represented as a rounded rectangle.

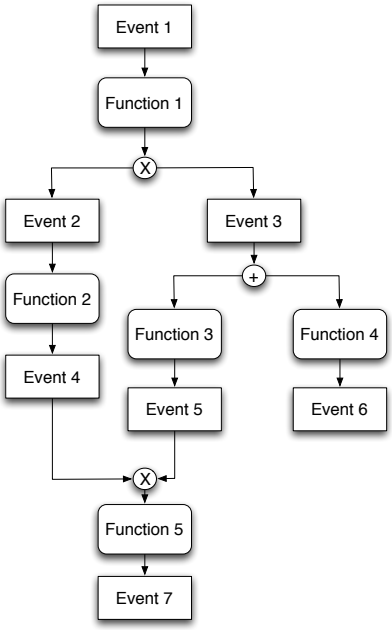


Figure 2: Simple EPC diagram (according to [40])

An EPC denotes an ordered graph that makes use of connectors and logical operators such as OR, AND and XOR in order to steer the activation of the path of the control flow in a process. A Connector is represented as a circle. Figure 2 gives a simple example of a process modeled with EPC.

Generally one can distinguish between opening and closing connectors. Opening connectors can have one incoming control flow and two or more outgoing control flows. Closing connectors can have two or more incoming control flows and exactly one outgoing control flow. An OR connector is used to activate one or more paths in the control flow, other paths that do not meet a certain condition are deactivated. An OR connector is represented as an empty circle. Differently, an XOR connector activates exactly one outgoing control flow based on the given condition. All other alternative flows are deactivated. An XOR connector is represented as a circle labeled with an 'X'. Finally an AND connector is used to model a parallel flow, stating that all follow-up paths are activated if a certain condition is true. An AND connector is represented as a circle labeled with a '+'.

Business Process Modeling Notation BPMN has been developed by the Business Process Management Initiative (BPMI) with the primary goal to provide a readily understandable graphical notation for all business users. It aims to provide

a standardized bridge between the business process design done by business analysts and experts and the implementation of the business processes done by technical developers. Additionally BPMN was designed to map with process execution languages such as the Business Process Execution Language for Web Services (WS-BPEL).

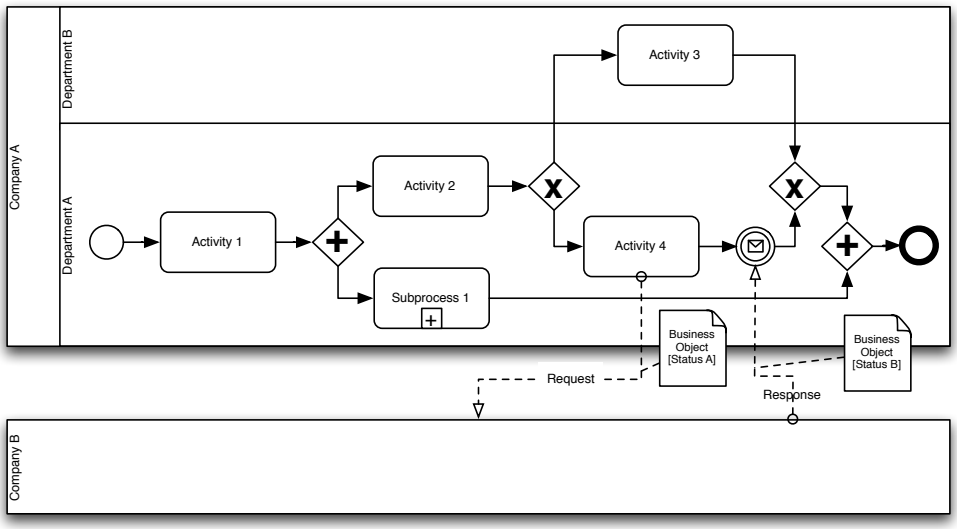


Figure 3: Simple BPMN diagram

The BPMNs core element set can be divided into four basic categories:

- **Flow Objects** define the behavior of a Business Process. BPMN distinguishes three types of Flow Objects: 1) Events, 2) Activities and 3) Gateways. Events arise in the course of the process flow which have a cause or an impact influencing the flow of the process. Based on when an event affects the process flow one can distinguish between events that occur at the beginning (Start), in the middle (Intermediate) or at the end (End) of the process. BPMN knows various types of events like Message, Timer, Error or Compensation. Their graphical notation is a circle with open centers or internal markers representing the type of the event. Figure 3 makes use of three events. A Start Event at the beginning, an Intermediate Event waiting for a message to arrive and an End Event stating the end of the process. Activities are representing units of work that are performed by a companies organizational unit. Activities may be atomic (Task) or non-atomic (Sub-Process) and are illustrated as rounded rectangles. An activity can be

a process too, but is then contained in a separate pool. Figure 3 shows Tasks (e.g. Activity 1), a collapsed Sub-Process (Subprocess 1) and an external Process (Company B). For modeling a divergence or a convergence of the control flow Gateways are used. They describe branches and merges, forks and joins and are represented as diamonds with internal markers. There exist Data-Based or Event-Based Exclusive Gateways, Inclusive Gateways, Complex and Parallel Gateways. Figure 3 contains a Parallel Gateway forking after Activity 1 and joining before the End Event and a Data-Based Gateway branching after Activity 2 and merging before joining with the Parallel Gateway.

- **Connecting Objects** are used to connect Flow Objects and other information. The three Connecting Objects available are 1) Sequence Flow, 2) Message Flow and 3) Association. A Sequence Flow is used to show the order of activities performed in a Process. A Sequence Flow is a solid line with a filled arrowhead. A Message Flow shows the flow of a message that is sent or received by different activities. A Message Flow is a dashed line with an open arrowhead. Associations are used to association various information with Flow Objects. An Association is a dotted line with a normal arrowhead.

- **Swimmlanes** are used to group the primary modeling elements. There exist two types of Swimmlanes: 1) Pools and 2) Lanes. Pools represent the participants in a process e.g. Company A and Company B in Figure 3. Lanes are used to sub-divide Pools in order to organize and categorize activities, Department A and B of Figure 3 are examples of Lanes.

- **Artifacts** provide a way to add additional information about a process which do not directly influence the flow of a process. Currently there are three standardized Artifacts, but BPMN allows the definition of any other required Artifact. Current Artifacts include: 1) Data Object, 2) Group and 3) Annotation. Data Objects provide information about what activities require as an input respectively what output they produce. Groups are used to visualize categories of activities for documentation and analyzing purposes. Annotations, typically in form of natural language text, are a mechanism to provide additional information for the reader of a diagram and are connected to specific objects within the diagram.

2.2 SOA and Web Services

Service Oriented Architecture (SOA) is an architectural design approach for software and IT infrastructure. It is based on loosely coupled Web services that pro-

vide dynamic binding, platform independence and high interoperability. SOA is an abstract concept, not an implementation.

Web services and -standards One implementation of a Service-oriented architecture (SOA) are Web services. Web services are software components made available through the internet which can be published, discovered and invoked. They make use of XML-based standards in order to interchange data. Concretely a Web service provides an interface for software operations over a network in order to perform a message-based data transfer. Web services are said to be self-contained, meaning their independency of other components or services [27].

> "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." [18]

A Web service is an abstract set of functionality characterizing a concrete piece of software or hardware that sends and receives messages in order to implement a service whereas one service may be implemented by different agents (e.g. programming languages). Such an abstract service description is called a Web service description (WSD) which documents the mechanics of the message exchange in a machine-processable format and includes among other things information about message formats, datatypes, transport protocols, message serialization formats or the network location where the service can be invoked.

Orchestration Web services independently provide atomic functionalities. In order to use them in business processes, they need to be composed to higher level processes. One standardized approach to do this is the Web Services Business Process Execution Language (WS-BPEL), an XML based language standardized by OASES (cp. [39]).

3 The Ontology Technological Space

The following section discusses the term ontology in the context of information and computer science and their fields of use. Further a closer look at ontology languages and their capabilities in the area of the Semantic Web as well as in the area of Web services is taken. Further a comprehensive example provides a deeper understanding of how ontologies are built.

The origin of the term ontology can be found in the field of philosophy and can be considered as a part of the major philosophical branch of metaphysics [13]. Their major task is to draw out a view of what things and what kind of things exist and attempts to address questions such as: *'What is being?'* or *'What characteristics do all beings have in common?'* [56]. It is about creating a system of categories and providing criteria in order to distinguish various types of objects (concrete and abstract, existent and non-existent) and their ties (relations, dependences and prediction) [9] .

In the field of information and computer science, ontologies have a similar purpose as their philosophical origin. They are briefly discussed in the next section.

3.1 Ontologies in the field of Computer Science

Ontologies strive at classifying and relating things in order to clarify the picture of what these things are and of what purpose they might be, aiming to make this knowledge processable by machines. It can be considered as a design artifact defining a specific vocabulary to describe entities in some domain of interest together with a set of assumptions about the vocabularies intended meaning [56].

A frequently cited definition is given by Gruber [31]:

> "An *ontology* is an explicit specification of a conceptualization."

and further:

> "A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose. Every knowledge base, [...] is committed to some conceptualization, explicitly or implicitly."

So an ontology explicitly specifies a *conceptualization*, which means the labling of objects, concepts and other entities along with their interrelation in some area of interest which are used to reach a formal *commitment* of meaning.

The foundations of ontology languages date back to work in the fields of knowledge representation and reasoning, database management, logic programming and object oriented programming. Modeling wise, ontologies have many similarities with other common approaches to cover domain knowledge like the Entity Relationship Model (ERM) or the Unified Modeling Language (UML) but exceed those in different dimensions ([24] , p. 19 - 20).

- The description format consists on semi-stuctured natural language text.

- It's primary goal is to reach an agreement on the meaning of terms and vocabulary across cooperating communities or individual applications.
- The description language of ontologies is typically syntactically and semantically richer, they are formalized in logic based representation languages an therefore describe domains in an unambiguous way.
- An ontology provides a formal definition of a domain, not the structure of a data container

An ontology can be specified in a variety of forms mainly differing in the degree of formality by which a vocabulary and its meaning is specified. This can even be in form of natural language which is typically not the case in the field of computer science. Uschold et al. [62] distinguish:

- highly informal: expressed in natural language
- semi-informal: expressed in a restricted and structured form of natural language
- semi-formal: expressed in a artificial formally defined language
- rigorously formal: expressed in meticulously defined terms with formal semantics, theorems and proofs

Nowadays ontologies find their use in many domains and their function spreads over wide areas of use. As a discussion of all the specific appliances of ontologies would go beyond the scope of this work , here just an abstract overview is given. We adhere to [62] where three general spaces of use of ontologies are identified: *Communication*, *Interoperation* and *Systems engineering*.

Communication As Ontologies reduce conceptual and terminological confusion they enable a shared understanding among people having different needs and viewpoints arising from their specific context. There may be different aspects where ontologies are able to facilitate the communication.

Interoperation Enterprise information systems typically combine many subsystems and tools and operate on various data sources and formats requiring the integration of and interoperation between these resources. Ontologies can assist such inter-operability by supporting translations between different representations either by directly translating between two specific representations or through acting as an inter-lingua, a pivot between multiple representations.

Systems engineering Different to the above mentioned operational role of ontologies in software systems, they are applicable to system design and development as well for instance when identifying the requirements and clear out relationships of various system components or doing automatic constituency checks.

Concluding we can say that, generally spoken, ontologies are applied in the fields of communication, interoperation and system engineering. Their task is to describe a universe of discourse unambiguously, bridge between different domains as well as enabling reasoning applications to reason over the logical consequences of their statements.

3.2 Ontologies in the field of Semantic Web

When Tim Berners-Lee et al. [20] dealt with the term Semantic Web in early 2001 they envisioned the future of the Web to be more than just a collection of web pages dedicated to human consumers. Instead the enrichment of the available information with machine readable semantics should open the way for performing complex reasoning tasks and understanding meaning by computer system autonomously and help to overcome the informations overloads of today and the future.

The major building blocks initially enabling the Semantic Web were technologies like the Extensible Markup Language (XML) [3] together with their scheme languages XML Schema and Document Type Definition (DTD), the Resource Description Framework (RDF) [10] and the Web Ontology Language (OWL) [11]. XML enables Web publishers and programmers to describe their own tags and annotate their resources in a standardized way. But still the interpretation of the meaning behind is up to humans. Based on XML, RDF provides the annotation of resources on the World Wide Web with various metadata which is cleary identified through the use of Unified Resource Identifiers (URIs) geared to enable interpretation done by machines. But even though metadata is exactly identified through their URIs in RDF there is no way of relating such terms. This directly leads to the use of ontologies.

A dedicated ontology language for the web is OWL. Different to other languages for the Web, which were designed to present information to human readers like the Hypertext Markup Language (HTML), the OWL's goal is to enable the Web for machines allowing automated processing and integration of Web data. While mainly extending the RDF standard, OWL adds additional vocabulary for describing properties and classes as it introduces the ability to describe relations

between classes, cardinality, equality, richer typing of properties, characteristics of properties, and enumerated classes ([24], p. 23).

Based on the required expressiveness one may choose between the three available sublanguages of the OWL, where each of them is an extension oft its predecessor:

- *OWL Lite* is the basic version providing a classification hierarchy and simple constraints (values of 0 or 1). It has a reduced expressiveness but in turn is of less complexity.
- *OWL DL* provides a maximum of expressiveness while still retaining computational completeness and decidability. It includes all available OWL language constructs, which may be used under certain constraints.
- *OWL Full* provides the same features as OWL DL, but does not pose the same constraints on their usage. Therefor, no computational guarantees are given when applying the full expressiveness of OWL Full.

A good introduction to ontological knowledge representations can be found in [12] which is the wine ontology example by the W3C.

3.3 Ontologies in the field of Semantic Web Services

Parallel to the Semantic Web initiatives, the realization of Service Oriented Architectures gained momentum through enabling technologies for Web services and a meaningful set of Web standards. These promising developments constituted a step ahead towards the seamless integration of distributed and heterogenous software components. Nevertheless available Web service technologies suffer from the same problems the common web does: their operation on a syntactic level.

The major drawback with this syntactic level is the currently huge amount of labor required in order to setup, maintain and change service landscapes. Machine processable semantics promises to eliminate (some of) theses drawbacks by introducing automation at every level of the process life cycle including service discovery, service execution and service composition and interoperation (cp. McIlraith et. al.).

Automatic Web service discovery using semantic service descriptions together with ontology-enhanced search engines allow the automatic location of services on the base of complex request properties. One could consider the task of searching for some particular travel service which is nowadays still a time consuming task done by humans, e.g.: "Find a service that sells train tickets for the route Vienna - Berlin and offers payment via Visa card.".

Automatic Web service execution Assuming automatic service discovery in place, autonomous software agents could search the web for appropriate services and execute them on the behalf of humans. A user could formulate an order like "Buy me the cheapest available train ticket from Vienna to Berlin and pay it with my Visa card". In todays web, this would involve a couple of human tasks including browsing the vendors site, login (or even doing registration first), filling out forms and all the rest of it.

Automatic Web service composition and interoperation comes into play, when a complex , high-level order description turns out to require a sequence of various services. Specialized software would be able to reason and interpret the provided service descriptions, selecting, composing and mediating between services automatically. Today this is a cumbersome task requiring manual composition and the mediation between service request formats generally to difficult for the average user and a dedicated job for IT-personnel.

OWL-S One prominent language aiming on this requirements is OWL-S. As its name suggests it is based on OWL (see Section 3.2). OWL-S provides an upper ontology for services describing the essential types of knowledge about a service, namely the `ServiceProfile`, `ServiceModel` and `ServiceGrounding`.

- The `ServiceProfile` basically answers the question of what the service can do for its clients. A service is said to present a `ServiceProfile`. This information can be used by software agents in order to determine if a service meets the required needs. It contains information like a description in natural language or contact information, but mainly functional descriptions as shown by Figure 4.

- The `ServiceModel` contains information of how a service fulfills its task and describes what happens when a service is executed. A service is said to be described by a `ServiceModel`. This information can be used to a) perform a more in-depth analysis of whether meets the required needs, b) compose several services fulfilling a more complex need, c) coordinate the activities of different participants during service enactment, and d) monitor the

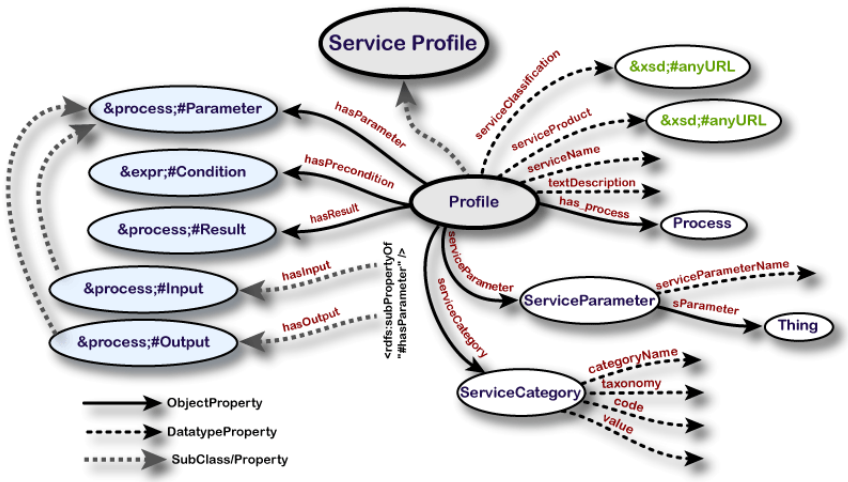


Figure 4: Selected classes and properties of the OWL-S Profile (source: [16])

service's execution. A subclass of the `ServiceModel` is the `Process` class introduced in OWL-S 1.1 providing a rich foundation for describing a service's inner structure as shown by Figure 5.

- The `ServiceGrounding` answers the question of how a service is used. A service is said to support a `ServiceGrounding`. The information in the grounding is used to provide information for the actual interaction with the service, including the communication protocol, message formats and other details such as the service's address and port. Additionally for each abstract type in the `ServiceModel` a `ServiceGrouding` provides an unambiguous way of exchanging data elements of that type with the service.

In order to annotate a published Web service, each distinct Web service is represented by instances of the `Service` class having the properties *presents*, *describedBy* and *supports* which in turn point to instances for its `ServiceProfile`, `ServiceModel` and `ServiceGrounding`. Each of these basic class provides an essential type of information about the service which can be further specialized by subclassing them. The upper ontology further specifies two cardinality constraints: 1) a service can be described by at most one service model, and 2) a grounding must be associated with exactly one service. For the properties *presents* and *describedBy* not further restrictions on the cardinality are given. Nevertheless OWL-S provides a default approach with its upper ontologies for profiles, models and groundings and allows the construction of alternative approaches too.

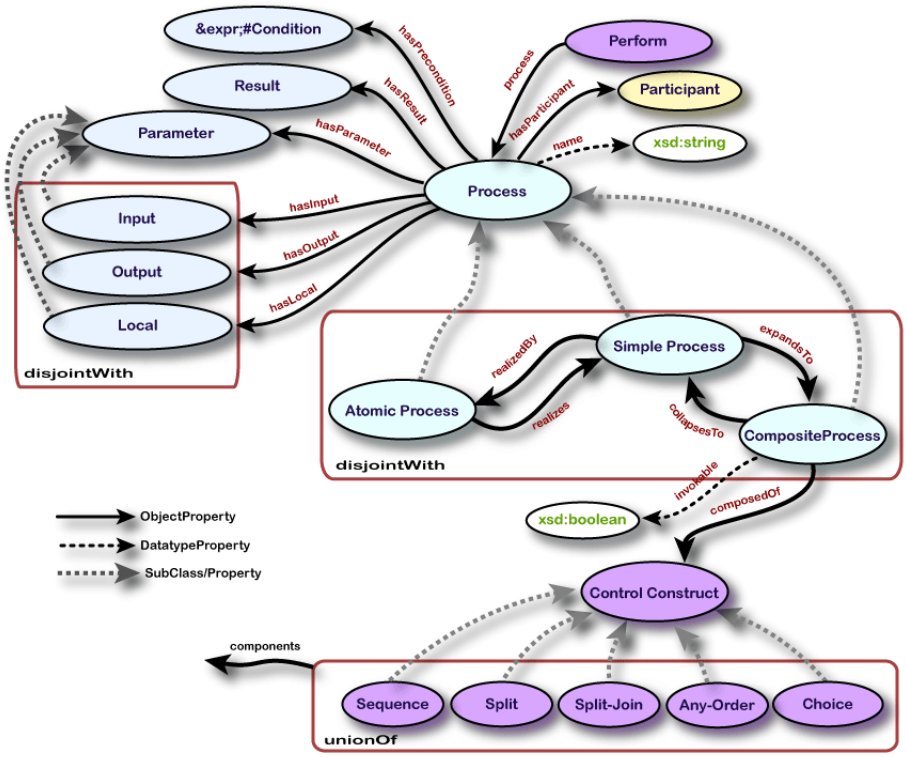


Figure 5: Top level OWL-S process ontology (source: [16])

There exist a loose collection of tools to support the creation and processing OWL-S each of them focusing on different specific aspects [58]. The set of tools are OWL-S Editor, OWL-S Matchmaker, OWL-S Virtual Machine (OWL-S VM), WSDL2OWL-S converter and OWL-S2UDDI converter.

- The OWL-S Editor can be used to develop semantic descriptions for Web services. It is easy to use and hides the complex constructs of the markup language.
- The OWL-S Matchmaker outputs different degrees of matching for individual elements of OWL-S descriptions and service profiles. It allows the ranking based on a criterion in order to select a service out of a large number of results, thus provides decision support to autonomously choose the most suitable service around.
- The OWL-S VM provides a client capable to invoke Web services based on their OWL-S process model. Its base functionality is to control the interaction between services based on the information described in OWL-S.
- WSDL2OWL-S reduces human labor as it provides the transformation of a WSDL WSD into OWL-S. This results in a specification of the ServiceGrounding and a partial specification of the ServiceModel and the ServiceProfile. Nevertheless the OWL-S description of a service is much richer than the description provided by WSDL and therefor has to be completed manually. Anyways, WSDL2OWL-S generates a basic structure of the OWL-S description and reduces work to a great extend.
- OWL-S2UDDI goes the other direction then WSDL2OWL-S and converts OWL-S profile descriptions into UDDI descriptions that can be directly published in a UDDI registry.
- An effort to integrate these and other loose tools into an cohesive framework is made with the OWL-S Integrated Development Environment (IDE).

WSMO and WSML The Web Services Modeling Language (WSML) is an ontology language dedicated to the domain of Web services. It is a formal language that provides the syntax and semantics for the Web Services Modeling Ontology (WSMO) which together build a unified language framework to semantically describe different aspects of Semantic Web Services (SWS), where WSMO serves as the conceptual model.

Differing in their logical expressiveness and the underlying language paradigms WSML comes with a set of language variants as shown in Figure 6 and outlined in [24].

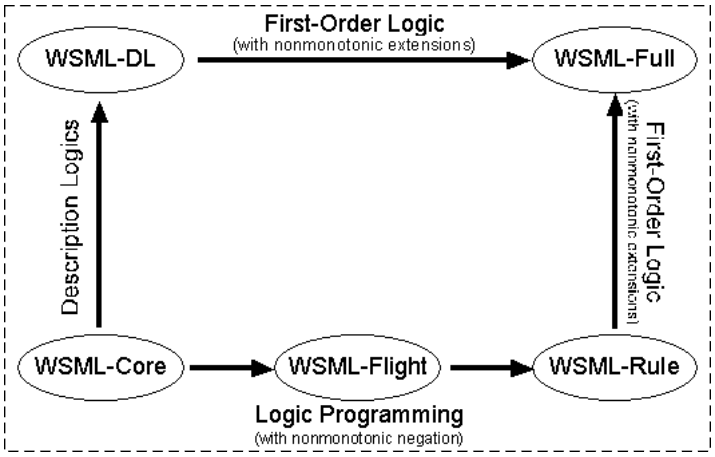


Figure 6: WSML variant space (source: [17])

- *WSML-Core* has the least expressive power of all WSML variants and builds to base for the other variants. Its main features are concepts, attributes, binary relations and instances as well as concept and relation hierarchies. Compared to OWL, WSML-Core can be seen as a subset of OWL Lite. WSML-Core is based on the intersection of the Description Logic $\mathcal{SHIQ}$ and *Horn Logic* based Description Logic Programs.
- *WSML-DL* fully captures the Description Logic of $\mathcal{SHIQ}(D)$. WSML-DL can be understood to be on the same level as OWL-DL.
- *WSML-Flight* provides a powerful rule language. It extends WSML-Core with meta-modeling capabilities, constraints and non-monotonic negation. WSML-Flight is based on a logic programming variant of F-Logic.
- *WSML-Rule* further extends WSML-Flight. Extensions include features from Logic Programming such as function symbols, unsafe rules and unstratified negation.
- *WSML-Full* unifies the power of WSML-DL and WSML-Rule. It supports non-monotonic negation of WSML-Rule but is still an open research issue.

An example of the WSML syntax is given by Listing 1 which shows the wine example introduced in Section 3.2 written in WSML.

```
concept Winery
concept Grape
concept Region
concept ConsumableThing
concept WineGrape subConceptOf Grape
concept PotableLiquid subConceptOf ConumableThing
concept Wine subConceptOf PotableLiquid

instance CentralCoastRegion memberOf Region
instance ChardonnayGrape memberOf WineGrape
instance LidemansBin65Chardonnay memberOf Wine

relation madeFromGrape (ofType Wine, ofType WineGrape)
relationInstance madeFromGrape(LidemansBin65Chardonnay, ChardonnayGrape)

concept WineDescriptor

concept WineColor subConceptOf WineDescriptor
instance White memberOf WineColor
instance Rose memberOf WineColor
instance Red memberOf WineColor

relation hasWineDecriptor (ofType Wine, ofType WineDescriptor)
relation hasColor (ofType Wine, ofType WineColor)
        subRelationOf hasWineDecriptor
```

Listing 1: The wine ontology in WSML

The four top-level elements of WSMO build `Ontologies`, `Goals`, `Web services` and `Mediators` [26].

- A `WSMO ontology` consists of *non-functional properties*, *imported ontologies*, *mediators*, *concepts*, *relations*, *functions*, *instances* and *axioms*.
  Non-functional properties are used to describe non-functional aspects of a WSMO element such as creator, creation date, etc. A non-functional property may be one defined by the Dublin Core Metadata Initiative or others defined by WSMO e.g. the version property. They are allowed in the definition of all WSMO elements.
  Imported ontologies allow the modularization of complex domains of discourse. Importing an ontology makes available all statements of the imported ontology, as long as there are no conflicts between the ontologies.
  Mediators are used for aligning, merging and transforming ontologies. This especially useful in order to resolve mismatches when importing ontologies such as renaming concepts or attributes or the combination of concepts in different ontologies and domains.
  Concepts are the basic elements when defining a terminology of a problem domain and represent classes of objects of a real or abstract world. A member of a concept is called an instance. The signature of a concept is given by its attributes, which are pairs of attribute names and types and represent

named slots for the data values for instances. Concept signatures and constraints on the concept can be inherited from superconcepts by subclassing them. Every instance of a subconcept is an instance of its superconcepts as well.
Relations are used to define interdependencies between concepts, receptively their instances. Each relation has a set of parameters which are single valued and can have a range restriction in form of a concept. A relation may consist of named or unnamed parameter sets. Similar to concepts, relations may have subrelations which inherit and refine the parameter signature of the relation.
A *function* is a special form of a relation where the range specifies the return value. They can be used to represent built-in predicates of common datatypes e.g. the conversion from kilometers to miles.
Instances represent objects in a real or abstract world. Their description follows the signature imposed by the underlaying concept definition.
Axioms can be considered to be logical expressions applied on other elements in an ontology. They are used to capture nuances of meaning of modeling elements in a unambiguous way.

- The `WSMO Web service` element describes all aspects of a Web services in an explicit and unified manner. Such an unambiguous model along with well-defined semantics can be interpreted by machines without any human intervention solving tasks like discovery, selection, composition, mediation, execution or monitoring automatically. A WSMO Web service can be seen as a computational model able to fulfill user goals. The elements of a WSMO Web service are its *non-functional properties*, *imported ontologies*, *mediators*, a *capability* and *interfaces*.
  Beside the WSMO core *non-functional properties*, a WSMO Web service additionally provides non-functional properties describing its quality of service (QoS) values, e.g. Accuracy (the error rate of the Web service), Robustness (functional correctness on invalid inputs), Trust (the trust-worthiness of the Web service) and many others which can be considered on service discovery, selection and negotiation.
  A Web service's *imported ontologies* may be used to provide a formal vocabulary used in the specification of a Web service.
  Mediators are used by a Web service to overcome conflicts between the terminologies of imported ontologies (ooMediator) or to overcome process and protocol heterogeneity when interacting with other Web services (wwMediator).
  The *capability* of a Web service describes its functionality by expressing the state of the world before the service execution (preconditions and assump-

tions) and the state of the world after a successful execution (postconditions and effects).
Interfaces describe two aspects of a WSMO Web service, namely choreography and orchestration. Choreography provides the information of how to communicate with the Web service. Orchestration provides information about other Web services of which a Web service makes use in order to achieve its capability.

- `WSMO Goals` represent objectives and user desires where Web services are said to fulfill certain goals. The elements of a goal definition are *imported ontologies* together with *mediators* for assisting the alignment, merging and transformation between imported ontologies. Further a goal definition may contain *non-functional properties*, a requested *capability* and requested *interfaces*.

- A `WSMO Mediator` can be of one of four different mediator types, namely *ggMediator*, *ooMediator*, *wgMediator* and *wwMediator*. *ggMediators* link two different goals which represents either a refinement of a source goal or state equal, substitutable goals. *ooMediators* import ontologies and resolve mismatches and clashes between ontologies. They may be used by other mediators as well to map different vocabularies. *wgMediators* link Web services to goals expressing that the Web service (totally or partially) fulfills the goal. *wwMediators* link two Web services.

4 The Model Driven Engineering (MDE) Technological Space

Model Driven Engineering is a field of interest in software engineering, that promises a solution to the growing complexity of software systems and to facilitate software evolution and maintainability through the use of *models* as first-class objects. Models provide means for specification, visualization, and documentation of software systems on a high level. *Metamodels* are used as a (semi-)formal specification for models and provide a foundation to enable the application of *transformation rules*. These rules capture expert knowledge how a model can be translated to another model or to another representation format (code) and can then be used to (semi-)automatically build complete software systems out of models. The translation to another model is usually called *model-to-model* transformation, whereas the translation to code is usually called *model-to-code* transformation. In this section at first MDE basics are defined, like *models* and *meta-models*. Then, we will present current approaches, like the Object Man-

agement Group's (OMG) Model Driven Architecture (MDA) standard and its open-source implementation, the Eclipse Modeling Framework (EMF). At last we will present an overview of the View-based Modeling Framework, that is defined using EMF and provides a practical and extensible approach for modeling processes and Service Oriented Architectures.

Model definitions In the literature on MDE one can find different definitions for *model*, we will stick to the following two, because of their prominence (OMG) and their usefulness (Seidewitz).

Definition of *model* by the OMG:

> "A model of a system is a description or specification of that system and its environment for some certain purpose. A model is often presented as a combination of drawings and text. The text may be in a modeling language or in a natural language." [45]

Definition of *model* by Seidewitz:

> "A *model* is a set of statements about some *system under study* [(SUS)]. Here, *statement* means some expression about the SUS that can be considered true or false (although no truth value has to necessarily be assigned at any particular point in time).
> We can use a model to describe an SUS. In this case, we consider the model *correct* if all its statements are true for the SUS. [...]
> Alternatively, we can use a model as a *specification* for an SUS. In this case, we consider a specific SUS *valid* relative to this specification if no statement in the model is false for the SUS." [57]

Meta-model definitions Accordingly to model definitions, also definitions for *meta-model* can be found.

Definition of *meta-model* by the OMG:

> "A metamodel is a model of a model." [45]

Definition of *meta-model* by Seidewitz:

> "A *metamodel* is a specification model for a class of SUS [(system under study)] where each SUS in the class is itself a valid model expressed in a certain modeling language. That is, a metamodel makes statements about what can be expressed in the valid models of a certain modeling language." [57]

Concluding we can say, a model represents (describes or specifies) a system in an abstract way, and thus facilitates understanding and prediction through focusing on relevant aspects for a certain goal. A meta-model is a model of a system of models (and consequently, a meta-meta-model is a model of a system of meta-models).

4.1 Model Driven Architecture (MDA)

Definition

> "MDA is an approach to system development, which increases the power of models in that work. It is *model-driven* because it provides a means for using models to direct the course of understanding, design, construction, deployment, operation, maintenance and modification." [45]

The MDA approach provides a layered architecture that facilitates *direct representation* and *automation* based on *open standards* [22].

Layered architecture The OMG has presented a four layered metadata architecture, for which a simple example is given in Figure 7. Note that, while the four layers are typically referred to in literature, the architecture could consist of an arbitrary number of layers (at least two). These layers are called linguistic layers, and each layer is a linguistic instance-of its level above, with the exception of the M0-layer, which represents the reality or the system under study.

M0 Convention is to begin the naming hierarchy at the bottom, starting with the layer containing runtime instances and naming that layer M0 (or information layer in traditional metadata architectures). A runtime object - in our example a wine object called "Pauillac" is unique in space and time, that is it has a unique object id and it also has memory allocated.

M1 The M1 layer (model layer) abstracts from the runtime instance and captures only the relevant aspects for our purpose, e.g. that a wine generally has a name.

M2 The M2 layer (metamodel layer) then abstracts from the M1 layer, and specifies the relevant aspects as "Attribute" and "Class".

M3 The M3 layer consequently is an abstraction of the M2 layer, in our simplified case all M2 elements are instance of the M3 element ”MOF:Class” [1]. This could be continued, however it is important to know that the topmost layer (MOF) is self-referential, that is, all its elements are expressible by itself.

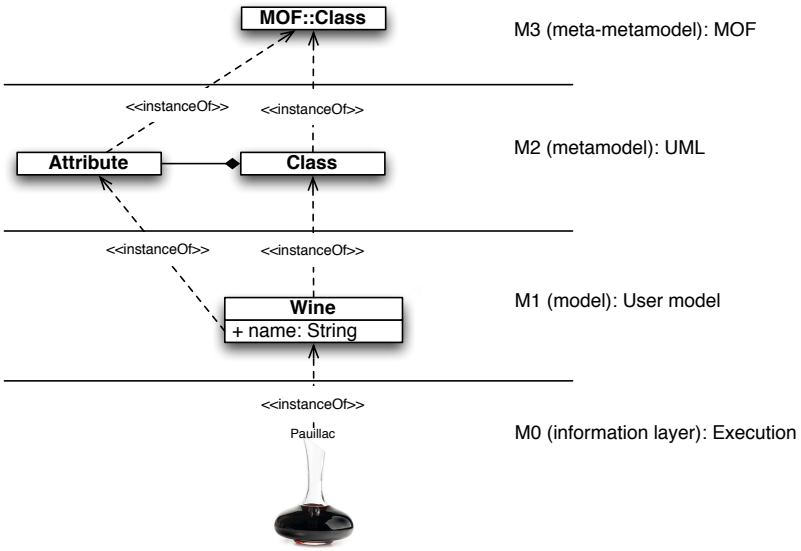


Figure 7: The MOF Metadata Architecture

Direct representation The goal of MDA is to focus on the problem domain instead of the technology domain. Therefore it incorporates the concept of views and viewpoints [38]. A family of notations is provided to model different perspectives, that is different objects of interest on the same abstraction level and/or different levels of abstraction. For different abstraction levels, MDA uses the terms *computational independent model (CIM)*, *platform independent model (PIM)* and *platform specific model (PSM)* [45]. Figure 8 shows an idealized, abstract software engineering process together with these artifacts.

Computational independent model (CIM) A CIM is primary used for communication between domain experts and system analysts. In software engineering the corresponding term is what is known as a *domain model*.

Platform independent model (PIM) A PIM is a view of a system focusing on its operation, independent of a specific platform. Platform independence,

[1] Here called MOF:Class to distinguish from the UML Class.

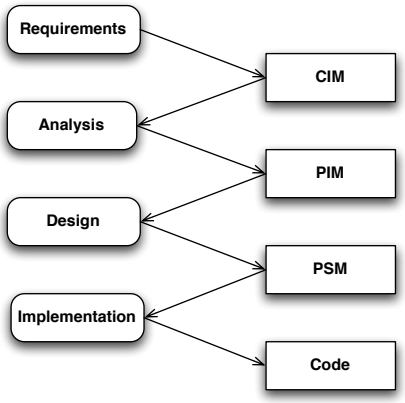


Figure 8: A typical software engineering process with its relationship to CIM, PIM, and PSM

though, is a flexible term, and depends on the goals one wants to achieve. For example, a PIM targeting the Java Virtual Machine (JVM) is (widely) platform-independent with respect to the underlying operating system.
It is not, however, platform-independent with respect to the programming language. Platform-indepence therefore lies - to some extent - in the eye of the beholder with respect to his goals. Nevertheless there may be more than one PIM layer, allowing different levels of abstraction.

Platform specific model (PSM) A PSM at last is a view of a system focusing on the details of that specific platform. The other dimension is the one of different concerns. Typically, different modeling notations of the UML family are used therefore, e.g. Class diagrams for the static structure, and Activity diagrams for system behavior.

Automation In order to provide a consistent way to look on a system as a whole, it is necessary to transform different models. Some of this transformation tasks can be done automatically without human interaction through a set of rules [23]. These rules determine, how

- one model can be *converted* to another model on the same abstraction level (e.g. structural PIM to behavioral PIM),
- one model can be *enriched* with details to provide a lower-level model (e.g. PIM to PSM), and

- one model can be used to *generate* a representation in another format (e.g. PSM to Code).

Open Standards In order to provide a framework that can be adopted by different industry partners, MDA defines/incorporates a set of open standards.

Meta-modeling The Meta Object Facility (MOF) is used to specify meta-models. It provides a common infrastructure to define and integrate different meta-models [46].

Modeling The OMG has defined a set of standards (meta-models) that allow the specification and visualization of software models, of which the UML is probably best-known [49,50]. Others are the Common Warehouse Metamodel (CWM) providing specification of metadata for data warehouses and the Object Constraint Language (OCL), a formal language that allows the description of conditions and constraints on UML models [44, 47].

Model exchange The XML Metadata Exchange (XMI) provides rules in order to map MOF to XML [48], allowing standard serialization and exchange of models and meta-models.

Model transformation To support model transformation, standards for model transformation (MOF Query/View/Transformation, QVT) and code generation (MOF Model to Text Transformation Language) have recently been published [51, 52].

Other standards The OMG has elaborated a set of other standards to cover additional parts of the model-driven process life-cycle, including Model-level Testing and Debugging and MOF Versioning and Development Lifecycle [1].

Conclusion MDA is one, object-oriented approach for MDE. It provides a multi-layer architecture incorporating a set of standards for meta-modeling, modeling, model transformation, model exchange and model lifecycle. MDA does only provide specifications, not an implementation.

4.2 Eclipse Modeling Framework (EMF)

"[MDA] is mostly vaporware." [4]

The Eclipse Modeling Framework [2] is an open source modeling framework implementing (parts of) the OMG MDA standard. The EMF language for defining models is called *Ecore*, a java-based implementation based on the Essential MOF (EMOF) subset of MOF 2.0 and is partially shown in Figure 9. It allows the definition of structural models, like UML class diagrams and XML Schema definitions [6].

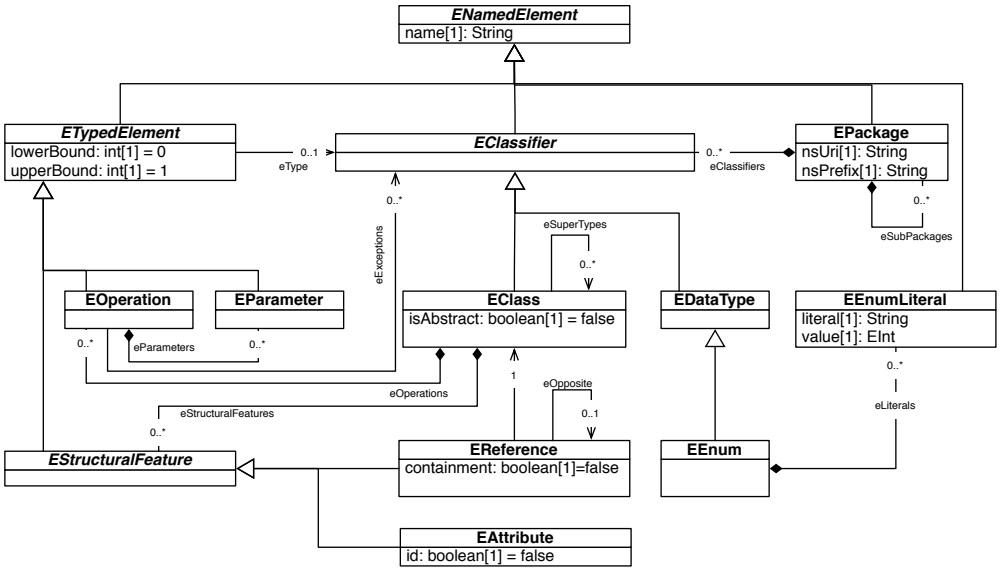


Figure 9: Ecore key types

Meta-model definition Ecore can also be used to define domain specific metamodels [5]. This can be done using the graphical Ecore editor shipped with EMF, XML (due to the use of XMI as model serialization format), or annotated Java. After defining a generator model, code is generated using Java Emitter Templates (JET).

Model definition Model definition can be done using generated model/editor code or again through the use of XML.

Model transformation Model transformations means to describe model transformation rules (aka mapping rules) between a source and a target metamodel. In other words, a mapping is described on the abstract syntax given by the different metamodels, which often is a tedious and complex task and requires knowledge and understanding of both target and source metamodel.

These transformation rules are then executed on the model in order to transform the source model into the target model. Further, one can distinguish between horizontal and vertical transformations on models, where a horizontal transformation transforms a model into a semantically corresponding model and a vertical transformation transforms a model into a model on another level of abstraction.

The Generative Modeling Technologies (GMT), a family of Eclipse research incubator projects, provides tools facilitating model transformation and code generation among other things. Model-to-model transformation, for example, can be done using the ATLAS Transformation Language (ATL), a part of the ATLAS MegaModel Management (AM3) [2] project. The openArchitectureWare[3] toolset provides a language family that facilitates model-to-text (Xpand), model-to-model (Xtend), and text-to-model (Xtext) transformations.

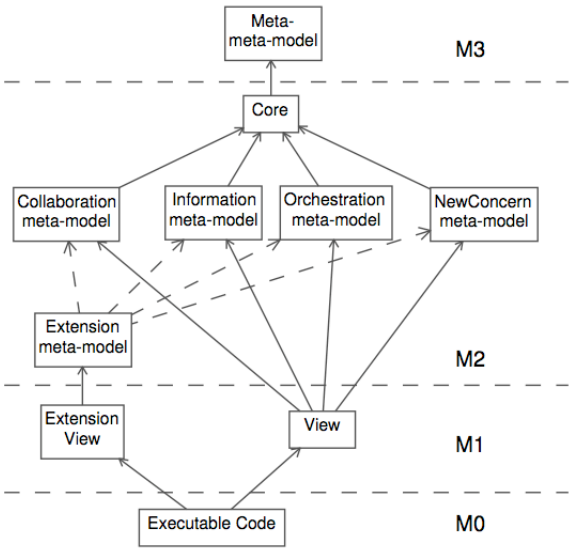
4.3 Viewbased Modeling Framework (VbMF)

The *Viewbased Modeling Framework (VbMF)*, implemented with EMF, is a modeling framework designed to model process-driven service-oriented architectures. In particular, the VbMF makes use of the idea of *architectural views*. An architectural view is a representation of a whole system from the perspective of a particular *viewpoint*, i.e. a related set of concerns. It represents a specific part of a systems architecture which is of value and interest to the stakeholders of the system. Splitting up a systems architecture in different views divides the overall system concern and allows a comprehensive reduction of complexity [38, 59] .

From the MDE point of view, the VbMF provides a core set of meta-models, namely *Core*, *Controlflow*, *Collaboration* and *Information* which are used to model platform independent models and can further be extended to model platform specific or other platform independent models.
As shown in Figure 10, *Ecore* acts as the meta-meta-model in the M3 layer used to define the VbMF views (aka. meta-models) in the M2 layer. In addition all extensions of the framework reside in the M2 layer. The actual models are part of the M1 layer.

The following sections describe the extension mechanisms of the VbMF as well as its fundamental platform independent views, namely *Core View*, *Controlflow View*, *Collaboration View*, and *Information View*.

[2] http://www.eclipse.org/gmt/am3/
[3] http://www.openarchitectureware.org/



(a) Modeling elements

Figure 10: Meta levels of the View-based Modeling Framework (Source: [59])

4.3.1 Core View and extension mechanisms

The fundamental view of the VbMF is the *Core View* which represents a common meta-model and defines the basic concepts of View, Process, Service and others, as shown in Figure 11.

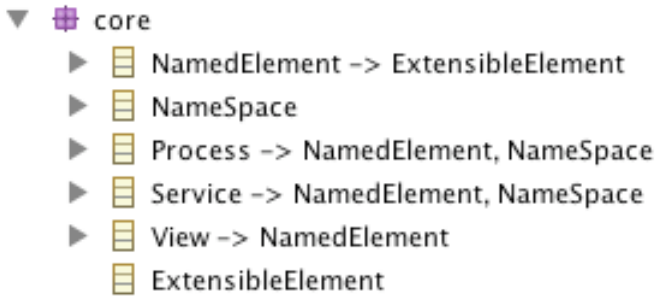


Figure 11: VbMF Core View (Generic EMF Editor)

The core view enhances the extensibility of the framework and acts as a foundation for other views. Each extension view directly or indirectly extends the core view. Each view meta-model can be further extended and refined via *extension points*.

> "An extension point is any entity that can add additional features (e.g., attributes or relations) to construct a new entity. Using relationships, such as generalization, extend, etc., we can gradually refine an existing metamodel toward another metamodel at a lower abstraction level." [28], p. 24

Separate views can be integrated to provide a comprehensive view on a process via *integration points*. In order to retrieve the integration points, the VbMF uses a name-based matching algorithm. These transformations between views are specified on the meta-model (M2) and executed on the actual models (M1). The concrete model-to-model transformations are specified in the Xtend language, the transformations from models to code is done with model-to-text transformations using Xpand.

When representing new concerns, this is typically done through deriving from the core meta-model and starting from scratch. These views can be used to define platform-independent models of business-processes.

Further, these views may be extended to capture additional features or to enable platform-specific representations. Concretely, within the VbMF this it is done for representing models of WS-BPEL processes. The corresponding BPEL views are extension views that capture BPEL specific concepts (for example XSD definitions for data definition, and Web services for service interaction).

4.3.2 VbMF ControlFlow View

The *ControlFlow View*'s concern, as its name suggests, is to define the flow of activities. Consequently, the *ControlFlowView* meta-model defines the necessary control structures (see fig. 12). Its primary entity is the `ControlFlowView` which extends the core `View` and contains exactly one (abstract) `Activity`. Each activity has a name and may be connected to other activities via its in- and outgoing `Link`. Basically, there are two types of activities: structured activities (sequences and flows) and unstructured activities (simple activities and switches). The most simple non-abstract activity is the atomic `SimpleActivity` derived from the abstract `Activity`. The abstract `StructuredActivity` is a generalization for activities that act as a container for possibly more than one subactivities. There are two non-abstract derivations of `StructuredActivity`, namely the `Sequence`, which semantically guarantees the order of contained activities, and the `Flow`, describing activities that may take place parallel.
The `Switch` activity acts as an container for one or more `Cases` and may contain an activity carried out if non of the cases can be applied. A `Case` itself has

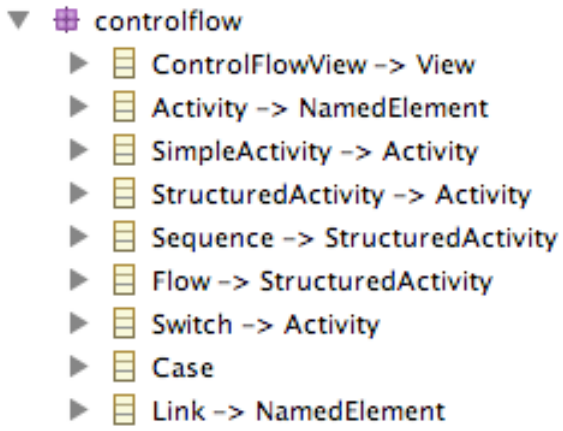


Figure 12: VbMF ControlFlow View (Generic EMF Editor)

a condition (as a string) and acts as a container for exactly one activity, that is carried out if the condition applies. The resulting object-structure is a tree, with the main activity as its root and simple activities as leaves.

4.3.3 VbMF Collaboration View

As almost every business process faces interaction with other parties or at least is used by another party, the collaboration between partners represent a vital concern and thus draws the need for a specific architectural view. To model the various interactions between participants in a process execution the *Collaboration* view provides elements for modeling partners and their service interfaces along with the description of the various interaction details necessary (see fig. 13). Its primary entity is the `CollaborationView` which extends the core `View` and acts as a container for `PartnerLinks`, `PartnerLinkTypes`, `Roles`, `Services`, `Interfaces` and `Interactions`. Partner descriptions typically consist of a `PartnerLink` along with its corresponding `PartnerLinkType`. Each `PartnerLink` has a name and may link `Roles` via the `myRole` and the `partnerRole` attributes, it refers to exactly one `PartnerLinkType` as well as to a set of `Interactions`.

A `PartnerLinkType` is responsible for the definition of the various partner correlations through linking available `Roles` with `PartnerLinks`. Together with an extension of the core's `Service` meta-class that contains appropriate interface descriptions in form of `Interface` elements a partner definition becomes complete. `Interfaces` are linked to one or more `Roles`, moreover an interface description can contain numerous `operations` that in turn describe their inputs, outputs and faults in terms of `Channels`. Finally the `Interaction` element acts a platform independent container of the actual interactions of a business

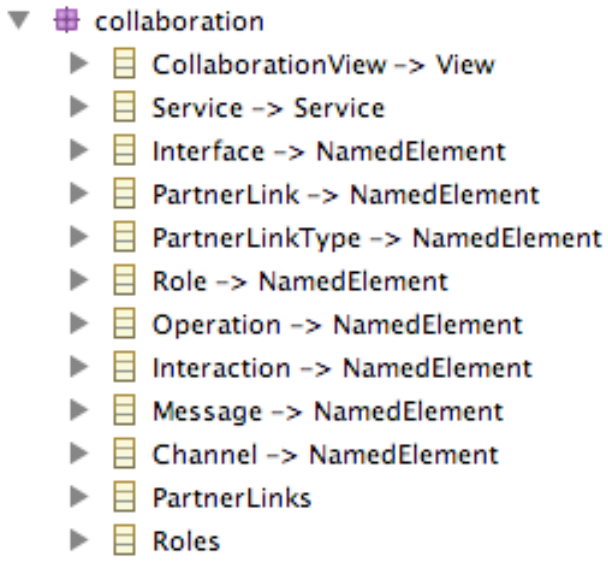


Figure 13: VbMF Collaboration View (Generic EMF Editor)

process solely linking to the providing `PartnerLink`.

4.3.4 VbMF Information View

The *Information View* meta-model handles data definition and transformation for business objects (see Figure 14).

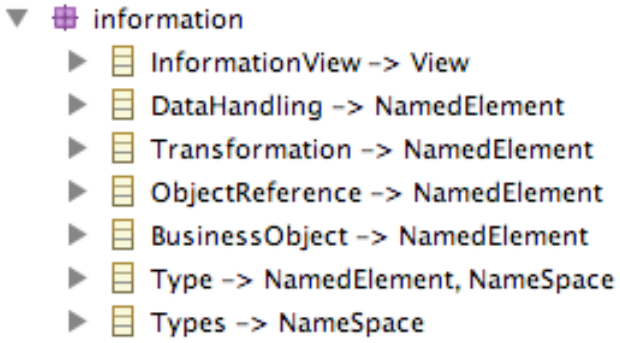


Figure 14: VbMF Information View (Generic EMF Editor)

The `InformationView` extends the core `View` and acts as a container for `BusinessObjects`, `DataHandlings`, as well as the `Types` container holding distinct `Types`. A `BusinessObject` represents a real-world business data like e.g. an account, an order or a customer, and it is typed by a reference to the corresponding technical `Type` which is defined by its name and its

corresponding namespace. A `DataHandling` is a container for one or more `Transformations`, which represent data manipulations within the process flow by a name and corresponding source and target `ObjectReferences`. At last, these `ObjectReferences` act as pointers to `BusinessObjects`.

5 View-based Ontology Integration Process

Having now introduced the relevant technological spaces we continue to cover the problem of getting from ontologies to executable code. Therefore we suggest and overall integration process shown by Figure 15 which is made up of three process steps representing the coarse development tasks when integrating ontologies into the MDE layer.

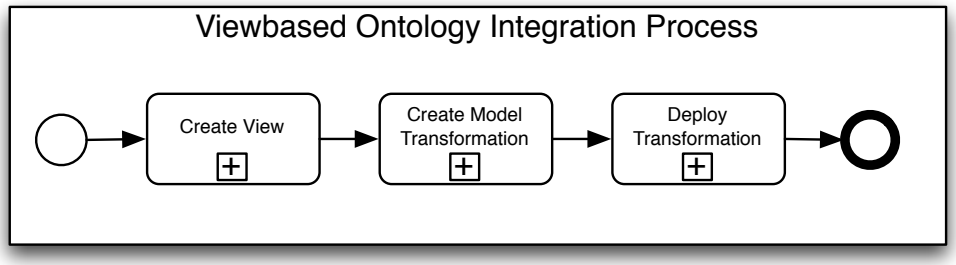


Figure 15: High-level View-based Integration Process

The first subprocess, Create View, is concerned with overcoming the divergences between ontologies and models. There are several crucial differences between those approaches which need to be addressed in order to import the source ontologies. Section 6 provides a discussion of the fundamental differences and shows how to overcome them in order to create an equivalent counterpart in form of a metamodel as well as adequate import functionalities.

In the second process step, various model-to-model transformations are created in order to translate models conforming to the before created metamodel into other models. When creating model transformation rules, developers must have a solid understanding of the problem domain and the involved models. Further, a transformation can become very complex and the developers face a variety of design decisions. Section 7 discusses the main issues when deriving model correspondences and shows how to apply some crucial transformation patterns.

In the final step the implemented transformation is required to be integrated with existing process landscapes. Section 8 shows how single transformation components can be orchestrated and finally exposed as a new transformation service.

6 View Creation

As ontologies are conceptual models they are somewhat similar to computational independent models (CIMs) in MDE, both can be used to describe domains in an abstract way. The main differences to MDE approaches are that ontologies shall allow unambiguous specification of semantics, and usually come with tools that enable querying and reasoning.
However, both ontologies and model driven approaches mostly define structures like classes, instances, inheritance, relations and attributes. It therefore seems obvious to use model-driven approaches to define and/or represent ontologies. Further, both technologies have been given a lot of attention in the last years, and there is a lot of ongoing research focusing on converging both approaches and outlining differences between them in Section 6.1.

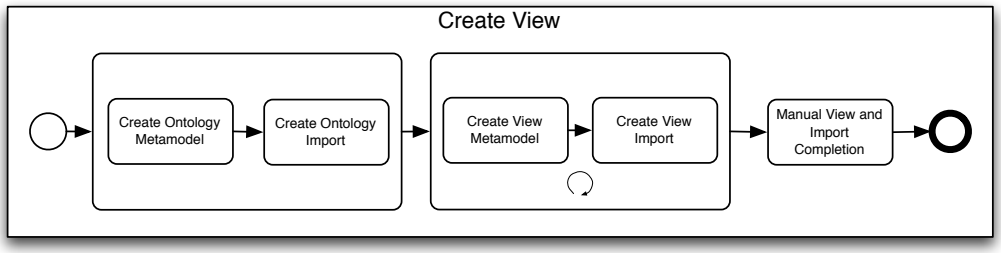


Figure 16: Create View subprocess

The knowledge gained by researching related work in the area will guide us to (semi-)automatically create model-based views on ontologies and populate them from WSML ontologies, this process is shown in Figure 16.
The first step in this process is to build an integrated ontology meta-model, able to express the details the ontology language is providing, and an importing facility that is capable of loading files of that language. This will be described in Sections 6.2 and 6.3. However, this is a very fine-grained level, so that model transformations are tedious to write and harder to maintain.
In order to reduce complexity and improve maintainability, additional views can be created, that focus on the representation of separate concerns of the ontology. One example is a conceptual view, for which we will show how they (and corresponding importing facilities) can (semi-)automatically be built by applying model-transformations to the ontology meta-model in Sections 6.4 and 6.5 .
We state semi-automatic, as the conceptual view is while still being generic already usable in this state (i.e. editor generation is possible and the importer facility populates the view), but has to meet some additional requirements to meet our concrete use case, e.g. the preservation of references to the original ontology. We

will discuss our modifications to the conceptual view in Section 6.6.

6.1 Related Work

There are numerous projects that try to combine the Ontology TS and the MDE TS, of which some will be outlined in this section, namely the OMG's Ontology Definition Metamodel (ODM), the Ecore Ontology Definition Metamodel (EODM) and the ModelCVS project. ODM and EODM aim at defining a complete MOF/Ecore based metamodel for ontologies, allowing to model ontologies in the MDE TS.
However, as ontologies aren't in widespread use for Software Engineering yet, they define mappings from and to UML/Ecore, an approach that is also picked up by the ModelCVS project. As these mappings address the first problem of translation between ontologies and executable code, we will collect those issues and summarize them in the next section.

6.1.1 Ontology Definition Metamodel (ODM)

The Ontology Definition Metamodel (ODM) is an upcoming OMG standard for defining vocabularies for Semantic Web technologies, such as Resource Description Framework (RDF), Web Ontology Language (OWL), Topic Maps, and Common Logic (CL) and their integration with the Model Driven Architecture. Therefore it defines a set of normative, MOF-based meta-models for each of these technologies.
To enable translation between models based on these new meta-models as well as existing meta-models such as UML and CWM, the standard additionally provides two mechanisms, namely normative *UML profiles* and informative *mappings*.
UML profiles are specified for RDF/OWL and topic maps. As a small-footprint approach, they allow the the reuse of existing (profile-capable) UML-modeling tools for ontology development, and the reuse of knowledge of existing UML-models to be easily translated to ontologies. While this translation can be used as a basis for ontology engineering, the translation provided can usually only be used as a starting point, and is not complete.
The mappings provided are specified in QVT and allow a transformation of models conforming to one meta-model to models conforming to another one. Figure 17 shows an overview of the mapping of common features. A problem with mappings is, that not always one element of a model conforming to one meta-model can be directly mapped to one element of another, this is what OMG calls *structure loss*. Additionally, the provided mappings are very general, so that their straight-forward application may result in unwanted results. The idea of the given

mappings is to be informative, that is, to use them as a starting point but to customize them for particular models (cp. [53]).

UML elements	Package	OWL elements	Comment
class, property ownedAttribute, type[a]	7.3.7 Classes 7.3.8 Classifiers 7.3.32 Multiplicities	class	
instance	7.3.22 Instances	individual	OWL individual independent of class
ownedAttribute, binary association	7.3.7 Classes	property	OWL property can be global
subclass, generalization	7.3.7 Classes 7.3.8 Classifiers	subclass subproperty	
N-ary association, association class	7.3.7 Classes 7.3.4 Association Classes	class, property	
enumeration	7.3.11 Datatypes	oneOf	
disjoint, cover	7.3.21 Generalization sets	disjointWith, unionOf	
multiplicity	7.3.32 Multiplicities	minCardinality maxCardinality	OWL cardinality declared only for range
package	7.3.37 Packages	ontology	
dependency	7.3.12 Dependencies	reserved name RDF:property	

Figure 17: Common features of UML and OWL (source: [53], p. 194)

6.1.2 EMF-based Ontology Definition Metamodel (EODM)

The EMF-based Ontology Definition Meta-model (EODM) is an open source [4] implementation approach for the OMG ODM based on the Eclipse Modeling Framework. It's goal is to provide a comprehensive tool suite for model driven ontology engineering and model driven software engineering, that is familiar to software developers and thus facilitates quick habitation of Semantic Web technologies.
The tool suite consists of a core object model, the EODM Ecore model, an OWL parser, model transformation facilities and an OWL Editor. The EODM Ecore model is an implementation of the ODM OWL meta-model and represents the core of the tool suite. The OWL parser provides parsing, RDF-triple building and inference of OWL files, reading them into the EODM Ecore model. Additionally it provides serialization for models to OWL ontology files. Using the model transformation facilities, UML class diagrams can be converted to OWL models

[4] As of October 8th, 2008 the project has been terminated: http://www.eclipse.org/projects-slides/EODM%20Termination%20Review.pdf

and vice versa. Transformation is done by importing a UML class diagram into an Ecore model, and then using a mapping scheme to transform Ecore to ODM respectively the other way round. An overview of the mapping is shown in Figure 18. Because of different background of UML and OWL, these transformations are usually encompassed with a loss of semantics. Finally the OWL editor, called EODM workbench, can be used to create new or augment existing (imported/-transformed) OWL ontology files (cf. [55]).

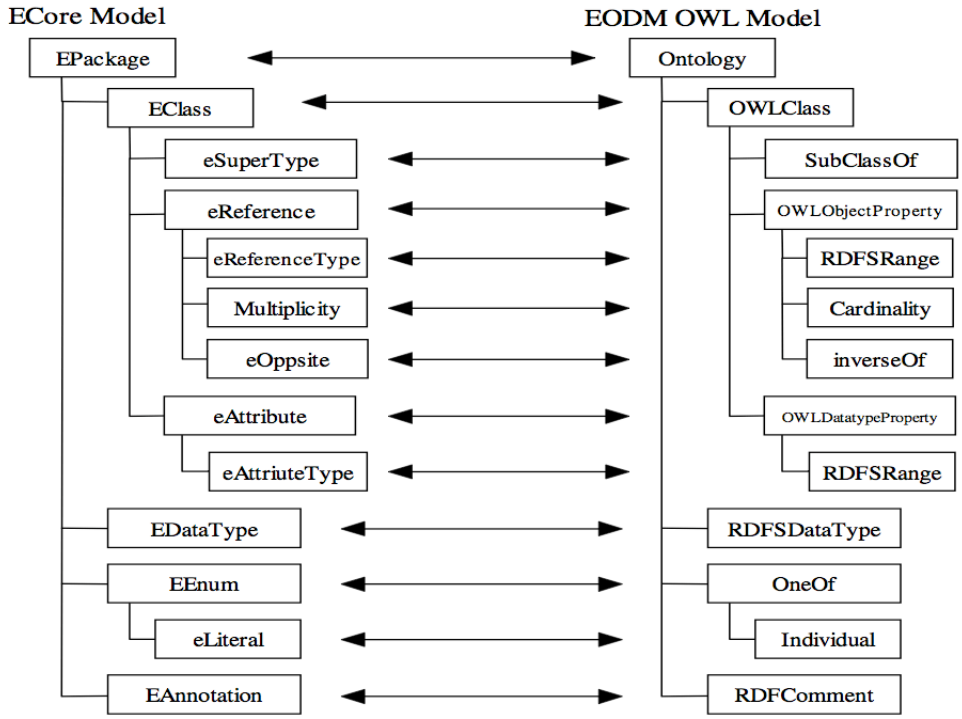


Figure 18: Transformation between Ecore and EODM OWL (source: [55], p. 71)

6.1.3 ModelCVS

The ModelCVS project [5] aims at providing a semantic infrastructure for model-based tool integration. Part of this is an approach to automatically derive OWL ontologies from Ecore based meta-models, which is called *lifting process*, of which a summary is shown in Figure 19. The OWL representations then simplify integration as they concentrate on the conceptual level of a modeling language only. They further can be used to facilitate reasoning which benefits semantic integration (cf. [63]).

[5] http://www.modelcvs.org/

Ecore Concept	OWL Concept	Possible Caveat
EFactory, EOperation, EParameter	*no mapping*	*ignored*
EPackage	OWLOntology	inverse hierarchy
EClass	OWLClass	non-exclusive instanceof
EAttribute	OWLDatatypeProperty	name clash / qualification
EReference	OWLObjectProperty	name clash / qualification
EDatatype	RDFSDatatype	*straight-forward*
EEnum & EENumLiteral	OWLDataRange & RDFSLiteral	*straight-forward*
EAnnotation	RDFSLiteral	*straight-forward*

Figure 19: ModelCVS Ecore to OWL mapping (source: [63], p. 532)

6.1.4 Differences between Ontologies and MDE

As a result of the last sections, in this section the differences between the Ontology TS (espically OWL and WSMO) and the MDE (particularly UML and Ecore, resp.) will be outlined and explained.
General problems arise from the fact, that the Ontology TS and the MDE TS have different backgrounds and consequently a different view on the world. While the Ontology TS is rooted in Artificial Intelligence and Logical Programming, the MDE TS comes from classical Software Engineering. Both, Ontologies and Software Models are usually used to capture knowledge but with a different focus, the first on generating new knowledge, the latter on generating implementations. The arising differences and their impact will be outlined in this section.

Different scope Some issues result from the fact that MDE and ontologies are built for different purposes, and use different underlying semantics (cf. [63]). Where MDE typically has an implementation focus, ontologies are used for knowledge representation. Consequently, in MDE sometimes concepts are modeled as attributes using primitive types, while in ontologies this would make no sense. On the other hand, abstract classes are used in MDE for implementation purposes, but are hardly useful in ontologies. However, when considering the context, that ontologies focus on knowledge representation, and MDE approaches usually focus on data representation this can usually be dealt with when designing the transformation rules.

Layering problem In ontologies, classes and instances typically reside on the same layer, and they don't have a strict separation of meta-levels, resulting in ontology elements almost naturally crossing meta-levels. That is, a class may have

instances which are classes themselves[6]. This problem is sometimes called *layer mistake* (cf. [53, 54]). As opposed to RDF/OWL, WSMO has a quite metadata architecture, that is a concept never is an instance and vice versa, thus avoiding the so called *layer mistake*. However, as can be seen in Figure 20, concepts and instances are used on the same layer. In contrast to the MDA approach, where instances would only occur on a layer below classes, this is not the case with ontologies. What we call a *conceptual* ontology usually defines concepts and relations, and *quasi-model* ontologies define instances which are *memberOf* those concepts and relations. However, the first may also define instances and the latter may also define new concepts, so strictly speaking, using the terms *meta-model* and *model* is wrong.
There is no normative mapping approach for these problems, for the same reasons as those mentioned in Section 6.1.1. One typical occurrence of the first problem, instances in the conceptual ontology, are enumerations using instances. We will show how this can be solved in Section 6.4. The second problem can be solved as well, one approach is to use the meta-model and considering this concepts as instances of this structure.

Structure conflation and structure loss OWL does not support constructs for binary or N-ary relationships. Neither Ecore nor MOF support N-ary relationships as well, whereas UML and WSMO define N-ary relations. When mapping these UML/WSMO constructs to OWL/Ecore, this complex constructs have to be simulated by simpler ones. The translation from a complex construct to two simple constructs is called *structure conflation* (as it is the case with binary associations), the translation from a complex construct to a group of simple constructs is called *structure loss* (cf. [53]). For a transformation to be reversible, then one needs to distinguish between classes representing classes and classes representing relations.

Feature Lack In general ontologies are much more expressive than MDE approaches, e.g. they allow anonymous classes defined by inference only. For example, in OWL there are six ways to describe a class: by a name, through enumeration of individuals, through a restriction on properties, and through intersection, union and complement of classes (cp. [53, 63]).
In contrast to OWL, the WSMO ontology only uses named specification for concepts, identifying this variant as ”necessary and sufficient” [25]. Axioms can be used to model enumeration, restriction, intersection, union, and complement on instances.

[6]Possible e.g. in OWL-Full, cp. [11]

However, practically speaking, this is not necessarily an issue. One can agree on semantically poorer variants (e.g. OWL-Lite, WSML-Flight) as a starting point.

Open world vs. Closed world semantics While model-driven approaches are typically built on a closed world assumption, semantic web technologies usually use the open world assumption. To illustrate the difference, we provide a simple example (in RDF like syntax).
Considering the statement

```
<#clemens> <#livesIn> <#Vienna>
```

as the one and only in a knowledge base, the query

```
<#clemens> <#livesIn> <#Graz> ?
```

would result in a ”false” under closed world semantics, whereas an open world semantics based system would answer ”undefined”. This is because, in a closed world, everything unknown is false, in an open world, everything unknown is undefined (cf. [37]). Pragmatically, the different semantics mostly don't play a role. Nevertheless, if it has to be they can be solved (1) on the ontology side, using the construct of *negation as failure* (NAF) in axioms which return *false* where otherwise *unknown* would have been the answer, and (2) on the MDE side, introducing *Special Case*s such as *Unknown* as part of the model (cp. [30,37,53]).

6.2 Create Ontology Metamodel

The Web Service Modeling Ontology (WSMO, cp. 3.3) has been specified as a language in Extended-Backus-Naur Form (EBNF), called Web Service Modeling Language (WSML) and a - semantically equivalent - object-oriented meta-model in MOF, its ontology top element is shown in Figure 20.

In order to be able to (semi)-automatically define an Ontology View as well as an importing facility, one has to be able to parse the corresponding ontologies. For WSMO, this could be done using the WSMO Java API [7]. However, writing Java code for these purposes is quite cumbersome. Fortunately WSMO has an XML representation, along with a corresponding XML Schema, that can be used by EMF to derive an Ecore model, as shown in Figure 21. An excerpt of the result is shown in Figure 22.

[7] http://wsmo4j.sourceforge.net/

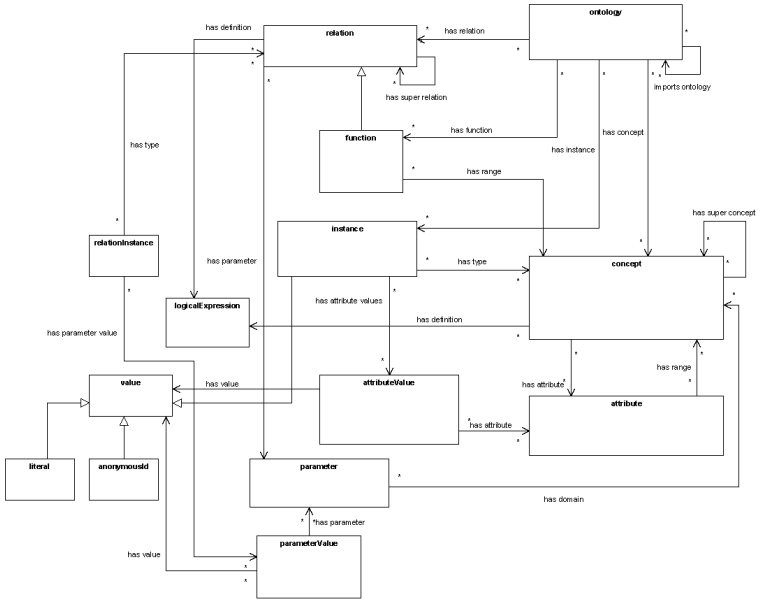


Figure 20: The WSMO Metamodel in MOF (source: [25])

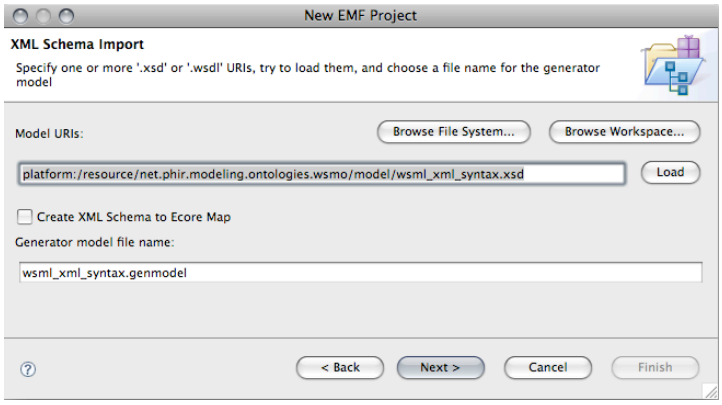


Figure 21: Import of XML Schema into an EMF Project

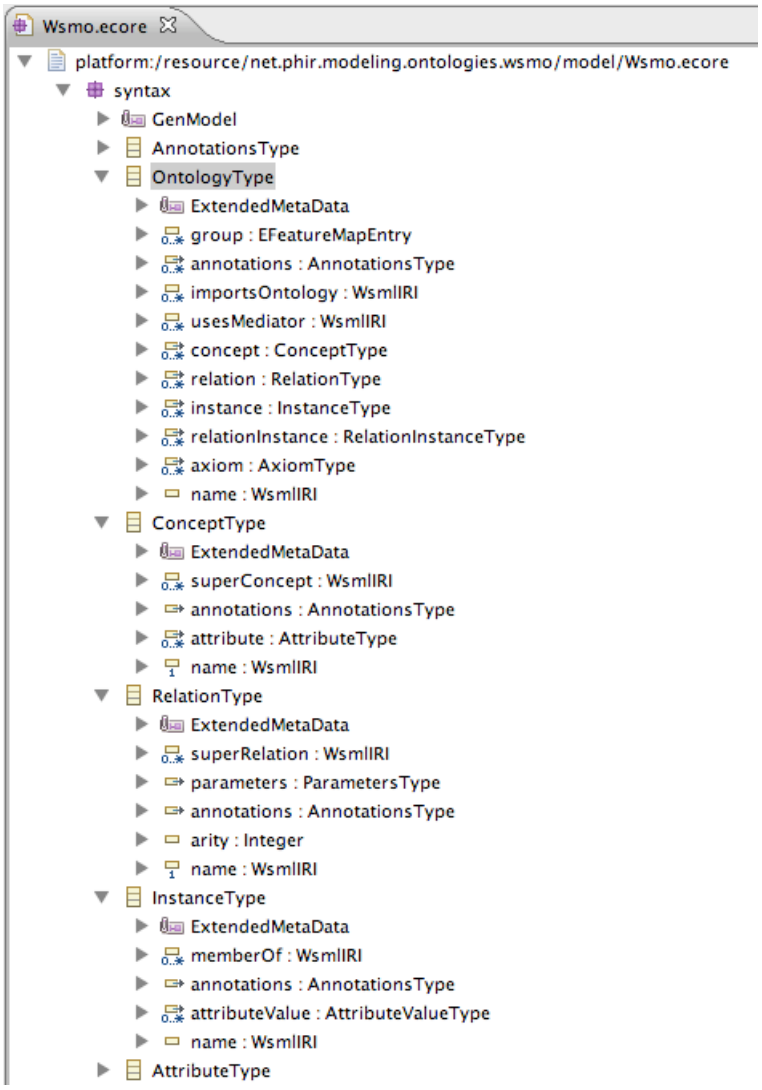


Figure 22: The WSMO Metamodel in Ecore, excerpt (Generic EMF Editor)

```
<xsl:template match="/">
        <xmi:XMI xmlns:xmi="http://www.omg.org/XMI" xmi:version="2.0" ...>
                <xsl:apply-templates select="*"/>
        </xmi:XMI>
</xsl:template>

<xsl:template match="*">
        <xsl:copy>
                <xsl:copy-of select="@*" />
                <xsl:apply-templates/>
        </xsl:copy>
</xsl:template>
```

Listing 2: XSLT translating WSML-XML to XMI conforming to WSMO.ecore

6.3 Create Ontology Import

Importing WSML ontologies into the WSMO Ecore metamodel created from the WSML XML Schema can then be done by wrapping their XML representation within an XMI header, which is a simple XSLT transformation, shown in 2.

6.4 Create View Metamodel

Though both the Ontology TS and the MDE TS have a different view on the world, they also incorporate constructs that have similar names and are used for similar purposes, e.g. Modularization, Classification, and Description. The goal of this section is to compare these related constructs and apply mappings to (semi-)automatically create a conceptual view of the imported ontologies using ATL model transformation rules, an conceptual overview of the transformation is shown in Figure 23.

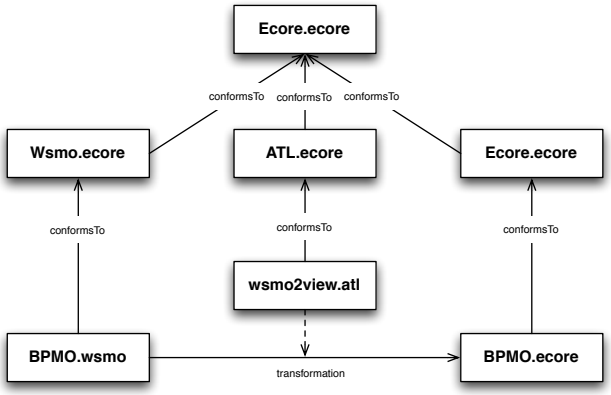


Figure 23: Transformation setup of the Create View Metamodel activity

```
...

-- Helper functions for IRI

helper context String def : getNamespace() :String =
        self.substring(1, self.indexOf('#')+1).trim();

helper context String def : getLocalName() :String =
        self.substring(self.indexOf('#')+2, self.size()).trim();
...

-- Helper function(s) for getAllSubConcepts()

helper context Wsml!ConceptType def : getAllSubConcepts() :
                Set(Wsml!ConceptType) =
        self.getDirectSubConcepts() -> collect ( e | e.getAllSubConcepts()) ->
         flatten() -> asSet() -> union(self.getDirectSubConcepts());

helper context Wsml!ConceptType def : getDirectSubConcepts() :
                Set(Wsml!ConceptType) =
        thisModule.getAllConcepts() ->
         select (e | e.superConcept.includes(self.name));
...

-- Helper function(s) for memberOf

helper context Wsml!InstanceType def : isMemberOf(concept : String) : Boolean=
        if self.memberOf.includes(concept)
        then true -- direct member
        else thisModule.isSubconceptOf(self.memberOf, concept) endif;

helper def : isSubconceptOf(concepts:Set(String),concept:String):Boolean =
        if not thisModule.conceptTypeExists(concept)
        then false
        else thisModule.getConceptType(concept).getAllSubConcepts() ->
                collect (e | e.name) -> flatten() -> asSet() ->
                intersection(concepts).size() > 0 endif;
```

Listing 3: ATL helper rule library (excerpt from wsmoUtil.atl)

Helpers One important building block of handling ontologies are reusable libraries providing basic functions, e.g. to get namespace and local name components of an IRI, find all sub concepts of a given concept, or determine if an instance is member of a given concept, as shown in Listing 3.

Mapping Primitive Types Primitive types (WSMO wraps the XML primivite types) can be mapped to EDataTypes. However, they are not part of the model source (i.e. the ontology), but of the meta-model (in fact, WSMO uses XML Schema primitves), so they cannot be resolved using model information. Therefore primitive types are created by a called rule once for a transformation, as can be seen in Listing 4.

```
rule createEDataType (ecoreDataType : String){
        to t : Ecore!EDataType(
                name <- ecoreDataType,
                instanceTypeName <-
                        thisModule.getMapEcorePrimitives().get(ecoreDataType)
        )
        do {
                t;
        }
}

helper getMapEcorePrimitives() : Map(String,String) =
 Map{('boolean','EBoolean'), ('string','EString'),  ...}
```

Listing 4: ATL rule translating Ontologies to EPackages

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace { _"http://www.sembiz.org/bpmo/bpmoOntology#",
     dc _"http://purl.org/dc/elements/1.1#" }

ontology bpmoOntology

     nonFunctionalProperties
          dc#title hasValue {"BPMO-Business Process Modeling Ontology"}
     endNonFunctionalProperties

// ... concept definition
```

Listing 5: The BPMO Ontology definition and header in WSML

Mapping Ontologies Both ontologies and packages provide modularization and act as a container for elements. However, *ontology imports* cause all contained elements to be visible to the top-level ontology, whereas a package has the opposite semantics: a sub-package is aware of all its super-packages (cf. [63]). Listing 5 shows the BPMO Ontology definition and header in WSML, Listing 6 shows the ATL rule that performs the mapping.
For each ontology an EPackage is created, its name and namespace determine the package's name and namespace URI, resp. The ontology's concepts and relations will be contained as the package's classifiers. The result of the mapping is shown in Figure 24.

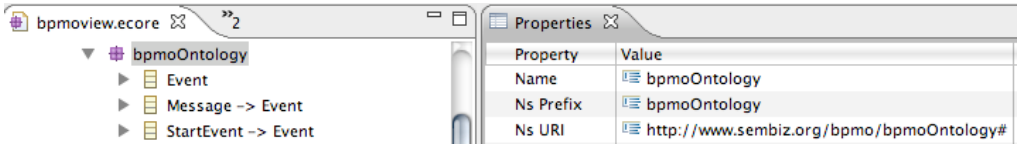


Figure 24: The BPMO Ontology as an EPackage (Generic EMF Editor)

```
rule ontology2epackage {
        from o : Wsml!OntologyType
        to p : Ecore!EPackage (
                 name <- o.name.getLocalName(),
                 nsURI <- o.name.getNamespace(),
                 nsPrefix <- o.name.getLocalName(),
                 eClassifiers <- Set{
                         o.concept,
                         o.relation
                 }
        )
}
```

Listing 6: ATL rule translating Ontologies to EPackages

```
ontology bpmoOntology
...

concept Event
   hasName ofType _string

concept StartEvent subConceptOf Event

concept Message subConceptOf Event
   hasData ofType (0 *) BusinessData

concept StartMessage subConceptOf {StartEvent, Message}

...
```

Listing 7: Concepts in WSML

Mapping Concepts In OWL, a class is a set of zero or more individuals. In UML, a class is a more general construct, also used for implementation purposes (e.g. abstract classes, that are used for implementation inheritance). In UML, an instance is instance of exactly one class, whereas in OWL an individual can be instance zero or more classes.
Similar to OWL classes, WSMO concepts can be mapped to EClasses. A problem arises in mapping instances that belong to more than one concept , this can be emulated by deriving a new EClass inheriting from those superclasses. To enable automation as far as possible, this can be done in the ontology, as shown by the concept `StartMessage` in Listing 7.
The translation rule is shown in 8. In order to set the super classes, it uses a helper that retrieves the direct super concepts of the actual one. The result of this rule is shown in Figure 25.

Mapping Attributes In MOF, UML, and Ecore, attributes and references belong to a class, whereas in OWL a property is independent of a certain class.

```
rule concept2eclass {
        from con : Wsml!ConceptType (
                not con.isEnumCandidate()
        )
        to cla : Ecore!EClass (
                name <- con.name.getLocalName(),
                eSuperTypes <- Set {
                    con.getDirectSuperConcepts()
                },
                eStructuralFeatures <- Set{con.attribute -> flatten()}
        )
}
```

Listing 8: ATL rule translating Concepts to EClasses

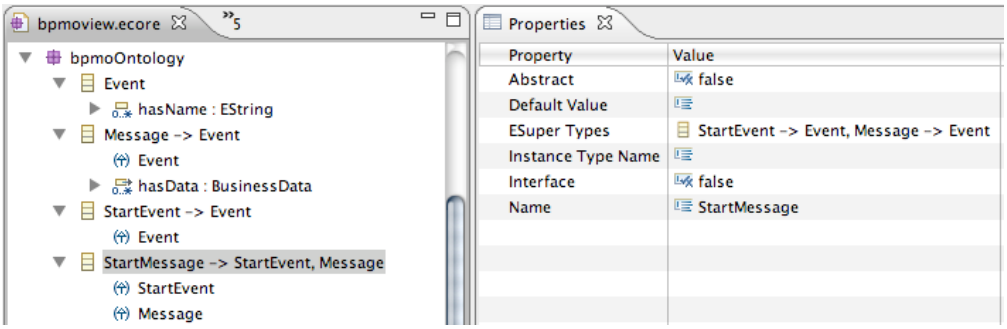


Figure 25: Mapping concepts to EClasses (Generic EMF Editor)

This may lead to name clashes when translating from OWL to MDA if different classes have attributes/references with the same name.
This is per definition no problem when translating from OWL to MDA. When translating from MDA to OWL, this could be solved using a naming convention (e.g. weaving the owning class name into the property name, cf. [53, 63]). In contrast to OWL, attributes in WSMO are defined local to a class, thus avoiding name clashes. Multiple typed attributes can be emulated by deriving a new concept, similar to the strategy for instances in the last paragraph.
Listing 9 shows the rules that perform the mapping of attributes.
The first rule, `attributeType2eattribute`, is responsible for mapping primitive attribute types. As they are not part of the ontology, a mapped rule can't be used to resolve their types. Therefore type and name are temporary encoded in the attribute's name, and resolved by the endpoint called rule `end()` that is executed at the end of the transformation.
The second and the third rule, `attributeType2enumattribute` and `attributeType2ereference` are responsible for mapping attributes that are of concept type. The reason why two rules are needed is because some concepts represent enumerations (see the paragraph on enumerations below). Those attributes then have to be translated to EAttributes, whereas the others have to be translated to EReferences.

Mapping Relations Relations like the one in Listing 10 can be mapped to EClasses, their parameters can be mapped to EReferences. In Ecore, references have to be named, whereas in WSML only the position is relevant and necessary. Therefore the references are enumerated using `thisModule.paramCounter`, as can be seen in Listing 11. The result of the mapping can be seen in Figure 26.

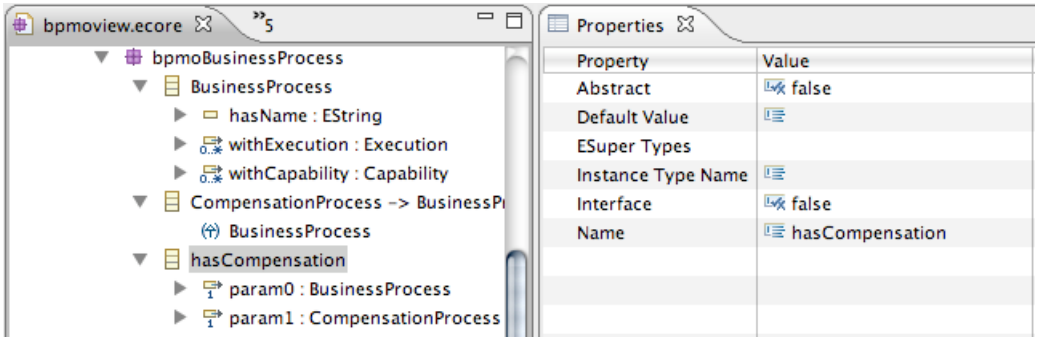


Figure 26: Mapping relations to EClasses and EReferences (Generic EMF Editor)

```
rule attributeType2eattribute {
        from s : Wsml!AttributeType (
                s.isPrimitiveType()
        )
        to t : Ecore!EAttribute (
                name <- thisModule.getBaseType() +
                 s.range+thisModule.getBaseName()+s.name.getLocalName(),
                lowerBound <- s.getMinCardinality(),
                upperBound <- s.getMaxCardinality()
        )
}

rule attributeType2ereference {
        from s: Wsml!AttributeType(
                not s.isPrimitiveType() and not s.isEnumAttribute()
        ) to t : Ecore!EReference (
                eType <- thisModule.getConceptType(s.range),
                name <- s.name.getLocalName(),
                lowerBound <- s.getMinCardinality(),
                upperBound <- s.getMaxCardinality()
        )
}

rule attributeType2enumattribute {
        from s : Wsml!AttributeType (
                not s.isPrimitiveType() and s.isEnumAttribute()
        ) to t : Ecore!EAttribute (
                eType <- thisModule.getConceptType(s.range),
                name <- s.name.getLocalName(),
                lowerBound <- s.getMinCardinality(),
                upperBound <- s.getMaxCardinality()
        )
}

endpoint rule end() {

        do {
                for (ea in Ecore!EAttribute.allInstances()->
                 select(e | e.name.startsWith(thisModule.getBaseType()))) {
                  ea.eType <- ea.getEDataType();
                  ea.name <- ea.getReducedName();
                }
        }
}
```

Listing 9: ATL rule translating Attributes to EAttributes and EReferences

```
ontology bpmoBusinessProcess

concept BusinessProcess
     hasName ofType (0 1) _string
     withExecution ofType Execution
     withCapability ofType Capability

concept CompensationProcess subConceptOf BusinessProcess
...

relation hasCompensation(ofType BusinessProcess, ofType CompensationProcess)
...
```

Listing 10: A relation in WSML

```
rule relation2eclass {
        from s : Wsml!RelationType
        to t : Ecore!EClass (
                name <- s.name.getLocalName(),
                eStructuralFeatures <- Set {s.parameters.parameter}
        )
         do {
                thisModule.paramCounter <- 0;
        }
}

rule parameter2reference {
        from s : Wsml!ParameterType
        to t : Ecore!EReference (
                eType <- thisModule.getConceptType(s.range.first()),
                name <- 'param'+thisModule.paramCounter.toString(),
                lowerBound <- 1,
                upperBound <- 1
        )
        do {
                thisModule.paramCounter <- thisModule.paramCounter + 1;
        }
}
```

Listing 11: ATL rule translating Relations and Parameters to EClasses and EReferences

```
ontology bpmoBusinessProcess
...
concept SelectType

instance XOR memberOf SelectType

instance OR memberOf SelectType

instance CASE memberOf SelectType

axiom allowed_select_types
     definedBy
          !- ?x memberOf SelectType and
          naf ((?x = XOR or ?x = OR or ?x=CASE)).
```

Listing 12: Example of an axiom representing an enumeration

Mapping enumerations Unlike OWL with its *oneOf* structure, WSMO doesn't explicitly define enumerations. Instead, axioms are used to describe a fixed set of instances that are valid for a class. Those classes with corresponding axioms, for which an example is shown in Listing 12, can then be translated to EEnums, and the instances to corresponding ELiterals.

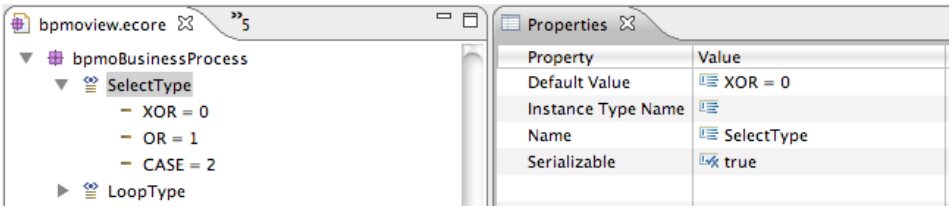


Figure 27: Mapping enumeration-like axioms to EEnums (Generic EMF Editor)

6.5 View Import

Beside the conceptual view definition, it also has to be populated in order to perform any further transformations. This can be done by using model transformations as well, using the WSMO meta-model as source meta-model and the conceptual view meta-model as a target meta-model, and importing the helper libraries.

As an example, for the concept shown in Listing 14, we want rules like the ones shown in 15. However, this is a repetitive task and may be tedious and error-prone, given the fact that a conceptual ontology may define a lot of concepts [8].

Fortunately, as ATL itself also is a model (i.e. there exists a corresponding ATL.ecore, from which ATL can be generated) we can write a transformation that

[8]Even for the rather small Sembiz BPMO ontologies, which together define 50 concepts

```
rule concept2eenum {
        from con : Wsml!ConceptType (
                con.isEnumCandidate()
        )
        to cla : Ecore!EEnum (
                name <- con.name.getLocalName(),
                eLiterals <- Set{con.getDirectInstances()}
        )
        do {
                thisModule.enumCounter <- 0;
        }
}

rule createEEnumLiteral {
        from s : Wsml!InstanceType
        to t : Ecore!EEnumLiteral (
                name <- s.name.getLocalName(),
                value <- thisModule.enumCounter
        )
        do {
                thisModule.enumCounter <- thisModule.enumCounter +1;
        }
}
```

Listing 13: ATL rules translating Concepts and Instances to EEnums and EEnum-Literals

```
ontology bpmoBusinessProcess
...
concept Execution
     executedBy ofType bpmo#BusinessRole
     interactsWith ofType bpmo#BusinessRole
```

Listing 14: Example of a WSML concept from which import rules will be created

actually *generates* the actual importing transformation. In fact, the rules shown in Listing 15 have been generated by the the rules shown in Listing 16. An overview of the transformation setup that generates the importing rules is shown in Figure 28.

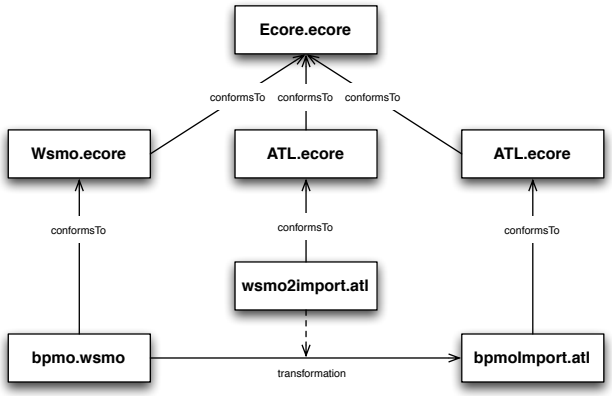


Figure 28: Transformation setup of the Create View Import activity

This resulting transformation's setup is then shown in Figure 29.

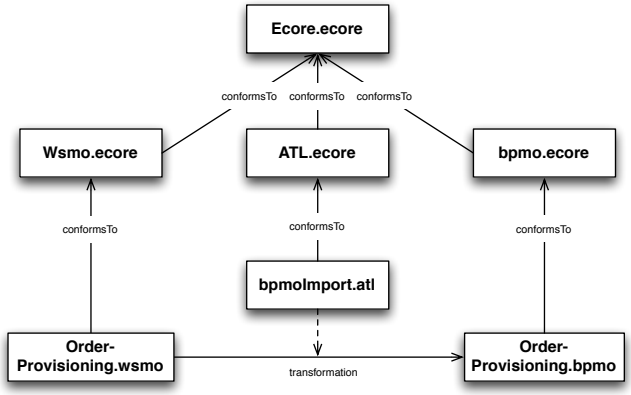


Figure 29: Transformation setup of the View Import activity

6.6 Manual View and Import Completion

Unfortunately, the created conceptual view provides not enough information to continue to work with. At this point, project-specific considerations and requirements have to be taken into account.

One reason is that it is necessary (and convenient) to maintain references to the source (i.e. the IRI of the corresponding ontology element). This is done by

```
module ...
...
uses wsmoUtil;

helper def : _bpmoBusinessProcess_Execution : String =
        'http://www.sembiz.org/bpmo/bpmoBusinessProcess#Execution';

helper def : _bpmoBusinessProcess_withExecution : String =
        'http://www.sembiz.org/bpmo/bpmoBusinessProcess#withExecution';

helper def : _bpmoBusinessProcess_withCapability : String =
        'http://www.sembiz.org/bpmo/bpmoBusinessProcess#withCapability';

rule instbpmoBusinessProcess_Execution2Execution {
        from s : Wsml!InstanceType (
                s.isMemberOf(thisModule._bpmoBusinessProcess_Execution) and
                s.hasSingleClassifier(
                'http://www.sembiz.org/bpmo/bpmoBusinessProcess#Execution'
                )
        ) to t : sembizgoal!Execution (
                executedBy <- thisModule.getInstanceTypes(
                        s.getAttributeTypeValue(
                                thisModule._bpmoBusinessProcess_executedBy)),
                interactsWith <- thisModule.getInstanceTypes(
                        s.getAttributeTypeValue(
                                thisModule._bpmoBusinessProcess_interactsWith))
        )
}
```

Listing 15: ATL import helpers and rules

```
rule conceptType2HelperConstant{
        from s : Wsml!ConceptType
        to chd:ATL!Helper
        (
                definition <- defi
        ), defi : ATL!Attribute (
                name <- '_'+s.getShortName(),
                type <- ATL!StringType,
                initExpression <- stringExp
        ), stringExp : ATL!StringExp(
                stringSymbol <- s.name
        ),  mr : ATL!MatchedRule (
                name <- 'inst'+s.getShortName()+'2'+s.name.getLocalName(),
                inPattern <- iP,
                outPattern <- oP,
        ...
        ),iP : ATL!InPattern (
                elements <- Set{iPe},
                filter <- conceptFilter
        ), oP: ATL!OutPattern(
                elements <- Set{oPe}
        ), iPe : ATL!StringSimpleInPatternElement(
                varName <- 's',
                typeName <- 'InstanceType',
                modelName <- thisModule.wsmlMM
        ), oPe: ATL!StringSimpleOutPatternElement(
                varName <- 't',
                typeName <- s.name.getLocalName(),
                modelName <- thisModule.outMM,
                bindings <- s.attribute
        ), conceptFilter : ATL!StringExp(
                stringSymbol <- 's.isMemberOf(thisModule._'+s.getShortName()
                 +') and s.hasSingleClassifier(\''+s.name+'\')'
        )
}

rule attributeType2HelperConstant{
        from s : Wsml!AttributeType(
                not s.isPrimitiveType() and not s.isEnumAttribute()
        )
        to bind : ATL!Binding (
                propertyName <- s.name.getLocalName(),
                value <- valueExpr
        ), valueExpr : ATL!StringExp(
                stringSymbol <- s.getValueExpression()
        )
}

-- attribute rules for primitive types and enumeration concept types
...
```

Listing 16: ATL helpers

making the top level classes derived from the ontology subclasses of the VbMF Core's extensible element.

Another reason is, that the generic conceptual view sees only the instances of the concepts it is derived from. This is however not enough for our particular case. As a (political) requirement, all modeling has to be done in ontologies. This means that ontologies are responsible for *data representation* which to some extent conflicts with the purpose of ontologies, focusing on *knowledge representation*, resulting in concepts occurring in instance ontologies and axioms that model relations between them. Axioms are then used to link instances to concepts in instance ontologies and also to represent business rules.

Depending on the project's context it may make sense to either adapt the conceptual view and the importing rules with reasonable effort to satisfy one's needs (as it will do for us) or to create additional views with their own concerns.

7 Create Transformation Rules

Now having shown how to bridge from the Ontology TS to the MDE TS, the next step for getting from ontologies to executable code is to bridge the MDE TS to the Workflow TS. This step may involve, depending on the meta-models describing the workflow TS, the creation of various model-to-model transformations and model-to-code/text transformations called *model transformations*.

As already mentioned above, setting up transformation rules for the transformation between metamodels is a non-trivial task requiring deep understanding of the problem domain and the metamodels of discourse. Having a supporting procedure when implementing the mappings should help getting it right faster. Figure 30 shows the Create Model Transformation subprocess of the overall integration process (which has been shown in Figure 15) which suggests several development steps for identifying model correspondences and creating model transformation rules.

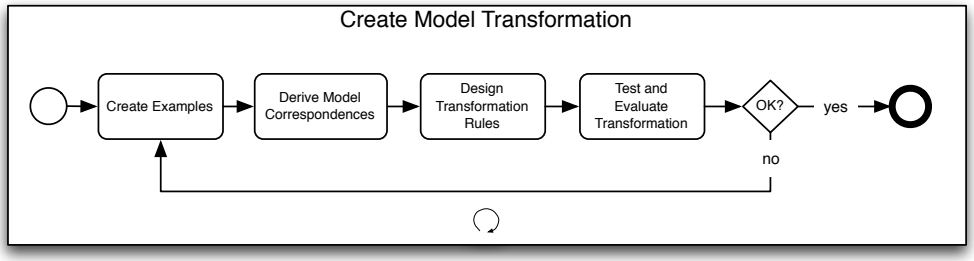


Figure 30: Create Transformation subprocess

In the beginning, the creation of examples describing the same problem is our starting point of choice (Section 7.2). Based on this knowledge, model correspondences can be derived. Section 7.3 is concerned with model correspondences an tries to give an overview of common problems and solutions when mapping a source to a target model. Having found and defined the model correspondences in the the meta-model layer, the next step is to transfer these findings into concrete transformation rules. When writing model-to-model transformations one should be aware of some common transformation patterns which provide proven solutions and support transformation design. Section 7.4 discusses the most common and, in our context, important ones and gives concrete examples. Finally, running and evaluating the transformation gives feedback about the current state of completeness, this topic is shortly covered in Section 7.4.7.

The steps above are applied in an iterative manner in order to divide and conquer the problem. It is a good practice to think of the most tricky problems first and to implement a prove of concept solution. Enhance the complexity of your test use cases and run the transformation as often as necessary until a satisfying state is reached.

7.1 Related Work

Actually a lot of work and research is going on in the field of MDE. As model transformations are a core part of almost any MDE approach, a lot of knowledge concerning the creation and design of transformation rules was created in recent years. An approach supporting the creation of model transformation rules is the Model Transformations By-Example (MTBE) approach shortly introduced in Section 7.1.2. The substantial Modelware [7] project tries to establish MDE and MDD for the development of industry-level systems including efforts in modeling theory, technologies and tools.

7.1.1 Modelware

Modelware is an ambitious project that aims at the large-scale deployment of MDD. This includes a lot of concerns like modeling technologies, engineering processes and methodologies, the development of tools and other things as shown by Figure 31 giving an overview of the Modelware work packages. Especially the projects first work package (WP1) is concerned with modeling technologies in general and tries to improve state of the art MDD theory and technologies. One concrete outcome of this work was the creation of a pattern catalogue for the design of transformation rules. Section 7.4 examines the (in our context most relevant) ones and gives appropriate examples.

7.1.2 Model Transformations By-Example

The MTBE approach is concerned with providing a more user friendly way for building model transformation rules on semantically corresponding meta-models. Their entry is the creation of (basic) transformation rules (ATL) based on the mapping between concrete examples of both modeling languages (EMF), showing the same problem domain. In their work done in [64] Wimmer et. al. investigate some caveats and general relations when creating mappings between two models and present an orthogonal and extending approach to existing model transformation approaches which allows user friendly support for the creation of transformation rules. Figure 32 gives an overview of the suggested framework.

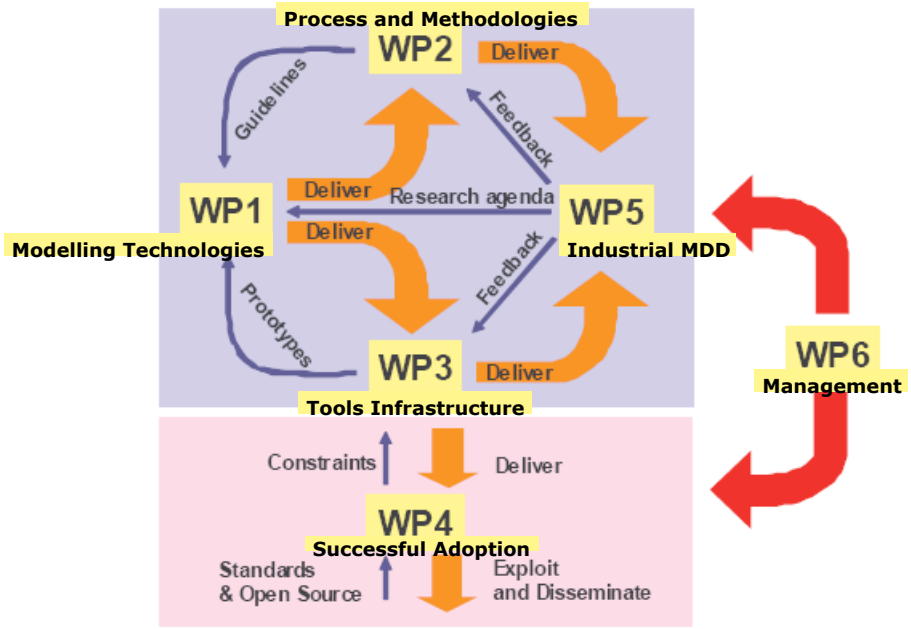


Figure 31: Overview of the Modelware work packages (source: [8])

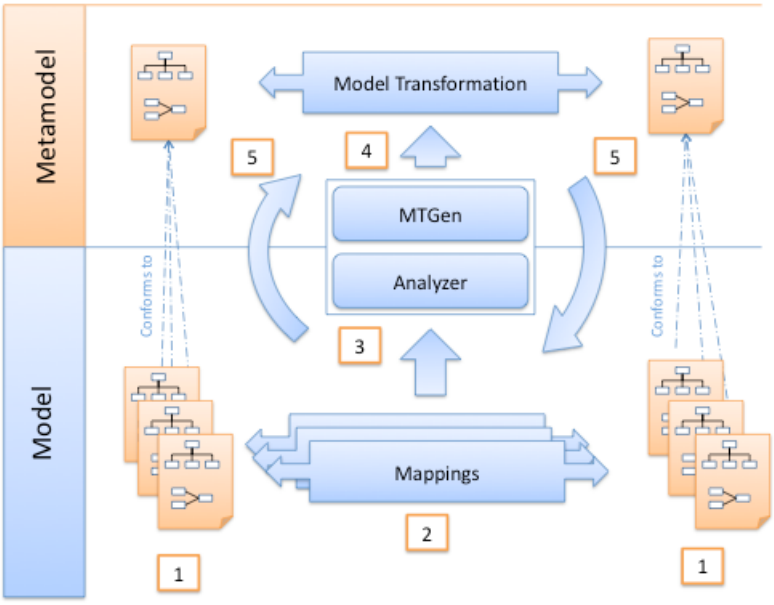


Figure 32: Overview of the Model Transformations By-Example framework (MTBE, source: [43])

In general, an intermediate model is used to represent the correlations between abstract syntax (as given by the meta-model) and concrete syntax (as given by the concrete example) as well as correlations between source and target model. These models act as input for a model transformation rule generation.

7.2 Create Examples

In a first step the creation of a sound example in the form of source and target models, describing the same problem, supports the developer on (a) understanding the different meta-models and (b) provides a definition of the desired transformation outcome. Setting up comprehensive examples showing the problem domain is our starting point of choice. Based on the existing examples acquiring the knowledge about semantic correspondences between the examples becomes a much more easier task and the analysis can happen on the model layer instead of on the meta-models solely. Additionally, by creating examples, implicit and hidden relations are discovered. Figure 33 shows the Hanival use case in a visualization in the WSMO Studio which attended our transformation development all the time.

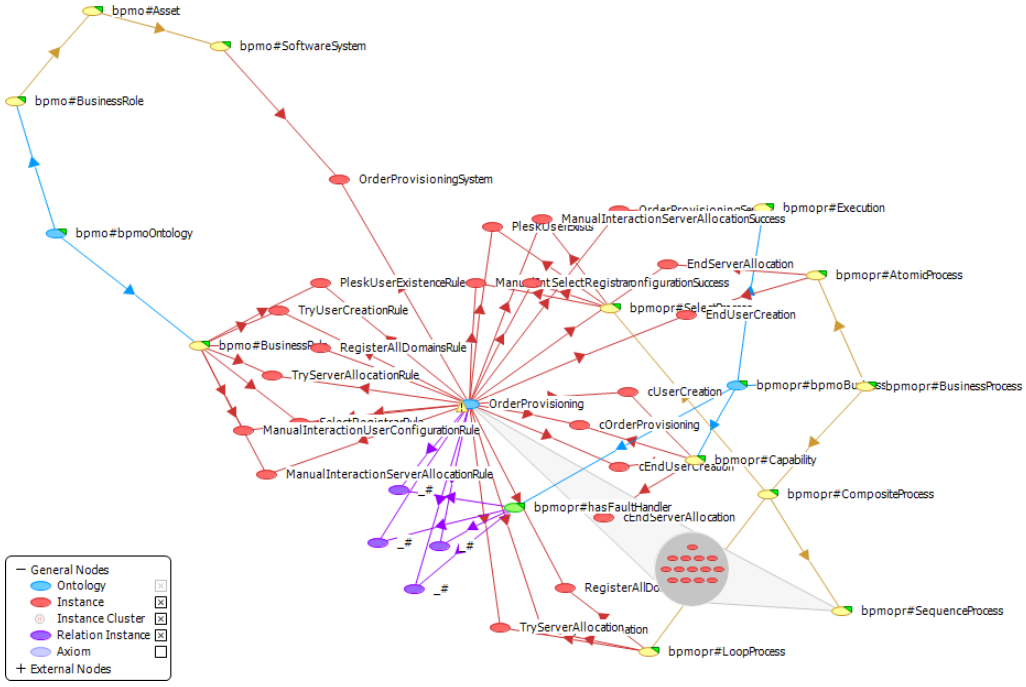


Figure 33: The OrderProvisioning process ontology (WSMT Ontology Editor)

On the target side we started to rebuild relevant and uncertain parts of the use case per hand using the VbMF Eclipse plugins. Later on, when the first transformation rules are implemented and executed, one can use the newly created parts

in order to extend them to appropriate new target examples. Figure 34 shows the BPELTransactionView of the OrderProvisioning process.

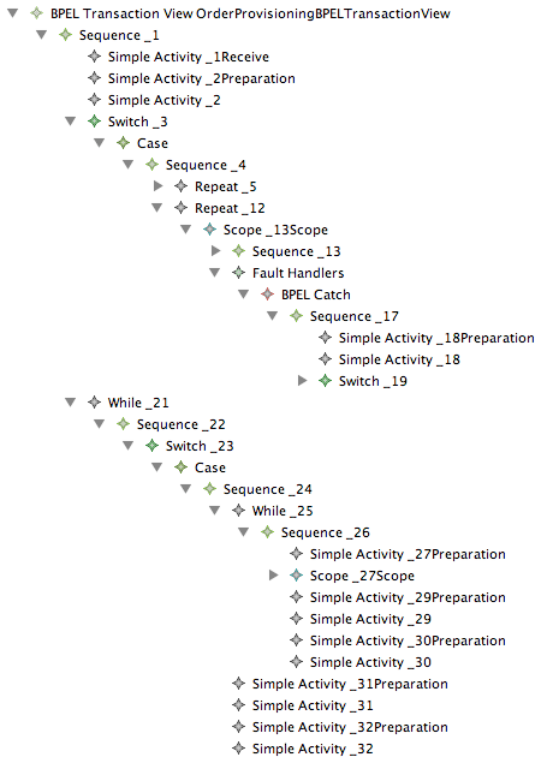


Figure 34: The OrderProvisioning BPEL Transaction View (excerpt, Generated EMF Editor)

7.3 Deriving Model Correspondences

Now that the creation of all required dedicated views for the ontologies is done, all the necessary infrastructure exists to go to the next step where equivalences or correspondences between the views are identified.

A developer primarily must be aware of a views concern and the purpose of its elements. In addition to a views meta-model concrete examples are pretty helpful to identify all its characteristics. The rest of this section is concerned with a first analysis on source and target models and shows how to derive correspondences and identify tricky parts. Finally the relevant elements of the source-model are related to elements of the specific target-model such that a very coarse grained initial mapping as well a more fine grained mapping is constructed.

7.3.1 Classification of source elements

In order to specify the correspondences of source and target elements an initial classification of the different elements supports transformation design as afterwards relevant, hidden and missing elements are identified.

Irrelevant First of all, elements in the source model exist which are for a different or non-existent purpose than provided by the target views. These elements are said to be irrelevant to the actual transformation and need no further attention from the transformation designer for his actual task. All other elements left are then at least relevant for the transformation. Figure 35 shows the irrelevant elements of the BPMO source-model.

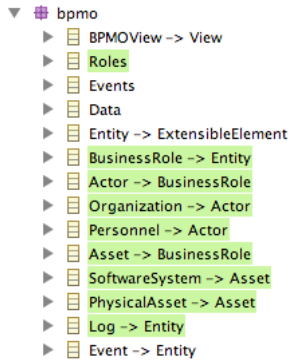


Figure 35: Example for irrelevant elements in the BPMOView (Generic EMF editor)

The identification of irrelevant elements in the BPMO source models was not always straight forward. Especially the BusinessRole was pretty misleading during the transformation design: A BusinessRole represents a real-world entity that executes and interacts with business processes. Its subclasses Actor and Asset represent organizational and technical roles so that BusinessRole can be used to model the entire organizational structure.

Having in mind the CollaborationView's element Role, one would possibly suggest a correspondence between those elements. Actually, during the analysis of the use cases we found that theses elements differ in their semantic and that the BPMO BusinessRole is irrelevant to our transformation: the VbMF models *executional roles* (as used by BPEL) while the BPMO models *organizational hierarchy and relations*. The relevant information for a complete transformation actually was *hidden* in the Execution element.

Hidden Concept Sometimes the information required to generate some specific object is not explicitly given. There may exist hidden concepts, the phenomenon of concept hiding, where required concepts are hidden in attributes and association ends. Consider situations where out of an attribute or a relation in the source model an element in target model has to be generated.

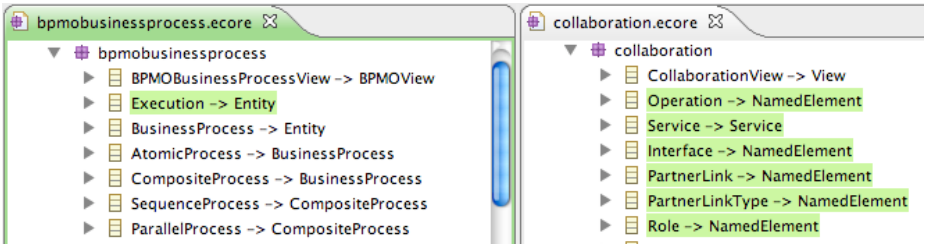


Figure 36: Example for hidden elements in the VbMF CollaborationView (Generic EMF editor)

As mentioned above, this occurred with the Execution element when deriving the CollaborationView's Role. For a complete BPEL and WSDL code generation the Role is important as it links several other elements. In our approach, for the relation end withExecution of the BusinessProcess, a Role in the CollaborationView is created.

Missing From time to time, parts of the target cannot be filled out of information available in the source. Here some, for a complete transformation, the designer is required to bypass some information. This ideally is done by additional models from the source side or by a finalizing human task.

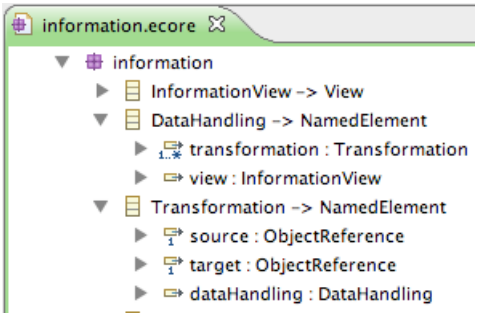


Figure 37: Example of missing information for the VbMF InformationView (Generic EMF editor)

For the InformationView we had no information for building DataHandling elements respectively Assign elements in the source which are shown in Figure 37. In our concrete case, setting up these elements is part of a human rework task. If you are confronted with missing information you might find a solution in by implementing the *Incomplete* pattern in Section 7.4.1.

7.3.2 Identification of element correspondences

When defining mapping rules, mapping correspondences between the metamodels occur in various granularities which in turn influence object generation in the target model. The following describes the most common relations between model objects.

1:1 Mapping One-To-One (1:1) mappings are the most simple and straightforward, but also fundamental cases. Here for one instance in the source the exactly one element in the the target is created.

Mostly this means just a mapping of the corresponding attributes. Figure 38 shows this by the mapping of BPMO Message events and AtomicProcess to the VbMF CollaborationView.

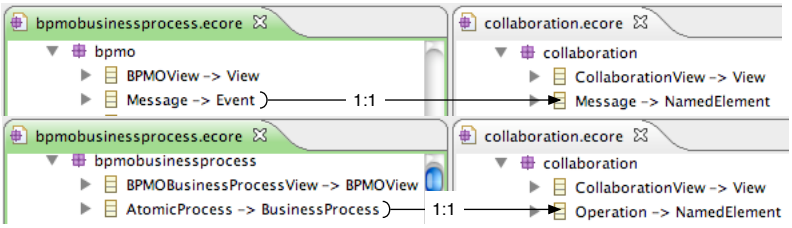


Figure 38: 1:1 mappings between BPMO views and the VbMF Collaboration view (Generic EMF Editor)

Listing 17 shows a possible mapping between those elements. Here, for one Message exactly one corresponding Message is created and its name is set. For the AtomicProcess a corresponding Operation is created and the attributes name, in, out and fault are set by passing attributes of the source element to other transformation rules.

```
create collaboration::Message this createMessage(bpmo::Message m):
        this.setName(m.identifier.name)
;

create collab::Operation this createOperation(bpmopr::BusinessProcess bp):
        this.setName(bp.identifier.name) ->
        this.in.addAll(bp.capability.preconditionMessages().createMessage()) ->
        this.out.addAll(bp.capability.postconditionMessages().createMessage())
;
```

Listing 17: Example of a simple 1:1 correspondence

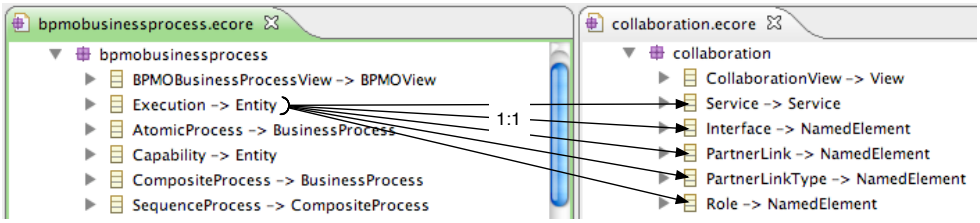


Figure 39: Example of a source class having multiple 1:1 mappings (Generic EMF editor)

```
createExecutionElements(List[bpmopr::Execution] el, coll::CollaborationView v):
        v.service.addAll(el.createService()) ->
        v.interface.addAll(el.createInterface()) ->
        v.partnerLink.addAll(el.createPartnerLink()) ->
        v.partnerLinkType.addAll(el.createPartnerLinkType()) ->
        v.role.addAll(el.createRole())
;
```

Listing 18: Example of a rule dispatching multiple 1:1 relations

A similar case is given by the mapping of Execution to elements of the CollaborationView shown by Figure 39.

Here for one *Execution* element in the source elements for the classes Service, Interface, PartnerLink, PartnerLinkType and Role need to be created in the target. Listing 18 shows a transformation rule that bundles the creation of all the elements for all the instances of Execution and distributes the executions to various creation rules (similar to those in Listing 17).

1:N Mapping One-To-Many (1:N) mappings occur when for one instance in the source model multiple equal elements have to be created in the target again and again.

An example is the transformation of BusinessProcess into Activitiy where multiple times the same Activity needs to be created in the target. Figure 40 shows the correspondences of BusinessProcesses and Activities. In BPMO *BusinessProcesse*s always *occur once* and are afterwards referenced by other elements. Within the VbMF *ControlflowView* acitvities are held in a containment hierarchy and may occur multiple times.

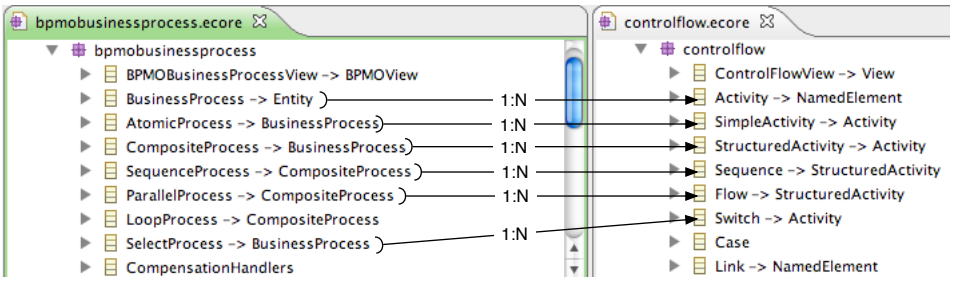


Figure 40: Example of a 1:N mapping between BusinessProcess and Activity (Generic EMF editor)

Therefor the once transformed BusinessProcces is not required to be created again and again but can be cached somehow (see the Path Reminder Pattern 7.4.5). Additionally a simple repetition may sometimes not be sufficient, consider a unique constraint on the target side which may require a special modification of attribute values (see the Precomputed Pattern in Section 7.4).

An other case is when for one object in the source one *or* another object in the target model is created multiple times. The selection of the actual element to be created happens conditionally, based on attribute values, relations or the inheritance structure of source elements.

```
createInteraction(bpmopr::BusinessProcess bp):
  (bp.withCapability.precondition().contains(bpmopr::StartMessage)) ?
    createReceive() :
      {
      (bp.withCapability.postcondition().contains(bpmopr::EndMessage)) ?
          createReply() : createInvoke()
      }
;
```

Listing 19: Example of an 1:N correspondence depending on a condition

Listing 19 gives an example of a transformation rule where, based on the type of event in the pre- and postcondition, different elements in the VbMF CollaborationView are created. In cases where the creation depends on the inheritance of the source element the Metamodel Polymorphic Rules Pattern in Section 7.4.6 may be of interest for the rule design.

N:1 Mapping Many-To-One (N:1) mappings are akin to 1:N mappings. Differently, just one specific object in the target model has to be created but many potential elements in the source model are available. Such a case is given with the relation between AtomicProcess and Execution. Here the AtomicProcess points to multiple Executions as shown in Figure 41 which brings up the need for some decision support.

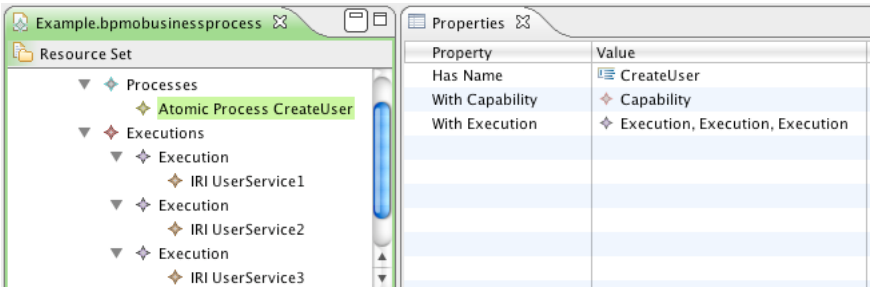


Figure 41: Example of a N:1 correspondance (Generic EMF editor)

A possible solution to such ambiguous situations is shown in Section 7.4 when introducing the Incomplete pattern.

7.3.3 Conclusion

At least, initially thinking of model correspondences helps to split the standard and straight-forward cases from the more complex transformation tasks. Additionally the investigated correspondences could be documented in a table as it is done in the example shown by Figures 42 and 43.

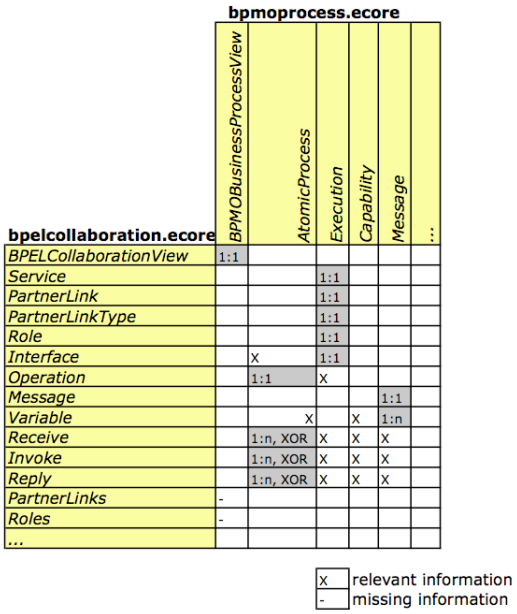


Figure 42: Example of model correspondences on the class level

One may create such mapping tables based on different view-points and levels on granularity. In the first example the mappings occur just on the class level. This provides a rough overview of the element mappings and a markup of relevant related classes.

Figure 43 provides a more detailed view by showing the class and the attribute level of the correspondences.

The analysis of model correspondences and the creation of mapping tables supports rule design. They a) provide a documentation of the mapping, b) make the standard cases visible and c) help finding the special cases. Additionally, one could implement such a mapping table in form of a meta-model that could be used to generate a good portion of the required transformation rules.

7.4 Design Transformation Rules

In this section, we will discuss the design of transformation rules, that perform a model-to-model transformation like the one shown in Figure 44.

As this is a complex task, we make use of common design patterns describing best practices and solutions to reoccurring situations developers face when writing model transformation rules.

A frequently cited description of patterns was given by Christopher Alexander [19] who wrote:

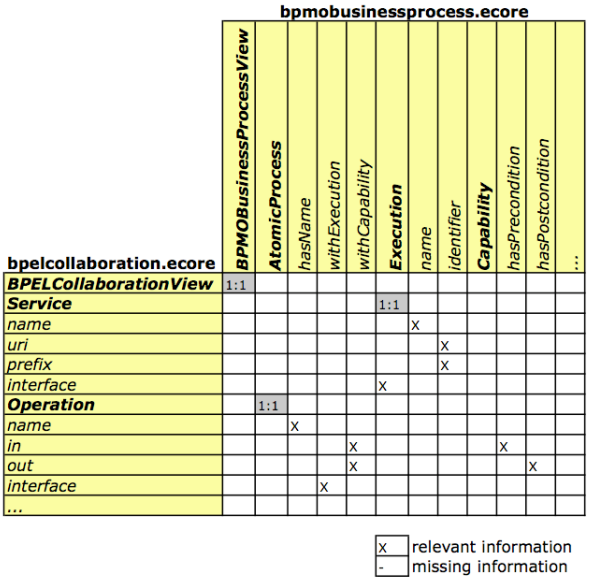


Figure 43: Example model correspondences on the class and attribute level

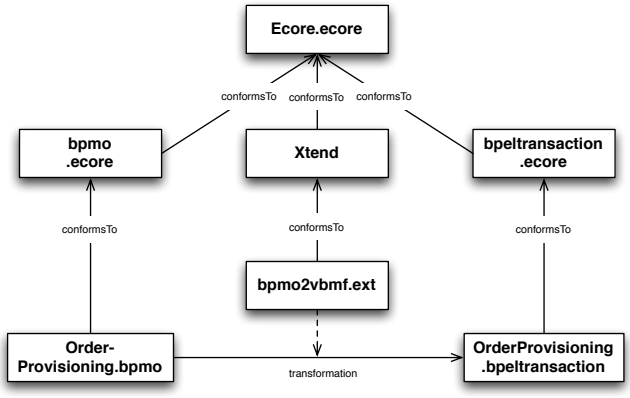


Figure 44: BPMO View to BPEL Transaction View transformation setup

> ”Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice”

The following paragraphs present the, in our context, most relevant design patterns for the development of transformation rules which are part of the pattern catalogue presented in [21] (a deliverable of the afore mentioned Modelware project). Developers of transformation rules should be aware of these patterns in order to judge whether a specific pattern is appropriate for their transformation task or not.

The following descriptions are structured as follows: An introducing problem description and motivation outlines the concern of the pattern. Then possible solutions along with examples are given to finally summarize the consequences when applying the pattern.

7.4.1 Incomplete

Synopsis Sometimes, information given by the source model is insufficient or non-deterministic and requires human interaction or additional input.

Motivation When the information in the source model poses an undecidable situation, in other words it requires a non deterministic decision in order to carry on, one may like to have a way to influence the transformation.

Solution A simple solution to a non-decideable situation would be to encode a standard choice (e.g. if there are many elements to choose, take the first one), or, for more complex situations, make use of a heuristic function or reasoning engines. Another way to address non deterministic mappings would be to interrupt the transformation and make the decision a human task. Let's consider an AtomicProcess pointing to multiple Executions. Without any concrete advise there is no hint which Execution to use and the selection has to base on some kind of heuristics (e.g. choose the first one).

One way to achieve the integration of human advises is to build up a supporting metamodel representing an incomplete guidance like in the simple example given by Figure 45. Here the transformation is done in at least two cycles. In the first round, the transformation initially builds up the supporting model storing for instance multiple choices of Executions, this is illustrated in Figure 46.

The transformation of the second round takes the supporting model as an additional input which then can include hints for incomplete situations.

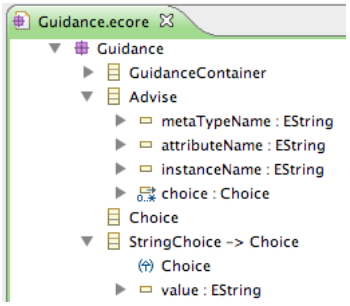


Figure 45: Simple Guidance Metamodel (Generic EMF editor)

```
guidance::Advise createExecutionAdvice(bpmopr::BusinessProcess p):
        createAdvice(p, p.withExecution)
;

create guidance::Advise this createAdvice
 (bpmopr::BusinessProcess p, List[bpmopr::Execution] el):
        setMetaTypeName(p.metaType.name) ->
        setAttributeName(''withExecution'') ->
        choice.addAll(el.createChoice())
;

create guidance::StringChoice this createChoice
 (bpmopr::Execution e):
        setValue(e.name)
;
```

Listing 20: Initializing the guidance model in the first transformation.

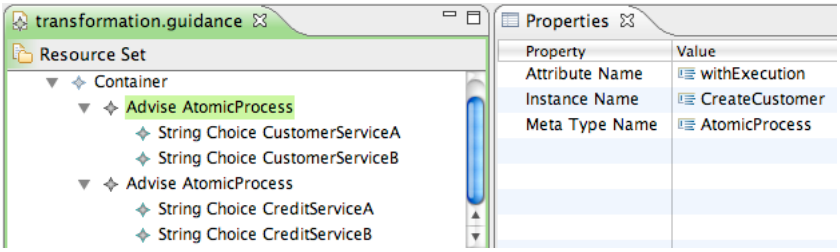


Figure 46: Guidance Example (Generated EMF editor)

```
collaboration::Service createService
 (guidance::GuidanceContainer c, bpmopr::BusinessProcess p):
   createService(
        p.withExecution.selectFirst(
            e | e.name.matches(c.getAdvice(p.name))
        )
   )
;
```

Listing 21: Using the guidance model in transformation rules.

Consequences Simple solutions like heuristics can be fast and easily created for simple and are a good choice in cases where they are sufficient. Using a guidance model this means that the transformation operator's knowledge can be stored, edited and reused. The supporting guidance model becomes an additional artifact of the transformation.

7.4.2 Parameters

Synopsis A transformation requires static information influencing the transformation outcome.

Motivation Model transformations often require static information to complete the target model, which in many times is just hardcoded in the transformation rules or are available via helper functions (which are hard-coded in the transformation as well).

Solution Parameters are elements of the transformation that are not available in the source model. Hard-coding them into the transformations rules is bad praxis because their values are only accessible through editing the transformation rules.

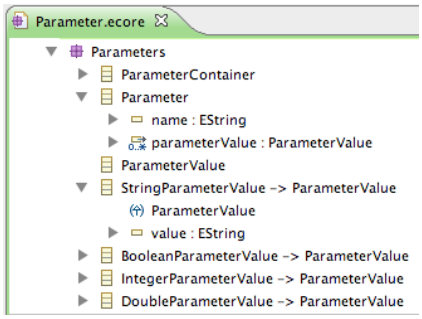


Figure 47: Parameter Metamodel (Generic EMF editor)

A solution which provides the outsourcing of the parameter settings is the creation of another supporting metamodel: a parameter metamodel. Like the guidance model, the parameter model can be taken as an input of the transformation and all possible settings can be done in a dedicated model without touching the actual transformation rules. Figure 47 shows an example of a simple generic parameters metamodel in Ecore.

As shown by Figure 48, parameters could be easily managed via the parameter model without touching the actual transformation code. Listing 22 shows an example of a transformation rule accessing a parameter via a helper function.

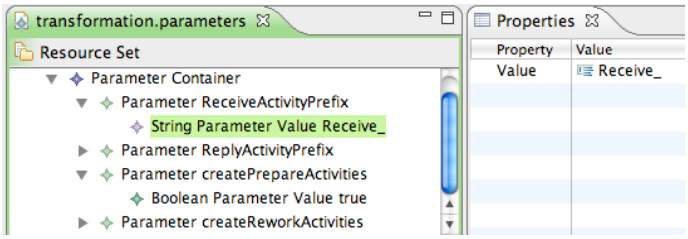


Figure 48: Parameter Example (Generated EMF editor)

```
String getReceivePrefix():
 GLOBALVAR params
    .selectFirst(p | p.name.matches(''ReceiveActivityPrefix''))
;

create bpelcollaboration::Receive this createReceive
 (bpmopr::BusinessProcess p):
      this.setName(getReceivePrefix() + p.name)
;
```

Listing 22: Using parameters in transformation rules.

Listing 28 in Section 8 shows how a parameter model can be integrated in the transformation.

Consequences Using a parameter model enables comfortable settings and can be done in a dedicated model without changing the transformation code. Further this increases reuse and allows an easy switching between different configurations.

7.4.3 Precomputed

Synopsis Sometimes transformations require some kind of pre-computation influencing or altering the transformation.

Motivation When doing a calculation, maintaining of a counter or even gather frequent required elements, a function at hand for such a particular task would be handy.

Solution Typically this is solved using helper functions. They are computed many times during a transformation and encapsulate general as well as very specific recurring functionality. Listing 23 shows a Java method supporting a transformation task.

Here, the occurrence of similar processes during the process flow is counted. Every time a process occurs in the flow it is added (addProcess(Object o)) and

```
public final static Integer addProcess(Object o) {
        count ++;
        if(map.containsKey(o)){
                List<Integer> li = map.get(o);
                li.add(count);
        }else{
                List<Integer> li = new ArrayList<Integer>();
                li.add(count);
                map.put(o,li);
        }
        return count;
}
```

Listing 23: Precomputed Example

```
Void nameset(core::NamedElement e, bpmopr::BusinessProcess p) :
        e.setName("_"+addProcess(p))
;
```

Listing 24: Usage of the pre-computed function.

the actual counter is returned. This function is used when creating Activities a multiple times which is required because of the inflicted unique name constraint posed by BPEL (Listing 24).

Consequences Using helper functions is good praxis. It helps to modularize the code and to keep it in good order which makes the transformation rules easier to understand and maintain. Large transformation and complex computations however may take their time.

7.4.4 Model Navigator

Synopsis Often, when transforming a model element, another element without direct relation to the actual element is required.

Motivation In order to resolve such indirect relations often an uncomely detour, for instance over the parent elements, is required which messes up the code and increases code complexity.

Solution Instead on writing a complex navigation expression every time required one should outsource such navigations into helper functions which solve the navigation task as shown by Listing 25.

```
getAtomicProcesses(core::NamedElement n):
        n.eRootContainer.eAllContents.typeSelect(bpmopr::AtomicProcess)
;
```

Listing 25: Model Navigator Example

```
create controlflow::SimpleActivity this createSimpleActivity2
 (bpmopr::AtomicProcess p):
...
;

cached getMessages(core::NamedElement n):
...
;
```

Listing 26: Path Reminder Example

Consequences As already said, the use of helper functions contribute to reusability and less complex code.

7.4.5 Path Reminder

Synopsis Usually during the transformation of the source model single elements are required many times for different purposes.

Motivation This is especially the case when an element is referenced from various other elements. The transformations of this element is only required the first time the transformation passes it, in all other cases a repeating transformation should be avoided.

Solution A possible solution to this problem is to maintain a cache storing already created elements. Some transformation engines, like Xtend, come with special creation or cached rules (Listing 26) which are able to cache their return values taking their input parameter as the key value.

Consequences When an element is required during the transformation, first the cache is asked for the element which has typically much faster response times and avoids unnecessary computation.

7.4.6 Metamodel Polymorphic Rules

Synopsis A lot of transformations need to deal with objects of the same superclass but belonging to a specific subclass which have to be treated differently

```
controlflow::Activity createActivity
 (bpmopr::BusinessProcess p):
 switch{
  case (p.metaType.name.matches("bpmopr::AtomicProcess")):
   createActivity((bpmopr::AtomicProcess)p)
  case (p.metaType.name.matches("bpmopr::SequenceProcess")):
   createActivity((bpmopr::SequenceProcess)p)
  case (p.metaType.name.matches("bpmopr::ParallelProcess")):
   createActivity((bpmopr::ParallelProcess)p)
  ...
  default: {info("No rule defined for type " + p.metaType.name )-> Void}
 }
;

create controlflow::Sequence this createActivity
 (bpmopr::SequenceProcess p):
        setName(p.name) ->
        activity.addAll(p.subprocess.createActivity())
;

create controlflow::Activity this createActivity
 (bpmopr::AtomicProcess p):
        setName(p.name)
;
```

Listing 27: A redirecting rule to achieve polymorphic behavior.

during transformation.

Motivation Having rules sharing a common name but specializing on the specific metamodel classes frees the developer of making the choice of the correct transformation rule.

Solution The name of the rule acts as a common entry point, the rule selection is based on the meta-class of the parameter. Some transformation engines allow polymorphic rules, but others like Extend do not. Here one could work around implementing a redirecting rule as shown by Listing 27.

Consequences Using polymorphic rules is a good praxis which again reduces the code complexity. However there may occur runtime errors when the rule for a specific type is missing, or, is the case of a redirecting rule, explicit type checking is required.

7.4.7 Test and Evaluation

Based on the initially defined examples and the target output of the transformation, frequent testing leads the way to correctly transform source to target. Consider setting up a run task for your transformation early in the transformation design. This

way the development can pass several cycles with a steadily increasing complexity and frequent testing.

Beside the comparison to example models/code, another way to verify the outcome is to open the created artifacts with common editors. Figure 49 shows how the use case validated by the Netbeans BPEL plugin. Going a step further test and

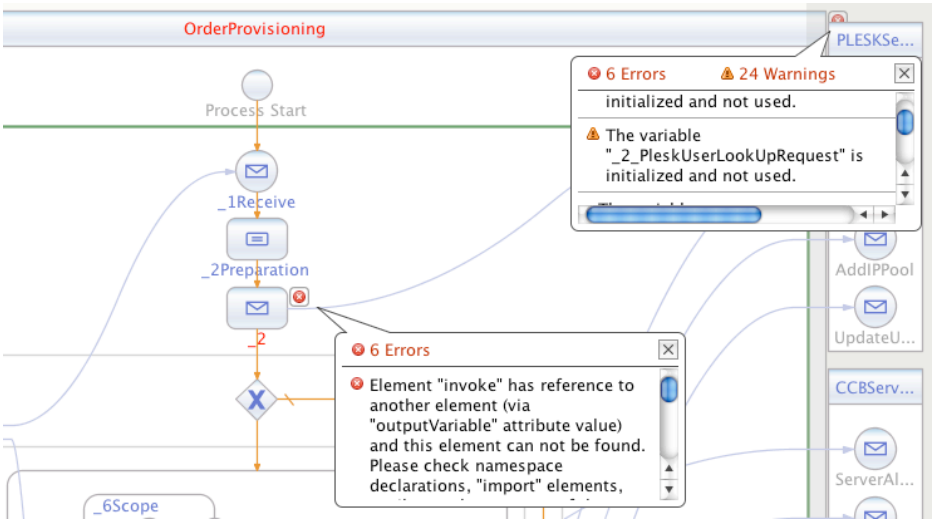


Figure 49: OrderProvisioning BPEL validation (Netbeans BPEL Editor)

evaluation can additionally include deployment and execution. Using a unit testing framework for such a task, as described in [42], is recommended. The BPEL unit testing framework implementation provides xUnit like white- and black-box testing for BPEL processes and allows a mocking of all participating partner tracks of the process.

8 Integration and Deployment

By now we have illustrated the building blocks of the integration task, i.e. View creation, View import, and model transformations. All these steps require input models (sometimes in different formats, e.g. WSML, XML, XMI/Ecore), transformation models (e.g. ATL, oAW Xtend, XSLT), and produce output models (again possibly in different formats).
However, a single step for its own is on a very fine-grained, technical level with little business value, so it is necessary to combine several steps to provide a valuable BUSINESS-DRIVEN SERVICE [34], providing an interface that hides implementation details. Because transformation details may change over time (as did ours during our work), it is necessary to be able to adapt the single steps with little effort.

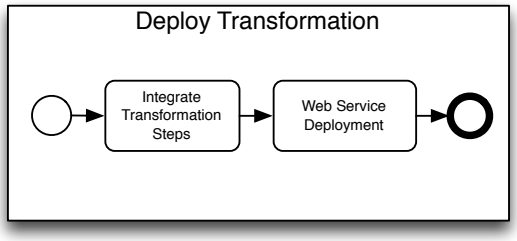


Figure 50: Deploy subprocess

In Section 8.1 we describe how the single transformation steps can be integrated, and in Section 8.2 we show how this combined transformation is deployed as a Web service (cp. Figure 50).

8.1 Integration of single Model Transformation Steps

In order to combine the single components to a complete transformation, it is necessary to orchestrate them. Literature research and practical work have shown that it is reasonable and necessary to (1) be able to easily exchange components and (2) support multiple transformation technologies. The openArchitectureWare (oAW) generator MDD/MDA framework provides a solution to both problems: a workflow engine solving (1) and extensible workflow components that can be used to create adapters for different technologies, thus solving (2).

8.1.1 Transformation Workflow

The oAW workflow engine executes a workflow file, like the one shown in Listing 28. A workflow is defined in an XML file and consists of a sequence of workflow components. A workflow component is basically a Java class implementing a particular interface [9], its qualified class name has to be specified in the component's "class" attribute in the workflow file. Parameters can be passed to components using XML elements, the workflow engine uses reflection to set the values on instantiated objects. In- and outputs can be passed from one component to another using so-called "slots", which are actually entries in a workflow-global map. Another possibility to exchange in- and outputs is using files. (c.f. [14])

Example Workflow In our example in Listing 28, the first step is to import Wsml files to the BPMO View. The first component there is an adapter for a Java based importing tool, that parses Wsml files using wsmo4j and writes them to an XML Document. After that an XSLT transformation adds an XMI header to the XML document, making it possible to load it using an XMI reader. Finally an ATL transformation does the actual import.
In the second step, the model transformation from the BPMO to the BPELTransaction View is performed using the builtin Xtend Component. For this component all the involved metamodels have to be declared. The "invoke" parameter specifies the transformation rule that actually performs the transformation for the "bpmoView".
The third step is a serialization component that serializes the slot content to XMI again. These latter two steps are then repeated for the transformation from the BPMO to the BPMOCollaboration and -Information View, resp.

```
<workflow>
<property file="transformation/workflow/bmpo2vbmf.properties" />
  <!-- 1) BPMO View import from WSML (using adapters) -->
  <component class="net.phir.oaw.adapter.wsml.Wsml2XmlTransformerComponent">
    <inputWsmlFile value="${importDir}/${bpmoWsmlFiles}"/>
    <outputXmlFile value="${xmlFile}"/>
  </component>

  <component class="net.phir.oaw.adapter.xsl.XslTransformerComponent">
    <xslFile value="${transformDir}/xsl/wsmlXmiHeader.xsl"/>
    <inputXmlFile value="${xmlFile}"/>
    <outputXmlFile value="${xmiFile}"/>
  </component>

  <component id="xmiParser" class="org.openarchitectureware.emf.XmiReader">
    <modelFile value="${xmiFile}"/>
    <metaModelFile value="${modelDir}/${Wsml}"/>
    <outputSlot value="inWsml"/>
```

[9] Usually `org.openarchitectureware.workflow.WorkflowComponentWithID`

```
  </component>

  <component id="wsmo2bpmoView"
    class="net.phir.oaw.adapter.atl.AdvancedATLTransformerComponent">
    <workDir value="${tempDir}"/>
    <metamodelPath value="Wsml -> ${modelDir}/${Wsml}"/>
    <metamodelPath value="bpmobusinessprocess ->
      ${modelDir}/${bpmobusinessprocessmm}"/>
    <inputModelToMetamodelMap value="inWsml -> Wsml"/>
    <outputModelToMetamodelMap value="out -> bpmobusinessprocess"/>
    <modelPath value="inWsml -> slot:inWsml"/>
    <modelPath value="out -> slot:bpmoView"/>
     <library value="wsmoUtil -> ${transformDir}/m2m/wsmoUtil.asm"/>
    <asmFile value="${transformDir}/m2m/wsmo2bpmoView.asm"/>
  </component>

<!-- 2) BPMO to BPEL Transaction View M2M component -->
  <component id="xmiParser" class="org.openarchitectureware.emf.XmiReader">
    ...
    <outputSlot value="parameters" />
  </component>

  <component class="oaw.xtend.XtendComponent">
    <metaModel class="oaw.type.emf.EmfMetaModel">
      <metaModelFile value="${bpmobusinessprocessmm}" />
    </metaModel>
     <metaModel class="oaw.type.emf.EmfMetaModel">
      <metaModelFile value="${parametermm}" />
    </metaModel>
    <metaModel class="oaw.type.emf.EmfMetaModel">
      <metaModelFile value="${bpeltransactionmm}" />
    </metaModel>
    <!-- Analog for other metamodels ... -->
    <globalVarDef name="params" value="parameters" />
     <globalVarDef name="cv" value="bpmoView" />
    <invoke value=
    "transformation::m2m::ext::bpmo2bpeltransaction::transform(bpmoView)" />
    <outputSlot value="bpeltransaction" />
  </component>

   <!-- 3) BPEL Transaction View XMI serialization -->
  <component id="xmiWriter" class="org.openarchitectureware.emf.XmiWriter">
    <inputSlot value="bpeltransaction" />
    <modelFile value="${bpeltransaction}" />
  </component>

  <!-- Analog for BPMO to BPELCollaboration and -Information -->
        ...
</workflow>
```

Listing 28: The bmpo2vbmf oAW Workflow

8.1.2 Workflow Component Adapters

Adapters can be used to invoke external components as part of the workflow, such as ATL or XSLT transformations. For our work we enhanced the ATL adapter that is available from the SCM repository of openArchitectureWare, and created adapters for performing XSLT transformations as well as WSML to XML and

vice versa transformations.
For the WSML to XML transformation, an excerpt of the implementation is shown in Listing 29. An adapter has to directly or indirectly implement the WorkflowComponent interface and usually provides some setter methods, that can be used in the workflow for parametrizing the component: The the parameter's name in the workflow has to correspond to one of the according class's setter properties in Java Beans style, i.e. `<outputXmlFile value="..."/>` and `public void setOutputXmlFile(String value)`, resp.
The AbstractWorkflowComponent provides default implementations for all but the invoke() method, so that only this one has to be implemented. We use the wsmo4j Parser and Serializer, to actually perform the transformation.

8.2 Transformation Deployment

The final step in deployment of the BPMO- ontology to VbMF transformation workflow is the integration with the SemBiz architecture, especially with the Deployment Process that it is part of.

8.2.1 Web Service Deployment

The functionality of the BPMO to VbMF transformation has been encapsulated as a Web Service. The Web Service is responsible for the exposition of the transformation workflow, it receives a couple of WSML files and caches them in a temporary directory. It configures the oAW workflow (i.e. overloads workflow properties), executes it and returns the serialized result.

8.2.2 Sembiz Process Deployment process

The Deployment process is orchestrating the still quite technical, encapsulated services, like the transformation (of BPMO ontologies to VbMF Views), the generation (of VbMF Views to BPEL/WSDL), and the deployment (of the generated BPEL/WSDL) and wrapping them as activities in a process of real business value: A (semi-)automatic transformation and deployment of semantic business processes (cp. WRAP SERVICE AS ACTIVITY in [33]). Listing 30 shows an excerpt of this process.

```
public class Wsml2XmlTransformerComponent extends AbstractWorkflowComponent{

  private Parser wsmlParser;
  private Serializer xmlSerializer;

  private String outputXmlFile;

  public void setOutputXmlFile(String outputFile) {
    this.outputXmlFile = outputFile;
  }

  ...

  public void invoke(WorkflowContext arg0, ProgressMonitor arg1,
      Issues arg2) {
    try {
      List<TopEntity[]> ontologies = parseWsmlFiles(getFiles(inputWsmlFile));
      writeToXml(ontologies, getFile(outputXmlFile));
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  private List<TopEntity[]> parseWSMLFiles(File[] fWsml) throws IOException,
      ParserException, InvalidModelException{
    List<TopEntity[]> ontologies = new ArrayList<TopEntity[]>();
    for (File f : wsmlFiles){
      ontologies.add(parseWSMLFile(f));
    }
    return ontologies;
  }

  private TopEntity[] parseWSMLFile(File fWsml) throws IOException,
      ParserException, InvalidModelException{
    FileReader wsmlReader = new FileReader(fWsml);
    return wsmlParser.parse(new BufferedReader(wsmlReader));
  }

  ...
}
```

Listing 29: An Adapter for the transformation of WSML to XML

```
<process name="SemBizDeployment"
   ...
  >
    ...
    <partnerLinks>
      <partnerLink name="BPMO2VbM" partnerLinkType="bpmo2vbmpl:BPMO2VbM"
        partnerRole="BPMO2VbMRole"/>
        ...
    </partnerLinks>

    <variables>
     <variable name="TranslateIn" messageType="bpmo2vbmw:TranslateRequest"/>
     <variable name="TranslateOut" messageType="bpmo2vbmw:TranslateResponse"/>
        ...
    </variables>

  <sequence>
        ...
      <assign name="PrepareTranslate">... </assign>
      <invoke name="Translate"
           partnerLink="BPMO2VbM"
           operation="Translate"
           portType="bpmo2vbmpt:BPMO2VbMPortType"
           inputVariable="TranslateIn"
           outputVariable="TranslateOut">
       <documentation>Translates BPMO files to VbMF Views</documentation>
      </invoke>
      ...
      <assign name="PrepareGenerate">...</assign>
      <invoke name="Generate"
           partnerLink="VbMF"
           operation="Generate"
            portType="vbmfpt:VbMFPortType"
            inputVariable="GenerateIn"
            outputVariable="GenerateOut">
          <documentation>Generates BPEL&WSDL out of VbMF Views</documentation>
        </invoke>
        ...
        <assign name="PrepareDeploy">...</assign>
        <invoke name="Deploy"
             partnerLink="VDE"
             operation="deploy"
             portType="vdept:VDEPortType"
             inputVariable="DeployIn"
             outputVariable="DeployOut">
          <documentation>
            Deploys the process using the VDE framework
          </documentation>
        </invoke>
      ...
```

Listing 30: The Sembiz Deployment Process

```
namespace {
          _"http://www.hanival.net/sembiz/ws/orderprovisioning#",
         Plesk _"http://www.hanival.net/sembiz/ws/Plesk#",   ... }

 ontology OrderProvisioning

 importsOntology{
           _"http://www.hanival.net/sembiz/ws/Plesk#Plesk",   ... }

instance OrderProvisioningService memberOf bpmopr#Execution
     bpmopr#executedBy hasValue OrderProvisioningSystem
     bpmopr#hasGrounding hasValue ''OrderProvisioningService.wsdl''

 instance Plesk#PleskService memberOf bpmopr#Execution
     bpmopr#executedBy hasValue Plesk
     bpmopr#hasGrounding hasValue "PleskService.wsdl"
```

Listing 31: Process Setup in BPMO

9 Evaluation

At first we will evaluate the transformation task by showing the transformation of the Hanival use case from the BPMO Ontology to BPEL and WSDL code. We will then summarize the open issues and future work in the following section.

9.1 Evaluation of the practical transformation task

In order to evaluate the BPMO to BPEL code generation we examine transformation semantics, restrictions and problems concerning the transformation of the Hanival use case, the OrderProvisioning process. Starting from the BPMO code, the derivation of its elements is related to the particular VbMF views to finally show the produced BPEL and WSDL counterparts.

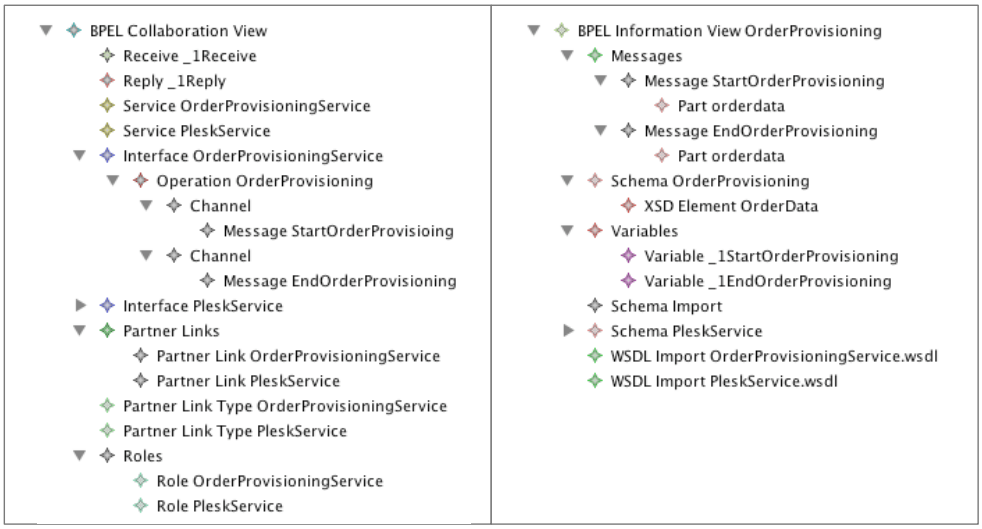
9.1.1 Mapping the Process Setup

The first thing to consider when setting up a BPEL process is the process's setup. A lot of crucial information like namespaces or the import of external resources is declared here. Listing 31 shows the process setup of the OrderProvisioning process in BPMO which starts with the initial section for namespace declarations. Here, the local namespace, as well as any other namspaces, e.g. the PleskService, are bound. Additionally, in order to use it's elements in the course of the process definition, an *importsOntology* entry has to exist for each participating ontology. Again, the PleskService gives an example for an imported ontology.

BPEL Transaction View In case of the process setup, only the root element for the OrderProvisioning process is created in the BPEL Transaction View. It's

the collaboration and information view which are more important for this step.

BPEL Collaboration View For the BPELCollaboration, again the view is created. Partner information, mainly coming from the *Execution*s, is translated in form of a *Service*, where the Service shares the execution's name as well as its namespace. Further for each Execution, an Interface as well as a PartnerLink, a PartnerLinkType and a Role is created an connected to related elements. Figure 51 (a) shows the collaboration model after transforming the process setup.



(a) (b)

(c)

Figure 51: OrderProvisioning - BPEL Collaboration View (Generated EMF Editor)

BPEL Information View Finally the BPELInformationView is generated as shown by Figure 51 (b). Different to the other VbMF views, the information view additionally carries the namespace for the OrderProvisioning. Here, for each involved Execution a WSDLImport is created using the Execution's `identifier` and `hasGrounding` attributes. Further, a Schema is made for every Execution as well as a SchemaImport for each external partner.

BPEL and WSDL Listing 32 and 33 show the final outcome of the process setup in terms of BPEL and WSDL. Looking at the generated code, it becomes

```
<process name="OrderProvisioning"
  targetNamespace="http://www.hanival.net/sembiz/ws/orderprovisioning#"
  xmlns:tns-1725924166 ="http://www.hanival.net/sembiz/ws/Plesk#">
 <import importType="http://schemas.xmlsoap.org/wsdl/"
    location="OrderProvisioningService.wsdl"
    namespace="http://www.hanival.net/sembiz/ws/orderprovisioning#"
    />
 <import importType="http://schemas.xmlsoap.org/wsdl/"
    location="PleskService.wsdl"
    namespace="http://www.hanival.net/sembiz/ws/Plesk#"
    />
<partnerLinks>
    <partnerLink name="OrderProvisioningService"
        partnerLinkType="tns1882850496:OrderProvisioningService"
        myRole="OrderProvisioningService"
        />
    <partnerLink name="PleskService"
      partnerLinkType="tns-1725924166:PleskService"
      partnerRole="PleskService"
      />
  </partnerLinks>
  ...
 </process>
```

Listing 32: Process Setup in BPEL

clear that a lot of elements depend on the crucial information kept in the Execution instances.

9.1.2 Mapping the Process Start and End

In BPMO, the root process is required to have an Execution as well as appropriate pre- and postcondition events in it's Capability. More precisely a `Receive` is made, if the precondition contains a Start resp. a StartMessage event. On the contrary a `Reply` is made if the postcodition contains an End or an EndMessage event. Listing 34 shows the relevant sections of the Hanival use case.

BPEL Transaction View For the process start, the first (or root) activity is made in the `BPELTransactionView`. In the case of the example, the first activity is a Sequence where it's first contained activity is made implicitly for the start (Receive) and the last contained activity is made implicitly for the end (Reply) of the process. Between start and end, all the other sub-activities are created. Because processes are referenced multiple times in BPMO but exist only once as well as naming constrains on BPEL activities, the names of the actual process are replaced according to their position in the process flow as shown by Figure 52 where the start Sequence is named "_1" (see 9.1.3).

```
<wsdl:definitions
   targetNamespace="http://www.hanival.net/sembiz/ws/orderprovisioning#"
   xmlns="http://www.hanival.net/sembiz/ws/orderprovisioning#"
   xmlns:tns-1725924166="http://www.hanival.net/sembiz/ws/Plesk#"
   ...
>
...
  <wsdl:portType name="OrderProvisioningService">
   ...
  </wsdl:portType>
  <plnk:partnerLinkType name="OrderProvisioningService">
    <plnk:role
        name="OrderProvisioningService"
        portType="tns1882850496:OrderProvisioningService"/>
  </plnk:partnerLinkType>
</wsdl:definitions>
```

Listing 33: Process Setup in WSDL

```
instance OrderProvisioning memberOf bpmopr#SequenceProcess
    bpmopr#hasName hasValue "OrderProvisioning"
    bpmopr#withCapability hasValue cOrderProvisioning
    bpmopr#withExecution hasValue OrderProvisioningService
   // other attributes ...

instance cOrderProvisioning memberOf bpmopr#Capability
     bpmopr#hasPrecondition hasValue cOrderProvisioningPrecAxiom
     bpmopr#hasPostcondition hasValue cOrderProvisioningPostAxiom

 axiom cOrderProvisioningPrecAxiom
     definedBy
          ?cOrderProvisioningPrec memberOf StartOrderProvisioning.

 axiom cOrderProvisioningPostAxiom
     definedBy
          ?cOrderProvisioningPostc1 memberOf
                EndOrderProvisioning.

concept StartOrderProvisioning subConceptOf {bpmo#StartEvent, bpmo#Message}
        bpmo#hasData ofType (1 1) hanival#OrderData

concept EndOrderProvisioning subConceptOf {bpmo#EndEvent, bpmo#Message}
        bpmo#hasData ofType (1 1) hanival#OrderData
```

Listing 34: The Process Start and End in BPMO

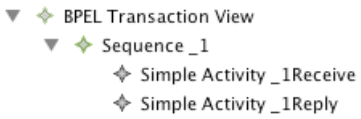


Figure 52: OrderProvisioning - BPEL Transaction View (Generated EMF Editor)

BPEL Collaboration View Here, the concrete interactions Receive and Reply are added for the root process which are named exactly like the implicit activities made in the BPELTransactionView. Figure 53 shows the Receive interaction "_1"Receive and it's properties. The operation named 'OrderProvisioning' in

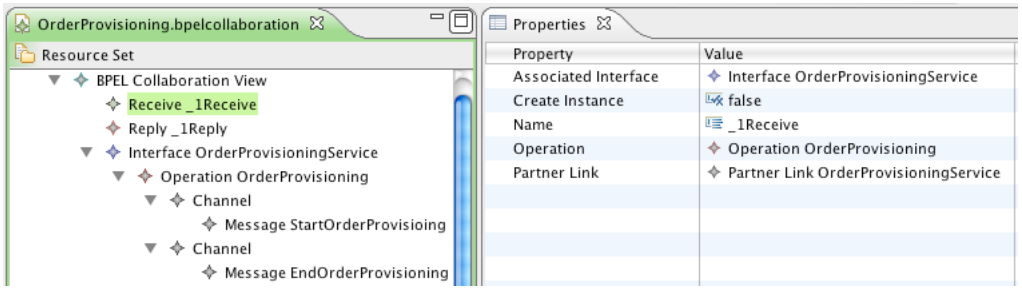


Figure 53: Process Start in the BPEL Collaboration View (Generated EMF Editor)

the Interface is then derived from the root process's name whereas `Interface`, `partnerLink` and `portType` are derived from the process's Execution and the used `variables` are derived from the `MessageEvents` (compare Listing 34).

BPEL Information View Here first a Schema element is created for the root process. It contains all required data definitions to fulfill the start and end interaction's needs. For external resources a SchemaImport is made which contains a link to the actual schema file.

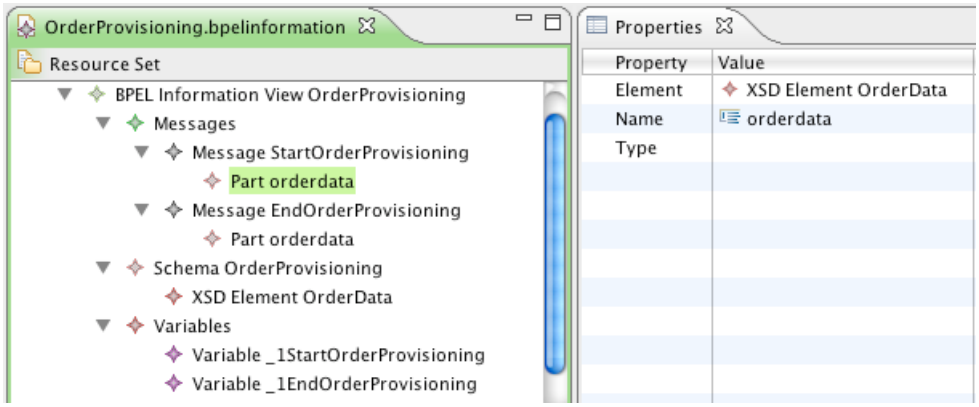


Figure 54: Process Start in the BPEL Information View (Generated EMF Editor)

Additionally Messages and variables related to process start and end are created. Again watch the naming of the variables in Figure 54.

```
<sequence>
    <receive createInstance="yes" name="_1Receive"
      operation="OrderProvisioning"
      partnerLink="OrderProvisioningService"
      portType="tns1882850496:OrderProvisioningService"
      variable="_1StartOrderProvisioning"/>
     ...
     <!-- rest of the process ... -->
     ...
    <reply name="_1Reply"
      operation="OrderProvisioning"
      partnerLink="OrderProvisioningService"
      portType="tns1882850496:OrderProvisioningService"
      variable="_1EndOrderProvisioning"/>
 </sequence>
```

Listing 35: The Process Start and End in BPEL

BPEL and WSDL The translation of the process start results in a variety of impacts on the BPEL and WSDL files gerated. Listing 35 shows the relevant part of the BPEL process. The process start results in a `Sequence` with a follow up `Receive` activity. The `Sequence` is the root container for the process-flow therefore containing all other activities. A `Receive` activity links to a `partnerLink`, a `portType`, an `operation` and refers to an input `variable`. Further, the `createInstance` attribute is set to *true*. Finally, a Reply is made at the end of the process pointing to the same operation, partnerLink and portType as the initial Receive, but owning a dedicated variable.

In WSDL, as shown by Listing 36, the process start and end results in the definition of the operation provided by the final process (OrderProvisioning) including the definition of it's input- and output messages.

9.1.3 Mapping Atomic Processes

From a composite business process, usually `AtomicProcesses` are referenced, which have to be mapped to Web service calls. There are some issues, which have to be taken into account when defining the mapping:

- Not every `AtomicProcess` actually represents a Web service call.
- Further, as one `AtomicProcess` may be referenced more than one time, it is not possible to use its name for the corresponding activity's name (due to the named matching algorithm of the VbMF).
- At last, data handling / variable preparation is not modeled in the ontology.

```
<wsdl:message name="StartOrderProvisioning">
  <wsdl:part element="provided:OrderData" name="orderdata"/>
</wsdl:message>
  <wsdl:message name="EndOrderProvisioning">
  <wsdl:part element="provided:OrderData" name="orderdata"/>
</wsdl:message>
<!-- Rest of messages -->
<wsdl:portType name="OrderProvisioningService">
  <wsdl:operation name="OrderProvisioning">
    <wsdl:input message="provided:StartOrderProvisioning"
    name="StartOrderProvisioning"/>
    <wsdl:output message="provided:EndOrderProvisioning"
    name="EndOrderProvisioning"/>
    <wsdl:fault
      message="provided:ManualInteractionServerAllocationException"
      name="ManualInteractionServerAllocationException"/>
    <wsdl:fault
      message="provided:ManualInteractionUserConfigurationException"
      name="ManualInteractionUserConfigurationException"/>
  </wsdl:operation>
</wsdl:portType>
```

Listing 36: Process Start in WSDL

To solve these issues, we first identify those `AtomicProcesses`, that represent Web services, which are those that have `Messages` as pre- and postconditions. We then assign a unique name for each atomic process reference, and create two simple activities in the *BPELTransaction View*. The first's name matches an assign activity in the *BPELInformation View*, the second's name matches an invoke interaction in the *BPELCollaboration View*. Additionally, variables are created for in- and output of this activity, and linked to the invoke and the assign. The latter have to be completed manually in the resulting BPEL code.

The first AtomicProcess within the main OrderProvisioning sequence is the PleskUserLookup. In listing 37, we show a comprehensive excerpt from the Plesk ontologies for this process. In terms of BPEL, it has to be translated to a Web Service operation call, but this has some additional requirements.

BPEL Transaction View In the BPELTransactionView, two simple activities are created for the PleskUserLookup, as can be seen in Figure 55. One represents the actual Web service call ("_2"), the other the assign activity before ("_2Preparation").

BPEL Collaboration View In the collaboration view, as can be seen in Figure 56, first the service description is created. This means, that for the execution PleskService, a Service, a Partnerlink, a partnerLinkType, a role and a corresponding interface are created. The PleskUserLookup represents one operation of this service's interface. The capability of the PleskUserLookup,

```
instance PleskService memberOf bpmopr#Execution
     bpmopr#executedBy hasValue PleskSystem
     bpmopr#hasGrounding hasValue
     _"http://localhost:8080/SembizServiceLandscape/PleskService"

instance PleskUserLookUp memberOf bpmopr#AtomicProcess
     bpmopr#hasName hasValue "PleskUserLookUp"
     bpmopr#withCapability hasValue cPleskUserLookUp
     bpmopr#withExecution hasValue PleskService

instance cPleskUserLookUp memberOf bpmopr#Capability
     bpmopr#hasPrecondition hasValue cPleskUserLookUpPrecAxiom
     bpmopr#hasPostcondition hasValue cPleskUserLookUpPostAxiom

axiom cPleskUserLookUpPrecAxiom
     definedBy
          ?cPleskUserLookUpPrec memberOf PleskUserLookUpRequest.

axiom cPleskUserLookUpPostAxiom
     definedBy
          ?cPleskUserLookUpPost memberOf PleskUserLookUpResponse.

concept PleskUserLookUpRequest subConceptOf {bpmo#IntermediateEvent,
                                             bpmo#Message}
        bpmo#hasData ofType (1 1) hanival#CustomerData

concept PleskUserLookUpResponse subConceptOf {bpmo#IntermediateEvent,
                                              bpmo#Message}
        bpmo#hasData ofType (1 1) PleskUserLookUpResponseData

concept PleskUserLookUpResponseData subConceptOf {bpmo#BusinessData}
        return ofType (1 1) hanival#CustomerExistsData
```

Listing 37: PleskUserLookup in BPMO (excerpt from the Plesk ontologies)

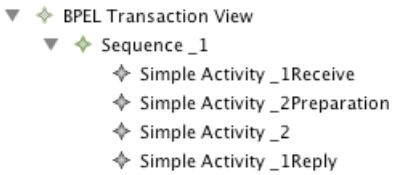


Figure 55: PleskUserLookup - BPMO Business Process View (Generated EMF Editor)

cPleskUserLookup describes in- and output of the operation through its pre- and postcondition. Therefore, the messages PleskUserLookupRequest and PleskUserLookupResponse are created and linked to the operation via its in- and out channels. Further, a corresponding invoke interaction is created, representing the

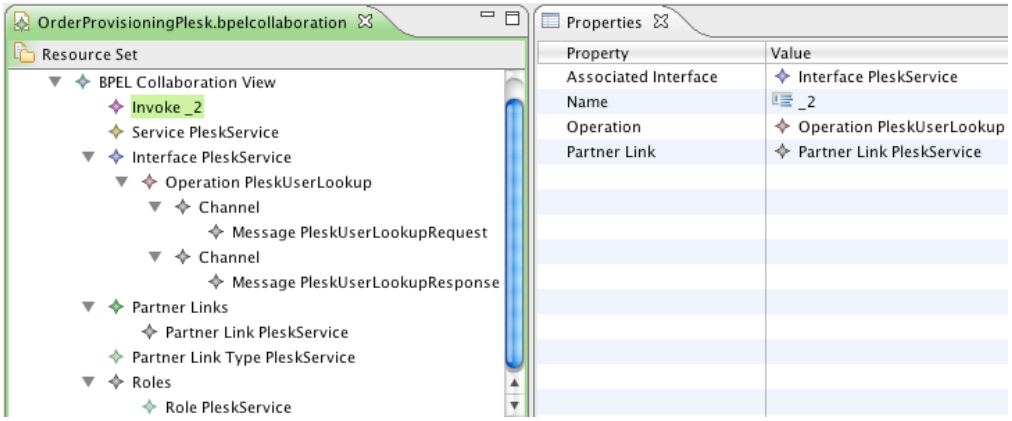


Figure 56: PleskUserLookup - BPEL Collaboration View (Generated EMF Editor)

Web service call and named like the second activity in the BPELControlflow view ("_2"). By using appropriate names, this interaction then is linked to in- and output variables (_2_PleskUserLookupRequest and _2_PleskUserLookupResponse) which themselves correspond to the messages that have been created for the operation.

BPEL Information View In the BPELInformation View (see Figure 57, two things have to be done for the PleskUserLookup: to define the data definition for the used variables and messages (finally BPEL variables and XML Schema) and to care about variable preparation (finally BPEL assigns).
The first (data definition) will be explained at the example of the PleskUserLookupRequest. For the BPMO Message PleskUserLookupRequest, a corresponding BPEL Information message is defined, name and namespace taken from the BPMO equivalent. For each BusinessData type in the Message's hasData attribute, a part is created in this message, referencing an XSD Element in the Plesk namespace. BusinessData are then recursively mapped to XSD Types and XSD Elements, mapping WSML attributes to XSD Elements. For the second task (variable preparation), an assign data handling is created, which name matches the one of the first activity in the BPELTransaction View ("_2Preparation"). Per default only the input variable, in this case _2_PleskUserLookupRequest, is manipulated. For the corresponding message's parts, a copy statement with a literal as source

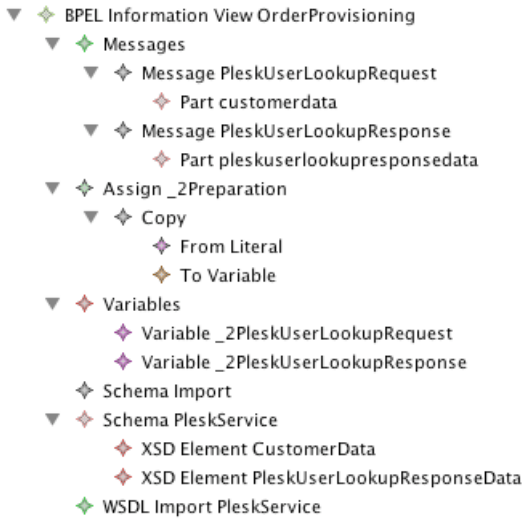


Figure 57: PleskUserLookup - BPEL Information View (Generated EMF Editor)

(from) and the variable _2_PleskUserLookupRequest as target (to) is set. As the BPMO meta-model doesn't capture data conversion, assigns have to be completed manually.

BPEL and WSDL The final output after code generation for the PleskUserLookup in terms of BPEL can be seen in Listing 38. Here along with partner description and variable declaration, an empty assign activity is for the actual service call, the invoke activity.

9.1.4 Mapping Atomic Process Invariants

`Atomic Process`es are not only used to model Web Service operations, but also to model timing and exception mechanisms. `Atomic Process`es without an `Execution` and with an `Exception` event as postcondition are mapped to `Throw` activities. Listing 39) gives an example of how a Throw is modeled in BPMO, Listing 40 shows the resulting BPEL code for this atomic process.

Another case would be if an `Atomic Process` had a `Timer` event as a precondition. Here a `Wait` activity would be created in the BPELTransactionView.

9.1.5 Mapping Composite Processes

The mapping rules for `CompositeProcesse`s are quite straight forward, as they only affect the *BPELTransaction View*. Consequently, and in contrast to atomic processes, a rewriting of names is not necessary. A `ParallelProcess`

```
<partnerLinks>
 <partnerLink name="PleskService"
  partnerLinkType="tns-1725924166:PleskService"
  partnerRole="PleskService"/>
  ...
</partnerLinks>
<variables>
 <variable messageType="tns-1725924166:PleskUserLookUpRequest"
    name="_2_PleskUserLookUpRequest"/>
 <variable messageType="tns-1725924166:PleskUserLookUpResponse"
    name="_2_PleskUserLookUpResponse"/>
 ...
<variables>
...
<assign name="_2Preparation">
 <copy>
  <from>
   <literal>
    <!-- variable initialization code -->
   </literal>
  </from>
  <to variable="_2_PleskUserLookUpRequest"/>
 </copy>
</assign>
<invoke inputVariable="_2_PleskUserLookUpRequest" name="_2"
   operation="PleskUserLookUp"
   outputVariable="_2_PleskUserLookUpResponse"
   partnerLink="PleskService" portType="tns-1725924166:PleskService"/>
```

Listing 38: PleskUserLookup in BPEL (excerpt from OrderProvisionging.bpel)

```
instance EndServerAllocation memberOf bpmopr#AtomicProcess
     bpmopr#hasName hasValue "EndServerAllocation"
     bpmopr#withCapability hasValue cEndServerAllocation

instance cEndServerAllocation memberOf bpmopr#Capability
     bpmopr#hasPostcondition hasValue cEndServerAllocationPostcAxiom

axiom cEndServerAllocationPostcAxiom
     definedBy
          ?x memberOf ManualInteractionServerAllocationException.

concept ManualInteractionServerAllocationException
        subConceptOf {bpmo#EndEvent, bpmo#Exception, bpmo#Message}
        bpmo#hasData ofType (1 1) ManualInteractionServerAllocationError

concept ManualInteractionServerAllocationError subConceptOf bpmo#BusinessData
        errorMessage ofType (1 1) _string
```

Listing 39: Invariant showing an Atomic Process with an Exception as postcondition

```
<assign name="_11Preparation">
  <copy>
    <from>
      <literal>
        <!-- variable initialization code -->
      </literal>
    </from>
    <to variable="_11_ManualInteractionServerAllocationException"/>
  </copy>
</assign>
<throw
  faultName="ManualInteractionServerAllocationException"
  faultVariable="_11_ManualInteractionServerAllocationException"/>
```

Listing 40: A Throw translated into BPEL

```
instance Par1 memberOf bpmo#ParallelProcess
  bpmo#subprocess hasValue {At1, At2}

instance At1 memberOf bpmo#AtomicProcess
instance At2 memberOf bpmo#AtomicProcess
```

is mapped to a BPEL `Flow` and a `SequenceProcess` is mapped to a BPEL `Sequence`. However, the recursive structure of sequences in BPMO through the head-/tailProcess attributes is mostly flattened out - a demonstration is given in the example below by Listing 9.1.5 an 9.1.5. Depending on its `LoopType`, a `LoopProcess` is either mapped to a BPEL `While` (in case of a `WHILE_DO`), a `DoUntil` (in case of a `WHILE_DO`), or a ForEach (in case of a `DO_UNTIL`). However, there is one issue: As a `LoopProcess` is a `CompositeProcess`, it may have more than one `subprocess`, though it is unspecified what happens if it has more than one. So the transformation only accepts `LoopProcess`es with only one `subprocess`.

ParallelProcess Listing 9.1.5 shows a fictional example with a ParallelProcess in BPMO.

Listing 9.1.5 shows the translation of the ParallelProcess into BPEL code.

SequenceProcess Listing 9.1.5 shows a small (fictional) example with several `SequenceProcesses`.

```
<flow name="Par1">
    <invoke operation="At1" ... />
    <invoke operation="At2" .../>
</flow>
```

```
instance Seq1A memberOf bpmo#SequenceProcess
  bpmo#subprocess hasValue {Seq2A, Seq1B}
  bpmo#headProcess hasValue Seq2A       // will be mapped to BPEL sequence
  bpmo#tailProcess hasValue Seq1B       // won't be mapped to BPEL

instance Seq1B memberOf bpmo#SequenceProcess
  ...

instance Seq2A memberOf bpmo#SequenceProcess
  bpmo#subprocess hasValue {At1, Seq1B}
  bpmo#headProcess hasValue At1         // will be mapped to BPEL invoke
  bpmo#tailProcess hasValue Seq2B       // won't be mapped to BPEL

instance Seq2B memberOf bpmo#SequenceProcess
  bpmo#subprocess hasValue {At2}
  bpmo#headProcess hasValue At2         // will be mapped to BPEL invoke

instance At1 memberOf bpmo#AtomicProcess
  ...

instance At2 memberOf bpmo#AtomicProcess
  ...
```

```
<sequence name="Seq1A">
  <sequence name="Seq2A">
    <invoke operation="At1" ... />
    <invoke operation="At2" .../>
  </sequence> <!-- end of Seq2A-->
  ...
</sequence> <!-- end of Seq1A-->
```

Of those `SequenceProcess`es, only the root and those that are referenced from a `headProcess` are mapped to BPEL directly, as can be seen in listing 9.1.5.

LoopProcess Listing 9.1.5 shows examples for `LoopProcess`es in BPMO.

Listing 9.1.5 shows the resulting BPEL code for this example.

```
instance RegisterAllDomain memberOf bpmopr#LoopProcess
     bpmopr#hasName hasValue "RegisterAllDomain"
     bpmopr#subprocess hasValue RegisterAllDomainSeq1
     bpmopr#hasLoopType hasValue bpmopr#WHILE_DO
     bpmopr#hasRule hasValue RegisterAllDomainsRule

instance TryServerAllocation memberOf bpmopr#LoopProcess
         bpmopr#hasLoopType hasValue bpmopr#DO_UNTIL
         bpmopr#subprocess hasValue ServerAllocation1
```

```
<while name="RegisterAllDomain">
        <condtion>
        <!-- here comes the condition breaking the loop -->
        </conditon>
        <sequence name="RegisterAllDomainSeq1">
        ...
        </sequence>
</while>
...
<repeatUntil name="TryServerAllocation"
        <sequence name="ServerAllocation1">
        ...
        </sequence>
        <condtion>
        <!-- here comes the condition breaking the loop -->
        </conditon>
</repeatUntil>
```

9.1.6 Mapping Conditional Branches

The mapping of `ConditionalProcesses` depends on their `SelectType`. Here, the mapping can't be done directly, because there exists no exact counterpart for the `XOR`, `OR` and `CASE` `SelectTypes` neither in VbMF nor in BPEL.

Listing 41 shows an example of a `XOR-SelectProcess` modeled in BPMO. Differently to the semantic source, where a single axiom provides the selection rules for all possible execution branches (Select1RuleAxiom), rules are divided into multiple conditions contained by `ConditionalProcesses` in the *BPMO Process View*.

BPEL Transaction View As shown by Figure 58, a `XOR-SelectProcess` can be mapped to a `Switch`, where every `ConditionalProcess` is mapped to a `Case` contained by the `Switch`'s `case` attribute. A `ConditionalProcess` without a condition can be mapped to a `Case` contained by the `Switch`'s `otherwise` attribute. The mapping of a `CASE-SelectProcess` results in the same structural form as the mapping of a `XOR-SelectProcess`.

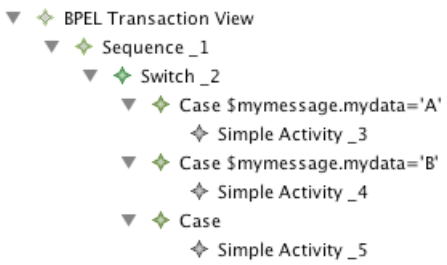


Figure 58: Mapping XOR and CASE SelectProcesses (Generated EMF Editor)

```
instance Select1 memberOf bpmopr#SelectProcess
     bpmopr#hasName hasValue "Select1"
     bpmopr#subprocess hasValue {BP1, BP2, BP3}
     bpmopr#hasSelectType hasValue bpmopr#XOR
     bpmopr#hasRule hasValue Select1Rule

instance Select1Rule memberOf bpmo#BusinessRule
     bpmo#hasName hasValue "Select1Rule"
     bpmo#hasAxiom hasValue Select1RuleAxiom

axiom Select1RuleAxiom
     definedBy
          Select1RuleAxiom(BP1) :-
                ?x memberOf MyMessage and
               ?x[hasData hasValue ?y] and
              ?y memberOf MyData and
                ?y[switchingValue hasValue 'A']
           Select1RuleAxiom(BP2) :-
              ?x memberOf MyMessage and
              ?x[hasData hasValue ?y] and
              ?y memberOf MyData and
              ?y[switchingValue hasValue 'B'].
```

Listing 41: A SelectProcess in BPMO

An `OR-SelectProcess` is not mapped directly, instead, every `ConditionalProcess` is mapped to a `Switch` containing exactly one `Case` as illustrated by Figure 59.

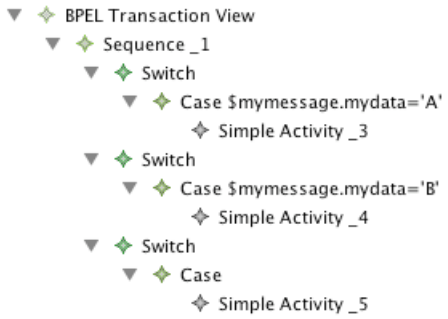


Figure 59: Mapping OR SelectProcesses (Generated EMF Editor)

BPEL and WSDL The final mapping result for `XOR`- and `CASE-SelectProcesses` in terms of BPEL can be seen in listing 42 while Listing 43 represents the outcome when the `SelectType` is set to `OR`.

```
...
<if>
        <condition>...</condition>
        <invoke operation='BP1' ... />
        <elseif>
                <condition>...</condition>
                <invoke operation='BP2' ... />
        </elseif>
        <else>
                <invoke operation='BP3' ... />
        </else>
</if>
...
```

Listing 42: XOR- and CASE SelectProcess mapping result in BPEL

```
...
<if>
        <condition>...</condition>
        <invoke operation='BP1' ... />
</if>
<if>
        <condition>...</condition>
        <invoke operation='BP2' ... />
</if>
<if>
        <condition>...</condition>
        <invoke operation='BP3' ... />
</if>
...
```

Listing 43: OR-SelectProcess mapping result in BPEL

```
instance ServerAllocation1 memberOf bpmopr#SequenceProcess
     bpmopr#hasName hasValue "ServerAllocation1"
     bpmopr#headProcess hasValue ccbs#ServerAllocation
     bpmopr#subprocess hasValue {ccbs#ServerAllocation }

relationInstance bpmopr#hasFaultHandler(ServerAllocation1 ,
        ManualInteractionServerAllocation1 ,
        ccbs#iServerAllocationError)

instance ManualInteractionServerAllocation1 memberOf bpmopr#SequenceProcess
     bpmopr#hasName hasValue "ManualInteractionServerAllocation1"
     bpmopr#headProcess hasValue ccbs#ManualInteractionServerAllocation
     bpmopr#subprocess hasValue {ccbs#ManualInteractionServerAllocation ,
        ManualInteractionServerAllocation2 }
     bpmopr#tailProcess hasValue ManualInteractionServerAllocation2

concept ccbs#ServerAllocationError
        subConceptOf {bpmo#IntermediateMessage , bpmo#Exception}
        bpmo#hasData ofType (1 1) hanival#CustomerData

instance ccbs#iServerAllocationError memberOf ccbs#ServerAllocationError
```

Listing 44: Fault- and Compensation handling in BPMO

9.1.7 Mapping Fault- and Compensationhandling

In BPMO, fault and compensation handling is expressed through instances of the relations `hasFaultHandler` and `hasCompensationHandler`. Listing 44 shows an excerpt of the relevant parts from the OrderProvisioning process. Remarkable here is that because *relationInstances* only accept *instances*, an instance of the ccbs#ServerAllocationError concept is required in order to express the fault handling.

Transformed to the `BPMO Process View`, FaultHandler and CompensationHandler are implemented as standalone elements, operating on the instance level.

BPEL Transaction View When mapping the control-flow, for each passed process the existence of `Fault-` and `CompensationHandlers` for this process is evaluated. If a `Fault-` or `CompensationHandler` exists, the resulting `Activity` is wrapped by a `Scope`. Then, for each `FaultHandler` in the *BPMOProcess View* a `BPELCatch` is made in the *BPELTransaction View*, where it's `faultName` and `faultVariable` attributes are derived from the related BPMO `Exception` concept (e.g.: ccbs#ServerAllocationError).

The finally produced BPEL process is not immediately executable but requires some refinement and completion which still has to be done by a human engineer. However almost the whole process is set up and the process flow is transformed completely as partly shown by Figure 60 which shows the process in an editor

```
<scope name="_6Scope">
  <faultHandlers>
    <catch faultName="tns-899235814:ServerAllocationError"
           faultVariable="_6_ServerAllocationError"
           faultMessageType="tns-899235814:ServerAllocationError">
      ...
      <invoke inputVariable="_9_ManualInteractionServerAllocationRequest"
              name="_9" operation="ManualInteractionServerAllocation"
              outputVariable="_9_ManualInteractionServerAllocationResponse"
              partnerLink="CCBService"
              portType="tns-899235814:CCBService"/>
      ...
    </catch>
  </faultHandlers>
  <sequence>
  ...
    <invoke inputVariable="_7_ServerAllocationRequest"
            name="_7" operation="ServerAllocation"
            outputVariable="_7_ServerAllocationResponse"
            partnerLink="CCBService"
            portType="tns-899235814:CCBService"/>
  </sequence>
</scope>
```

Listing 45: Fault handling in BPEL (excerpt from OrderProvisionging.bpel)

after it's generation. Just a view minor tasks and configurations have to be done in order to execute.

9.2 Open Issues and future Work

Though the transformation realizes a high degree of automation, a closer alignment between the meta-models of both sides (BPMO and VbMF) could provide additional potential for improving automation. Below we list cases, where a closer alignment would be especially useful:

XML Schema Attributes The BPMO uses the WSML meta-modeling capabilities in order to describe the structure of business data, in deriving new concepts from the BPMO concept BusinessData. This definition takes place on the concept level. The transformation component's task is to convert those concepts into XSD Type definitions, which describe input and output of Web services and variables in BPEL processes. While it is possible to directly derive meaningful and valid XSD type definitions without having a more detailed meta-model in BPMO, this solution has drawbacks concerning the created XSD structure. For example, when creating a Web service and specifying the corresponding, in XML Schema in many cases it is an arbitrary design decision wether to use an XSD attribute or an XSD element to store certain data, which means every Web service has some

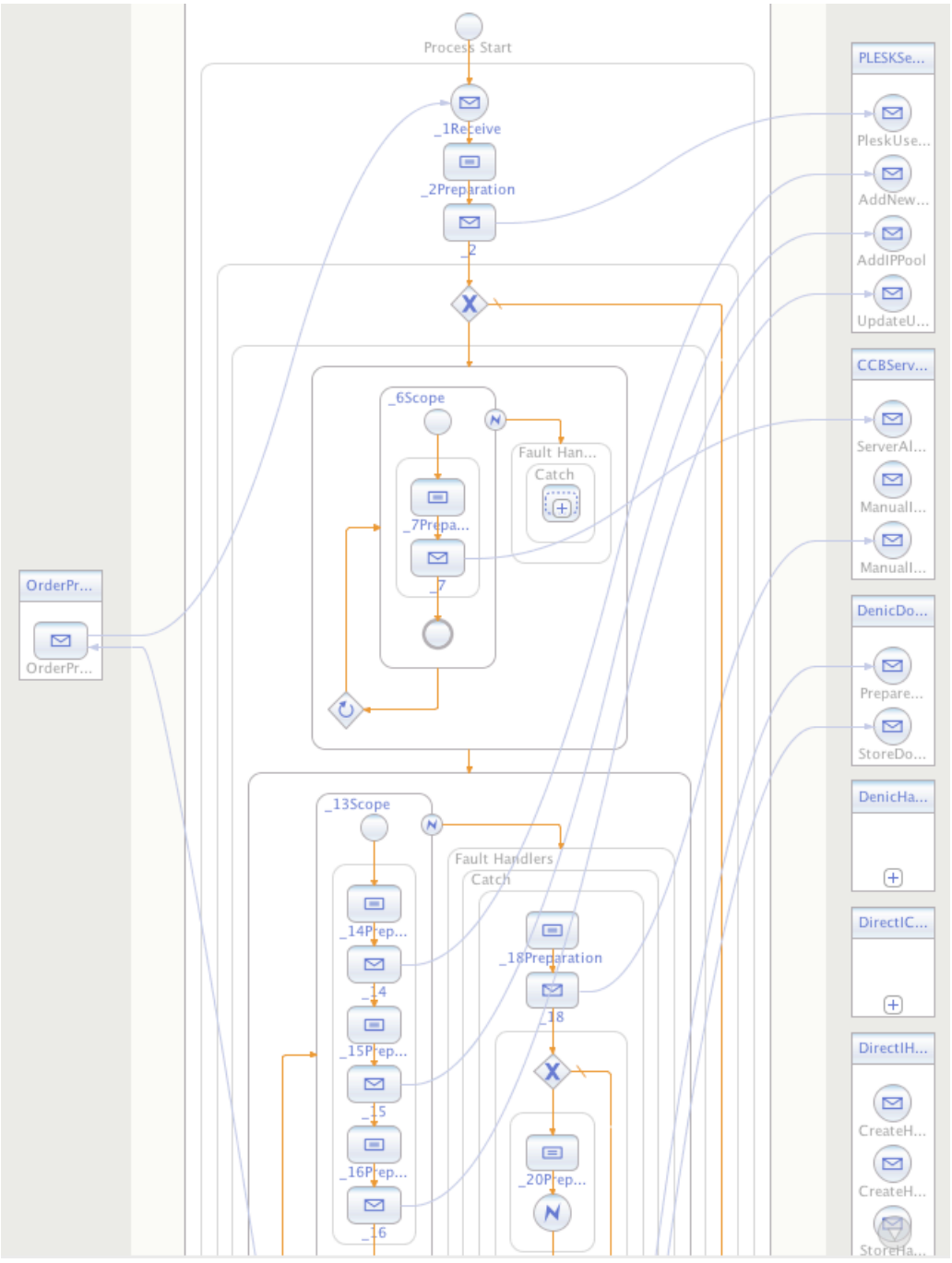


Figure 60: Generated OrderProvisioning BPEL (NetBeans BPEL editor)

extent of freedom in modeling semantically equivalent data. Contrary, this syntactical distinction is not necessary and can't be directly expressed in the WSML meta-model, which means that the transformation component has to specify a meaningful conversion rule. Currently, the used policy is to use XSD Elements only. Consequently, Web services either have to be designed with alignment to the produced types or wrapper services have to be generated. This could be done automatically, e.g. using XSLT. A more flexible solution for this problem could be achieved by a more fine-grained meta-definition for BusinessData structures.

Assign activities In BPEL, assign activities are used to initialize variables and manage data conversion in the control flow. While BPEL has to deal with these fine-grained structures that have to match syntactically, BPMO focuses only on semantic equivalence and similarity, abstracting the syntax details. In BPMO, something similar could be achieved using axioms that describe semantical equivalence between different BusinessData or their parts/attributes. But anyways the axioms would probably not be sufficient, as they have to be syntactically transformed to XPath expressions used in BPEL assign activities. A more flexible solution for this problem could be achieved by a more fine-grained meta-definition for BusinessData conversion which eases a transformation to BPEL.

Axiom to XPath conversion Axioms are very powerful mechanisms for describing BusinessRules and conditions. However, transforming to XPath on a syntactic level is non-trivial when considering the full expressiveness of axioms, so concessions have to be made regarding the form of axiom statements. Further, BPEL execution engines tend to use their own XPath dialects, e.g. for describing conditions for switch/cases, loops and assigns. In order to achieve full platform independence concerning a BPEL engine, it would necessary to provide platform specific XPath rewriting on deployment.

Reverse engineering existing processes and round-tripping Currently, the transformation component works one way only (BMPO to VbMF/BPEL). Using the mechanisms of MDD and the existing intermediate BPMO View, it would be possible and feasible to implement reverse-engineering capabilities, thus enabling a (semi-) automated population of the semantic layer using existing BPEL process and Web service descriptions. This would require a) parsing of BPEL, WSDL and XSD files and populating corresponding VbMF BPEL View models, b) model-to-model transformations to populate the BPMO View, and c) code generation for BPMO/WSML.
Consequently, the development of reverse engineering tools and capabilities could also be used to reflect execution information back to the semantic layer. Addi-

tionally to the efforts necessary to realize reverse engineering, this would require appropriate changes to the meta-models (VbMF as well as BPMO) as well.

10 Conclusion

In this thesis, we have introduced a generic view-based integration process for the generation of executable code from ontologies.

In the first step we use MDSD tools for the creation of ontology meta-model views as well as conceptual ontology views. These ontology views can be used then to translate into implementation views for which code generation facilities already exist. An EMF based ontology meta-model can automatically be derived from an XML Schema definition which typically exists for Semantic Web ontologies. XML representations of actual ontologies are employed to import models conforming to this ontology meta-model. However, the information provided by this view is fine-grained and complicates model transformations to existing views. In order to reduce the follow-up development effort, we present an automatic lifting transformation that creates a conceptual ontology view as well as initial importing transformation rules. Additionally we have investigated the cases where the conceptual ontology view alone is insufficient and manual refinement may be necessary, i.e. when references to the source have to be preserved or when concepts need to be treated like instances.

In a second step, to perform a translation between different views, various model-to-model transformations need to be developed. As this is a complex task, a systematic analysis of source and target views is necessary. It is important to identify standard and special cases to ease design decisions. Our approach is guided by best practices like using examples. They are used to support identifying source information as irrelevant, relevant, hidden, or eventually missing. Additionally, model correspondence cardinalities can be derived from the examples. Further, we have shown how design patterns for model transformations can be applied to address common issues in general as well as special cases identified during analysis.

In the last step we have covered the integration of the developed transformation with the overall project architecture. We have shown how single transformation steps can be composed to a complete transformation workflow. After that, this workflow is exposed as a Web service, that is integrated with the project deployment BPEL process.

The evaluation of the practical work has shown how the concrete transformation from BPMO Ontologies to executable BPEL processes actually works, using a real world use case. Though the reached degree of automation is high, manual work still has to be done by IT specialists. With adequate funding, most open issues could be solved within reasonable time.

A Zusammenfassung auf Deutsch

In dieser Diplomarbeit stellen wir einen generischen, Sichten-basierten Integrationsprozess für die Generierung von ausführbarem Code aus Ontologien vor.

Im ersten Schritt verwenden wir modellgetriebene Softwareentwicklungstools für die Erstellung von Metamodell-Sichten sowie Konzeptuellen Sichten auf Ontologien. Diese Ontologie-Sichten können dann in Implementierungs-Sichten transformiert werden, für welche Möglichkeiten der Codegenerierung schon existiert. Ein EMF basiertes Metamodell kann automatisch aus einer XML Schema Definition, welche für Semantic Web Ontologien üblicherweise existiert, erstellt werden. XML Repräsentationen von Ontologien können dann benutzt werden, um Modelle zu importieren, die konform zum Ontologie Metamodell sind. Die Information dieser Modelle ist jedoch sehr detailliert, was Modelltransformationen zu bereits existierenden Sichten erschwert. Um den folgenden Entwicklungsaufwand zu verringen, zeigen wir eine automatische Transformation, die Konzeptuelle Sichten sowie Import-Transformationsregeln erzeugt. Weiters untersuchen wir jene Fälle, wo eine Konzeptuelle Sicht nicht ausreichend ist und manuelle Verfeinerung notwendig ist, zum Beispiel wenn Modellreferenzen erhalten werden müssen oder wenn Konzepte als Instanzen behandelt werden müssen.

Im zweiten Schritt müssen mehrere Modelltransformationen entwickelt werden, welche die eigentliche Übersetzung zwischen den Ontologie- und den Implementierungs-Sichten durchführen. Da dies eine komplexe Aufgabe ist, sind Quell- und Zielsichten systematisch zu analysieren. Es ist wichtig, zwischen Standard- und Spezialfällen zu unterscheiden, um Entscheidungen beim Entwurf zu erleichtern. Unsere Herangehensweise wird von bewährten Verfahren geleitet, wie die Benutzung von Beispielen. Diese werden verwendet, um Informationen in der Quelle als irrelevant, relevant, versteckt, oder sogar fehlend zu identifizieren. Weiters können die Kardinalitäten von Modellentsprechungen aus den Beispielen abgeleitet werden. Danach zeigen wir, wie Entwurfsmuster für Modelltransformationen angewandt werden können, um allgemeine Probleme sowie Spezialfälle, die während der Analyse identifiziert wurden, zu lösen.

Im letzten Schritt behandeln wir die Integration der entwickelten Transformation mit der allgemeinen Architektur des Projekts. Wir zeigen, wie einzelne Transformationsschritte zu einem kompletten Ablauf kombiniert werden können, der dann als Web Service exponiert und mit dem Deployment BPEL-Prozess integriert wird.

Die Evaluation der praktischen Arbeit zeigt, wie die konkrete Transformation von SemBiz Ontologien in ausführbaren BPEL Code eigentlich arbeitet, am Beispiel eines realen Anwendungsfalls. Obwohl der erreichte Grad an Automatisierung hoch ist, ist dennoch manuelle Nachbearbeitung durch IT-Spezialisten

notwendig. Mit ausreichenden Mitteln könnten die meisten Hindernisse innerhalb einer angemessenen Zeitspanne ausgeräumt werden.

B Lebenslauf Clemens Blamauer

Clemens Blamauer studiert seit 2002 Wirtschaftsinformatik an der Technischen Universität Wien. Im Mai 2006 schloss er das Bakkalaureatsstudium, Schwerpunkt Software Quality Engineering, mit dem Titel "Bakkalaureus der Sozial- und Wirtschaftswissenschaften (Bakk.rer.soc.oec)" ab. Danach setzte er das Masterstudium Wirtschaftsinformatik mit dem Schwerpunkt Internet Computing fort. Während seines Studium war er am Institut für Software Technologie und Interaktive Systeme (Business Informatics Group) als Tutor für die Lehrveranstaltung "Web Engineering" sowie als Studienassistent am Institut für Informationssysteme, Abteilung Verteilte Systeme, tätig.

C Lebenslauf Daniel Lintner

Daniel Lintner begann das Studium Wirtschaftsinformatik an der Universität Wien und an der Technischen Universität Wien im Jahr 2001. Er erhielt seinen Abschluss "Bakkalaureus der Sozial- und Wirtschaftswissenschaften (Bakk.rer.soc.oec)" mit dem Schwerpunkt Software Quality Engineering im März 2006.
Während seines Studium war er am Institut für Software Technologie und Interaktive Systeme (Business Informatics Group) als Tutor für die Lehrveranstaltung "Web Engineering" tätig. Seit 2006 setzt er das Masterstudium Wirtschaftsinformatik mit dem Schwerpunkt Internet Computing fort.

References

[1] Catalog of omg modeling and metadata specifications. http://www.omg.org/technology/documents/modeling2008/11/03.

[2] Eclipse Modeling Framework Project (EMF). http://www.eclipse.org/modeling/emf/, 2008/11/03.

[3] The extensible markup language. http://www.w3.org/XML/, 2008/10/14.

[4] From Models to Code with the Eclipse Modeling Framework. http://www.eclipsecon.org/2005/presentations/EclipseCon2005l.pdf, 2008/11/03.

[5] Generative model transformer (gmt) project description. http://dev.eclipse.org/viewcvs/indextech.cgi/gmt-home/description.html, 2008/11/03.

[6] Mastering eclipse modeling framework. http://www.eclipsecon.org/2005/presentations/EclipseCon20052008/11/03.

[7] Modelware. http://www.modelware-ist.org/, 2009/01/10.

[8] Modelware - description of work. http://www.modelware-ist.org/images/ModelWare/wp_overview.gif, 2009/01/14.

[9] Ontology. a resource guide for philosophers. http://www.formalontology.it/, 2008/10/31.

[10] The resource description framework. http://www.w3.org/RDF/, 2008/10/14.

[11] The web ontology language. http://www.w3.org/TR/owl-features/, 2008/10/14.

[12] The web ontology language guide. http://www.w3.org/TR/2004/REC-owl-guide-20040210/, 2008/10/14.

[13] *FOIS '01: Proceedings of the international conference on Formal Ontology in Information Systems*, New York, NY, USA, 2001. ACM. Conference Chair-Nicola Guarino and Conference Chair-Barry Smith and Conference Chair-Christopher Welty.

[14] Introduction to openarchitectureware 4.1.2. http://www.dsmforum.org/events/MDD-TIF07/oAW.pdf, 2009/01/08.

[15] Sembiz project home page. http://www.sembiz.org/, 2008/10/30.

[16] Owl-s: Semantic markup for web services. http://www.w3.org/Submission/OWL-S/, 2008/10/31.

[17] Wsml language reference. http://www.wsmo.org/TR/d16/d16.1/v1.0/, 2008/10/31.

[18] Web services architecture. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/, 2008/10/14.

[19] C. Alexander. *The city as a mechanism for sustaining human contact*. University of California in Berkeley, California, 1966.

[20] T. Berners-Lee, J. Hendler, and L. Ora. The semantic web. *Scientific American*, 284(5):34–43, May 2001.

[21] J. Bézivin, G. Olsen, F. Allilaire, S. Bonnet, T. Bailey, K. Mantell, and R. Vogel. D1.6-3 identification of transformation patterns. Information Society Technologies, September 2006.

[22] G. Booch, A. Brown, S. Iyengar, J. Rumbaugh, and B. Selic. An mda manifesto. *MDA Journal*, May 2004.

[23] An introduction to Model Driven Architecture. http://www.ibm.com/developerworks/rational/library/3100.html, 2008/10/30.

[24] E. Cimpian, H. Meyer, D. Roman, A. Sirbu, N. Steinmetz, S. Staab, and I. Toma. *Ontologies and Matchmaking*, chapter 3, pages 19 – 54. Springer Berlin Heidelberg, 2008.

[25] J. de Bruijn, C. Bussler, J. Domingue, D. Fensel, M. Hepp, M. Kifer, B. König-Ries, J. Kopecky, R. Lara, E. Oren, A. Polleres, J. Scicluna, and M. Stollberg. D2v1.3 Web Service Modeling Ontology (WSMO). Digital Enterprise Research Institute, 2006.

[26] J. de Bruijn, H. Lausen, R. Krummenacher, A. Polleres, L. Predoiu, M. Kifer, and D. Fensel. D16.1v0.21 the web service modeling language wsml. Digital Enterprise Research Institute, 2005.

[27] S. Dustdar, H. Gall, and M. Hauswirth. *Software-Architekturen für Verteilte Systeme*. Springer-Verlag Berlin Heidelberg, 2003.

[28] S. Dustdar, J. Hoffmann, T. Holmes, A. Sirbu, H. Tran, and U. Zdun. D2.2 Semantic Querying, Discovery, and Composition Framework. Distributed

Systems Group Information Systems Institute Vienna University of Technology Austria, Digital Enterprise Research Institute Leopold-Franzens Universität Innsbruck Austria, 2007.

[29] C. A. Ellis and G. J. Nutt. Office information systems and computer science. *ACM Comput. Surv.*, 12(1):27–60, 1980.

[30] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.

[31] T. R. Gruber and T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5:199–220, 1993.

[32] M. Hammer. Beyond reengineering: How the process-centered organization is changing our work and our lives. *Harper Business*, 1996.

[33] C. Hentrich and U. Zdun. Patterns for business object model integration in process-driven and service-oriented architectures. In *PLoP '06: Proceedings of the 2006 conference on Pattern languages of programs*, pages 1–14, New York, NY, USA, 2006. ACM.

[34] C. Hentrich and U. Zdun. Patterns for process-oriented integration in service-oriented architectures. In *Proceedings of 11th European Conference on Pattern Languages of Programs (EuroPlop 06)*, pages 141–189, Konstanz, Germany, 2006. Universitätsverlag Konstanz.

[35] M. Hepp, F. Leymann, J. Domingue, A. Wahler, and D. Fensel. Semantic Business Process Management: A Vision Towards Using Semantic Web Services for Business Process Management. In *Proceedings of the 2005 IEEE International Conference on e-Business Engineering (ICEBE)*, 2005.

[36] 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, *ECMDA-FA*, volume 5095 of *Lecture Notes in Computer Science*, pages 246–261. Springer, 2008.

[37] I. Horrocks, B. Parsia, P. Patel-Schneider, and J. Hendler. Semantic web architecture: Stack or two towers? pages 37–41. 2005.

[38] IEEE. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE, 2000.

[39] D. Jordan, J. Evdemon, A. Alves, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guizar, N. Kartha, C. K. Liu, R. Khalaf, D. König, M. Marin, V. Mehta, S. Thatte, D. van der Rijn, P. Yendluri,

and A. Yiu. Web services business process execution language version 2.0. OASIS, April 2007.

[40] G. Keller, M. Nüttgens, and A.-W. Scheer. Semantische prozeßmodellierung auf der grundlage ereignisgesteuerter prozeßketten (epk). Saarbrücken, 1992.

[41] F. Leymann and D. Roller. Web Services and Business Process Management. *IBM Systems Journal*, 41(2), 2002.

[42] P. Mayer and D. Lübke. Towards a BPEL unit testing framework. In *Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications (TAV-WEB)*. ACM, 2006.

[43] A. Müller and G. Müller. Model transformation by-example: An eclipse based framework. Master's thesis, Institut für Softwaretechnik und Interaktive Systeme, Vienna University of Technol- ogy Austria, 06 2008.

[44] OMG. Common Warehouse Metamodel, Version 1.1. The Object Management Group Inc., 2003.

[45] OMG. MDA Guide Version 1.0.1. The Object Management Group Inc., 2003.

[46] OMG. Meta Object Facility (MOF) Core Specification, Version 2.0. The Object Management Group, Inc. (OMG), 2006.

[47] OMG. Object Constraint Language, Version 2.0. The Object Management Group Inc., 2006.

[48] OMG. MOF 2.0 /XMI Mapping, Version 2.1.1. The Object Management Group Inc., 2007.

[49] OMG. OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2. The Object Management Group Inc., 2007.

[50] OMG. OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2. The Object Management Group Inc., 2007.

[51] OMG. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Version 1.0. The Object Management Group, Inc. (OMG), 2008.

[52] OMG. MOF Model to Text Transformation Language, v1.0. The Object Management Group, Inc. (OMG), 2008.

[53] Ontology Definition Metamodel. http://www.omg.org/docs/ptc/07-09-09.pdf, 2008/11/19.

[54] J. Z. Pan and I. Horrocks. Metamodeling architecture of web ontology languages. In *In Proceedings of the Semantic Web Working Symposium*, pages 131–149, 2001.

[55] Y. Pan, G. Xie, L. Ma, Y. Yang, Z. Qiu, and J. Lee. Model-Driven Ontology Engineering. *Journal on Data Semantics*, pages 57–78, 2006.

[56] A. J. Pretorius. Ontologies - introduction and overview. unpublished, 2004.

[57] E. Seidewitz. What models mean. *IEEE Softw.*, 20(5):26–32, 2003.

[58] O. Shafiq, M. Moran, E. Cimpian, A. Mocan, M. Zaremba, and D. Fensel. Investigating semantic web service execution environments: A comparison between wsmx and owl-s tools. In *ICIW '07: Proceedings of the Second International Conference on Internet and Web Applications and Services*, page 31, Washington, DC, USA, 2007. IEEE Computer Society.

[59] H. Tran, U. Zdun, and S. Dustdar. View-based and Model-driven Approach for Reducing the Development Complexity in Process-Driven SOA. In *Int. Conf. on Business Process and Services Computing (BPSC)*, 2007.

[60] 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, 2008.

[61] H. Tran, U. Zdun, and S. Dustdar. View-based reverse engineering approach for enhancing model interoperabiity and reusability in process-driven soas. In *10th International Conference on Software Reuse (ICSR'08)*. Springer, 2008.

[62] M. Uschold and M. Gruninger. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11:93–136, 1996.

[63] M. Wimmer, T. Reiter, H. Kargl, G. Kramler, E. Kapsammer, W. Retschitzegger, W. Schwinger, and G. Kappel. Lifting metamodels to ontologies - a step to the semantic integration of modeling languages. In *ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems*, volume 4199 of *Lecture Notes in Computer Science*, pages 528–542. Springer Berlin Heidelberg, 11 2006.

[64] M. Wimmer, M. Strommer, H. Kargl, and G. Kramler. Towards model transformation generation by-example. In *HICSS '07: Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, page 285b, Washington, DC, USA, 2007. IEEE Computer Society.