# Designing and Enacting Cross-organisational Business Processes: A Model-driven, Ontology-based Approach


Stephan Roser

Dissertation

Programming Distributed Systems

Department of Computer Science

University of Augsburg

| | |
|---|---|
| Supervisor: | Prof. Dr. Bernhard Bauer (University of Augsburg) |
| Advisors: | Prof. Dr. Jörg P. Müller (Clausthal University of Technology) |
| | Prof. Dr. Wolfgang Reif (University of Augsburg) |
| Day of defense: | 9th May 2008 |

# Abstract

Under the pressure of globalization, companies are urged to constantly adapt to new market situations and competitors innovations. Focusing on their core business and core competencies, they engage in Cross-organisational Business Processes (CBPs) with new partners all over the world in ever changing constellations. Companies are organized into global networks and outsource those activities that can be performed more quickly and effectively or at lower costs, by others. These developments create new challenges for enterprise Information and Communication Technology (ICT), requiring ICT systems to support constantly changing enterprise collaboration relationships and to create application systems that support or automate business process enactment starting from business level descriptions and models of CBPs.

Model Driven Software Development (MDSD) provides techniques to realize and automate the propagation of changes at the business level to the technical level. MDSD can be used to provide end-to-end support for the realization of business processes, from the business level (users' view) down to deployed applications (ICT view) on specific platforms via well-defined, largely automated model transformations and refinements. However, there still exist several problems that prevent MDSD from being practically applicable for efficient and effective CBP enactment.

This thesis provides contributions that enable MDSD projects to improve their possible impact on software development and the way ICT systems support business. It develops solutions to three main problems, namely for the areas of CBP modelling and enactment infrastructure, ICT architecture selection, and model and transformation evolution. The contributions of this thesis are as follows:

1. We develop architecture patterns, a code generation framework, and model transformations that facilitate the generation of executable code from high-level, domain-specific models. These artifacts help to bridge the semantic gap between domain-specific, high-level business process descriptions and the technologies used to implement them in service-oriented ICT systems. Domain and IT experts benefit since our contribution provides a set of sensible and customizable transformations they can reuse to improve the development of their ICT systems.

2. We investigate new evaluation and decision methods as well as guidelines for the selection of appropriate ICT architectures. We develop a model for decision support that helps IT architects to derive an appropriate architecture paradigm for a given use case or application domain. The decision model combines the Analytic Hierarchy Process (AHP) with scenario-based architecture evaluation techniques. Further, we describe how contingencies influence ICT system coordination architecture. Our decision method, the scenario descriptions, and a set of guidelines

help IT architects to select and reuse appropriate ICT system coordination architectures for CBP enactment in a timely manner. This helps to gain productivity wins by reducing the development time and improving the quality of development with existing and tested solutions.

3. We develop the Ontology-based Model Transformation (OntMT) approach that facilitates the exchange of models between different enterprises as well as the reuse and evolution of model transformations. OntMT helps to overcome differences in syntax and semantics of modelling formalisms with as little effort as possible. Therefore, we apply semantic and reasoning technology to the domain of MDSD. We develop a higher-order model transformation language that allows to modify model transformations. Organisations and modellers benefit from OntMT since it allows them to exchange, customize, and evolve models and model transformations more efficiently.

# Acknowledgements

I would like to thank all people that supported me in writing my thesis:

- First of all, I am deeply grateful to my supervisor Prof. Dr. Bernhard Bauer for giving me the opportunity to conduct research in the field of Model Driven Software Development. His guidance, support, and motivation were the basis for the successful completion of this thesis.
- I am also indebted to Prof. Dr. Jörg P. Müller for his support in the ATHENA IP project as well as for his feedback and contribution to joint research and publications. He also volunteered to be advisor of my thesis.
- I want to thank Prof. Dr. Wolfgang Reif who accepted to be advisor of my thesis.
- Special thanks go to my colleagues from the Programming Distributed Systems group (in alphabetical ordering) Wolf Fischer, Holger Kasinger, Florian Lautenbacher, Raphael Romeikat, and Viviane Schöbel, who created a friendly and cheerful working atmosphere that I enjoyed a lot. Each of them contributed in his own way to this thesis, e.g. by working together in projects, discussing joint research, or sharing everyday work with me.
- I want to thank all students that helped me to save time with their implementations and to improve the results of my work with their valuable feedback.
- Especially, I want to thank all other people, who directly or indirectly influenced this thesis. The fruitful discussions with them at workshops and conferences as well as in research and industrial projects were an important source of inspiration.
- I gratefully thank all my friends who shared their leisure with me. Their informal support and encouragement had positive influence on this thesis.
- Finally, I want to thank my parents Hannelore and Hans-Dieter Roser. They have always supported and encouraged me to do my best in all matters of life.

# Contents

# Chapter 1

# Introduction

Today, companies are urged to adapt to market pressures and competitors' innovations with increasing speed. They globally search for opportunities and resources and perform only those functions for which the company has expert skills. Companies are organised in global networks and outsource those activities that can be performed more quickly and effectively or at lower costs, by others [270]. However, outsourcing and interacting in global networks will also increase overhead costs for collaboration, coordination, intermediation, etc. One approach to describe the influence of organisational structure on these overhead costs is the transaction cost model. In transaction cost economics [322, 323] Williamson distinguishes between production costs and transaction costs. Production costs are the direct production expenses, i.e. the costs for transforming inputs into outputs. Transaction costs are those costs incurred by indirect production expenses through imperfect economic exchange [318, p.97]. These are costs for information processing necessary to coordinate the work, costs for searching and information about required goods, bargaining costs, or costs for management, monitoring, and administration. With the aim to reduce overall costs, Williamson characterizes transaction costs with the variables *frequency*, *uncertainty*, and *specificity* of assets. These variables are used to determine for which organisational structure transaction costs will be lowest.

In today's economies transaction costs have often become more and more important. For example transaction costs make up more than 30% of the total costs of an automobile and about 50% of the total costs of a Logitech mouse [280]. Business systems govern nearly all forms of transactions by facilitating business relationships and value chains. Thus, in modern economies low transaction costs heavily depend on the capabilities of business systems to keep up with constantly evolving business relationships and cross-organisational value chains. This requires methodologies, methods, software architectures, and infrastructures to support changes to business processes that are defined at the business level and propagated down to the level of Information and Communication Technology (ICT) systems.

Model Driven Software Development (MDSD) [278], a specialization of the new trend of Model Driven Engineering (MDE) [37], provides techniques to realize and automate the propagation of changes at business level to the technical level. MDSD can be used to provide end-to-end support for the realization of business processes, from the business level (users' view) down to deployed applications (ICT view) on specific platforms via well-defined, largely automated model transformations and refinements. MDSD treats models as primary development artifacts, uses models to raise the level of abstraction at which developers create and evolve software [116], and reduces com-

plexity of the software artifacts by separating concerns and aspects of a system under development [122].

Within the context of the European integrated project Advanced Technologies for Interoperability of Heterogeneous Enterprise Networks and their Applications (ATHENA IP) [15], the MDSD paradigm was extended to fit the needs of modelling Cross-organisational Business Processes (CBPs), in order to realize such CBPs. The ATHENA IP was a three years project funded by the European commission with the mission to be a main contributor in the European efforts to enabling enterprises to seamlessly interoperate. In this thesis we also present results that we have achieved in the ATHENA IP.

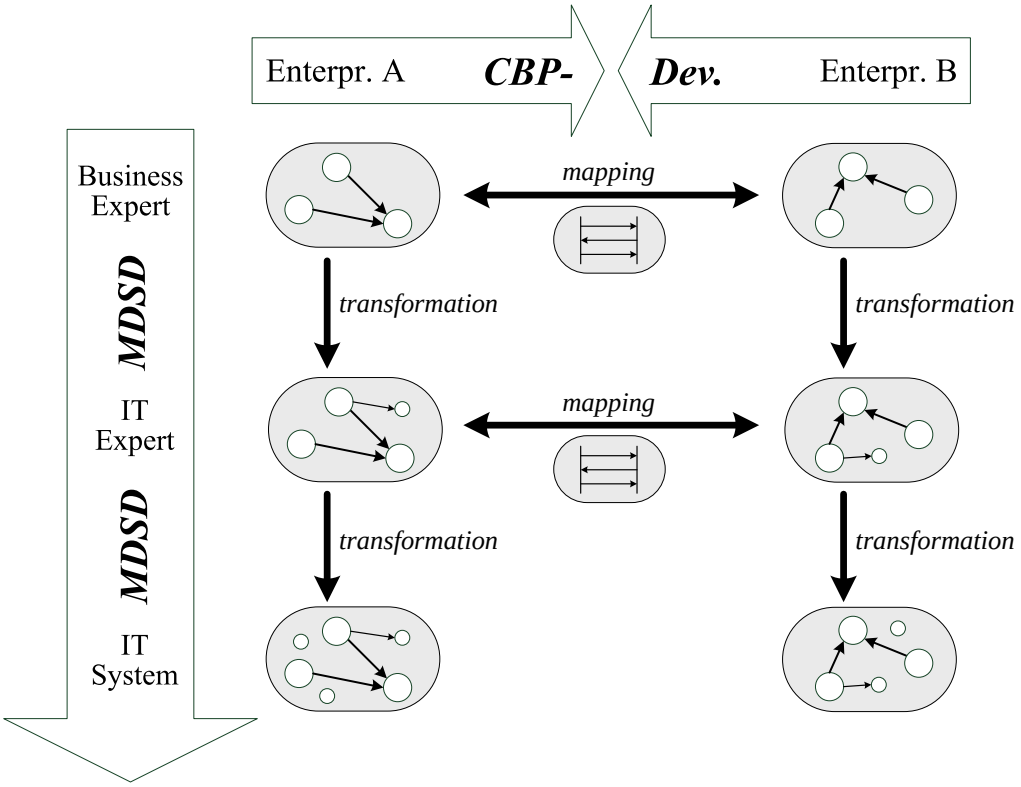


Figure 1.1: Scenario realizing cross-organisational business process modelling and execution

The main goals of the ATHENA IP were to realize more flexible businesses, which are able to move into new markets and product areas rapidly, and to realize more economic businesses through improvements in efficiency, productivity, and cost effectiveness. We contributed to develop and generate executable processes like Web Services Business Process Execution Language (WS-BPEL) [205] processes from models of cross-enterprise collaborations by applying MDSD techniques. Figure 1.1 illustrates the conceptual framework that was applied to develop and enact CBPs with MDSD. The vertical dimension distinguishes the different layers of abstraction applied in MDSD. Enterprise *A* and *B* develop models for their processes at three levels of abstraction, i.e. business-level, platform independent Information Technology (IT) level, and IT system level. The gaps between these abstraction levels are overcome by vertical transformations like presented in [26]. These transformations encode knowledge about architecture and platform in order to transform models from higher level to models of lower abstrac-

tion level. The horizontal dimension represents the collaborative modelling between two enterprises *A* and *B*. To develop CBPs both enterprises have to exchange at least parts of their models as a basis for collaborative modelling and to align their organisations and processes. Hence, models of enterprise *A* and *B* are shared at different abstraction levels via mappings.

## 1.1 Problems and Challenges

In the ATHENA IP the feasibility of model-driven CBP development and enactment was demonstrated. However, there still exist several problems that prevent MDSD from being practically applicable for efficient and effective CBP enactment. This section describes important problems and derives concrete challenges for the areas of CBP modelling and enactment infrastructure, ICT architecture selection, as well as model and transformation evolution.

**CBP Modelling and Enactment Infrastructure.** According to Booch et al. [46] one of the *"greatest difficulties associated with software development is the enormous semantic gap that exists between domain-specific concepts encountered in modern software applications and standard programming technologies used to implement them"*. MDSD has to make use of Domain Specific Languages (DSLs) [114, 183, 289] and shifts the focus of software development away from the technology domain towards the ideas and concepts of the problem domain. To bridge this semantic gap, transformation engines and generators are used to generate code and other target domain artifacts with input from both modelling experts and domain experts [262].

**Problems.** Support of modellers by adequate MDSD tools and assets is a critical precondition to successfully apply MDSD.

- Developing model transformations is a time consuming and error-prone task that requires deep knowledge of model transformation techniques as well as of the problem and solution domains.
- Domain and IT experts are needed to develop sensible model transformations that bridge the gap between high-level business process descriptions and executable code for service-oriented ICT systems.

**Challenge 1.** Provide frameworks, tools, and model transformations that facilitate the generation of executable code from high-level, domain-specific models.

**ICT Architecture Selection.** Once developed, models and model transformations embody critical solutions and insights to enterprises' challenges and hence are seen as assets for an organisation [162]. Assets are artifacts that provide solutions to problems and should be reusable in and customizable to various contexts. If it takes longer to reuse models and model transformations than what seems reasonable, people will start to recreate the content (i.e. the models and model transformations) themselves [162]. The benefit of applying MDSD can be described by the individual daily wins an organisation makes in its software development process through automating the development process but also by capturing and reusing knowledge and critical solutions. To reuse these knowledge and solutions, and to achieve productivity improvements, organisations

have to be able to discover, understand, and customize them for their business and ICT systems in a timely manner.

**Problems.** Treating models as assets requires efficient reuse mechanisms.

- If the selection and customization of models and model transformations does not deliver significant improvements in quality and development time, people will continue building the models from scratch.
- Organisations have to be able to understand and select model transformations for their business and ICT systems in a timely manner.

**Challenge 2.** In the context of developing CBPs, one has to select model transformations which encode software architecture patterns that suit the organisations' and businesses' needs.

**Model and Transformation Evolution.** Without appropriate support MDSD is difficult to adopt. This is especially the case for model transformations, which are often not straightforward to implement and require a significant investment effort. Organisations have to be able to customize model transformations for their business and ICT systems in a timely manner to meet interoperability problems, that arise from the use of DSLs and the constant evolution of models and modelling languages. It has to be possible for organisations to exchange models and reuse existing models and model transformations with as little effort as possible.

**Problems.** Syntactic and semantic differences in representation formats are obstacles that hinder the efficient exchange, customization, and evolution of models and model transformations.

- Modelling languages and metamodels differ though they were developed describing the same application domain. People often associate different semantics with the same metamodel.
- Metamodels evolve as new versions are released, e.g. the metamodels for UML 1.x and UML 2.x. Since models and model transformations depend on metamodel specifications, one not only has to deal with the evolution of metamodels, but also with respect to the evolution of models and model transformations.

**Challenge 3.** It has to be possible to efficiently exchange models between enterprises and customize model transformations for the support of various DSLs despite of differences in syntax and semantics of modelling formalisms.

**Further Challenges.** The described challenges are only those challenges we address in this thesis. They are not an exhaustive list of obstacles one faces with the introduction of MDSD in organisations. Another important aspect of applying MDSD is the definition of methods specifying skills, roles, and responsibilities of people that are involved in the MDSD process. IBM, for example, identifies up to eight organisational roles for their process modelling and execution suite [320, p.4ff]. As discussed in [254], other solutions like AgilPro [28] manage with less roles.

## 1.2 Objectives, Approach, and Contributions

This section identifies objectives to deal with the problems and challenges for the areas of CBP modelling and enactment infrastructure, ICT architecture selection, and model and transformation evolution as described in Section 1.1. Figure 1.2 provides an overview of the objectives of this thesis. We describe the approach we take to meet these objectives and list the contributions of this thesis.

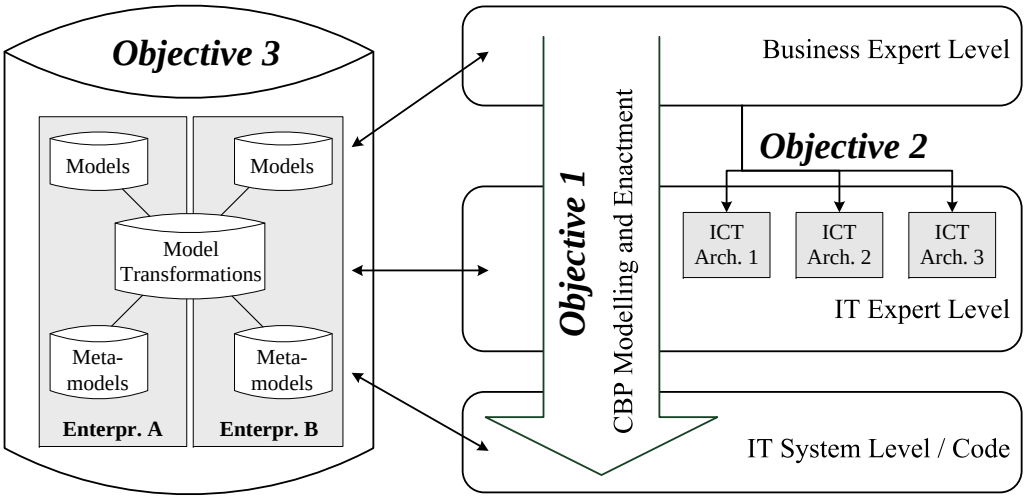


Figure 1.2: Objectives overview

**CBP Modelling and Enactment Infrastructure.** Figure 1.2 depicts model-driven CBP enactment that spans over multiple abstraction levels. Higher-level models are (semi-)automatically refined to more specific models at ICT expert level via transformations. From these models ICT system models and code are generated. As described in *Challenge 1*, this has to be supported by adequate frameworks, tools, and transformations.

**Objective 1.** *Facilitate the generation of executable code from high-level business process descriptions.*

**Approach.** To address this objective, we follow the design science approach. The design science paradigm seeks to extend the boundaries of human and organisational capabilities by creating new and innovative artifacts (for details see [125]). Hence, we develop model transformations and MDSD infrastructure components, that help to bridge different levels of abstraction in ICT system development. The outcomes facilitate the generation of service-oriented ICT systems from high-level business process descriptions.

**Contributions.** We develop the following artifacts for model-driven CBP modelling and enactment:

- Software architecture patterns that enable ICT system coordination in a service-oriented environment.

- Implementation of a set of model transformations that are based on software architecture patterns and allow automated generation of service-oriented Platform Independent Models (PIMs) from computational independent CBP models.
- A model and code generation framework that facilitates the generation of executable workflow code from higher-level process models.
- A case study in which the model and code generation framework is applied to generate WS-BPEL code from arbitrary higher-level process descriptions.

**ICT Architecture Selection.** For the transformation of business expert models to ICT expert models, decisions about ICT architecture have to be made (cp. *Objective 2* in Figure 1.2). According to *Challenge 2* it has to be possible to select and reuse transformations that encode appropriate ICT architectures.

**Objective 2.** *Investigate new decision methods and guidelines for selecting appropriate ICT architectures.*

**Approach.** We study this problem with behavioural science research, where our systems under study are the software architecture patterns developed for the first objective. We develop an evaluation model for the selection of ICT system architecture. The model helps us to explain phenomena, i.e. the influence of contingencies on the architecture selection, that occur with respect to the ICT architectures' use and its impact on ICT systems and organisations [125]. The model captures knowledge that aids in the productive application of information technology to organisations.

**Contributions.** We develop the following artifacts for the selection of adequate ICT system architectures:

- A model for decision support suitable for IT architects to derive an appropriate architecture paradigm for a given use case or application domain. The decision model combines the Analytic Hierarchy Process (AHP) [256] with scenario-based architecture evaluation techniques.
- Scenario descriptions that allow the evaluation and selection of appropriate ICT system coordination architecture paradigms for CBP enactment.
- A set of guidelines of how contingencies influence ICT system coordination architecture based on our experiences in model-driven CBP modelling and enactment.
- We validate our observations (the guidelines) by applying our decision support method, the scenario descriptions for CBP enactment, and the guidelines about contingency influence to application scenarios with differing characteristics.

**Model and Transformation Evolution.** The use of different metamodels in various enterprises results in the need to exchange models via mappings and transformations. Another issue that necessitates the change of models and model transformations is the evolution of metamodels. When a metamodel changes, not only the models ('instances of' the metamodel) but also the model transformations change. To efficiently exchange models and adjust model transformations it is necessary to overcome the differences in syntax and semantics of modelling formalisms (see *Challenge 3*).

**Objective 3.** *Facilitate the technical aspects of exchanging models between different enterprises and the reuse and evolution of model transformations.*

**Approach.** For this objective we again apply the design science paradigm. We develop an approach that uses ontologies and reasoning techniques to overcome differences in syntax and semantics of modelling formalisms. The approach increases the interoperability of enterprises models and modelling tools by automatically generating transformations between the different modelling syntax. The approach also provides mechanisms to automatically customize model transformations to various modelling formalisms, which allow enterprises to more efficiently reuse model transformations.

**Contributions.** We develop the following artifacts to support the exchange of models and the evolution of model transformations.

- We develop the Ontology-based Model Transformation (OntMT) approach, which provides means to automatically deal with model and model transformation evolution scenarios. We implement OntMT and apply it to two real world case studies.
- We introduce and describe an architecture for a semantic-enabled modelling and development suite. Semantic-enabled tools support developers and modellers in a sophisticated manner by making use of reasoning results. We implement OntMT as such a semantic-enabled tool.
- We develop concepts and techniques to realize and implement the OntMT approach. These are bootstrapping rules to generate QVT Relations model transformations [233], a higher-order model transformation language for QVT Relations model transformations as well as representation and reasoning techniques to infer relationships between metamodels. We develop a correlation algorithm and a rating that allow to generate and choose applicable model transformations.
- We develop a higher-order model transformation language that allows to modify model transformations and lends itself for automating reuse of model transformations.

**Overall Objective.** In general, this thesis aims to contribute to a broader dissemination and easier adoption of MDSD. On the basis of the ATHENA IP framework for model-driven CBP enactment (see Figure 1.1) it illustrates the benefits and opportunities of MDSD but also discusses pitfalls and problems in its application. It provides solutions to the problems and challenges identified in Section 1.1. Although the solutions are presented via model-driven CBP enactment, this thesis also yields more general results from which other application domains of MDSD can benefit. It provides solutions to the objectives described in this section in order to enable MDSD projects to fully develop their possible impact on software development and the way ICT systems support business.

## 1.3 Outline

The structure of this thesis is illustrated in Figure 1.3. The arrows indicate suggested reading sequences. However, experienced readers that are familiar with the technology

presented in Chapter 2 and Chapter 3 may skip these chapters or parts of them. Readers have choice of first reading Chapter 4 and Chapter 5 or reading Chapter 6. Chapter 5 uses content introduced in Chapter 4. Summary and conclusions are given in Chapter 7.

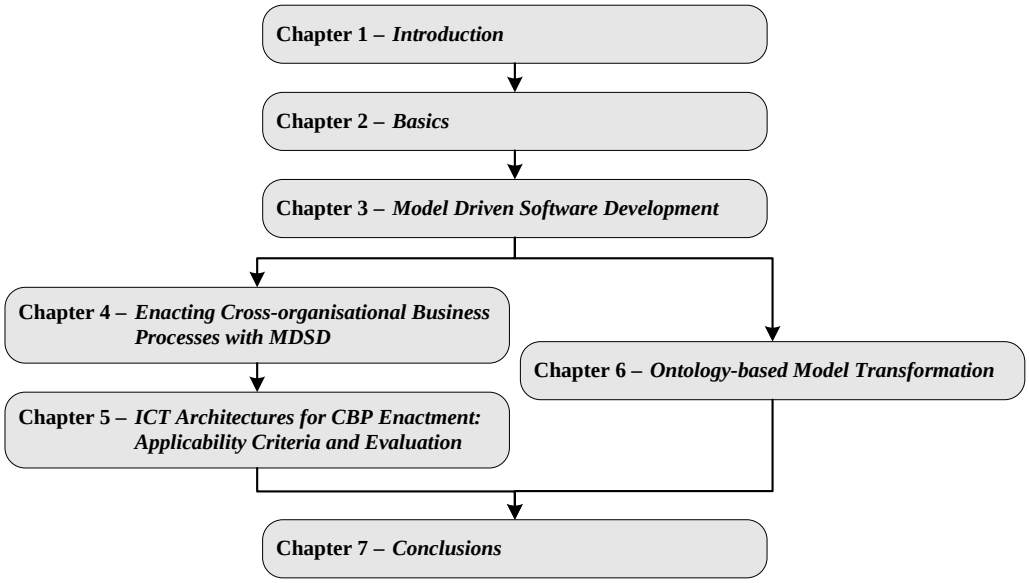


Figure 1.3: Structure of the thesis

**Chapter 1** provides an introduction and motivation to this thesis. It identifies problems and challenges for the practical application of MDSD and defines research challenges and objectives for this thesis. We further describe the method of work we apply to achieve these objectives and summarize our contributions.

**Chapter 2** introduces technology that is relevant for this thesis but does not primarily belong to the domain of MDE and MDSD respectively.

**Chapter 3** describes the current state of the art of MDE and MDSD. It illustrates how software development can benefit from MDE. In addition to approaches, tools, initiatives, and projects that realize MDSD, it introduces and explains core concepts of MDSD like models, metamodels, and model transformations as they are described in literature.

**Chapter 4** demonstrates the definition and implementation of a gradual transition process that transforms CBPs from high-level descriptions to executable, service-oriented implementations. We develop software architecture patterns for service-oriented ICT systems, implement these in model transformations, and present how arbitrary process models are finally transformed to block-structured WS-BPEL code.

**Chapter 5** presents an evaluation and decision method to understand and choose appropriate ICT architectures. This method aims to increase the productivity wins in software development that are gained through MDSD by enabling organisations to more easily reuse existing models and architectures. It helps organisations in understanding the impact of internal and external factors (i.e. contingencies) on the selection of adequate ICT software architectures.

**Chapter 6** introduces the approach of Ontology-based Model Transformation (OntMT). OntMT provides a solution that combines semantic technology with MDSD concepts and allows (semi-)automated generation, reuse, and customization of model transformations. OntMT fosters automated support for the evolution of models and model transformations. This provides the basis to improve interoperability in collaborative MDSD, where models are exchanged between the various participants involved in the MDSD process.

**Chapter 7** summarizes the contributions presented in this thesis and provides an overview of future work and research directions.

# Chapter 2

# Basics

In this chapter we introduce technology that is used or referenced in this thesis but does not primarily belong to the domain of MDE and MDSD respectively. In Section 2.1 we introduce the basic concept of service-orientation and give an overview over the field of service-oriented modelling. Section 2.2 presents process execution and modelling techniques. In Section 2.3 we introduce architecture evaluation and decision methods from which we make use. Finally, we give an overview over semantic technologies in Section 2.4 and discuss their relationship to the field of MDE.

## 2.1 Service-oriented Paradigm

Service-orientation is based on the concept of service, which can be defined *"as a well-defined, self-contained function that does not depend on the context or state of other services"* [45]. In ICT architectures based on service-orientation one can distinguish various notions of basic service models (from [85]):

- Application Service: A service that contains logic derived from a solution or technology platform.
- Business Service: A service that contains business logic.
- Hybrid Service: A service that contains both business and application logic. Most services created as part of traditional distributed solutions fall into this category.
- Controller Service: A service that composes others.
- Process Service: A service that represents a business process as implemented by an orchestration platform and described by a process definition.

In recent years various metamodels, UML profiles and approaches for service-oriented modelling have been developed and published. [143] describes a UML profile for software services that allows the modelling of services, Service Oriented Architecture (SOA), and service-oriented solutions. It focuses on the structural aspects of service modelling and has commonalities with Web Services Description Language (WSDL) [302]. The UML profile presented in [175] focuses on the behavioural aspect of process description and provides a mapping to Business Process Execution Language for Web

Services (BPEL4WS)[1] [129]. In the context of the ATHENA IP [15] a metamodel called Platform Independent Model for Service Oriented Architectures (PIM4SOA) [1, 30, 235, p.51ff] was developed, that allows the specification of service-oriented, platform independent models. This metamodel comprises service, process, information, and quality of service aspects for SOA. It supports the smooth integration into web service composition standards, in particular Web Services Business Process Execution Language (WS-BPEL) [205]. In cooperation with MID[2] we have developed an UML profile for service-oriented modelling called SPL4AOX [23, 235, p.78ff] has been created for and implemented in the Unified Modeling Language (UML) tool MID Innovator AOX eXcellence 2007 [187]. The UML profile description has been extended with mappings to BPEL4WS 1.1 and WSDL.

Also standardization organisations have been concerned with service-orientation and modelling services and service-oriented systems. The Web Service Architecture (WSA) defined by the W3C [312] intends to provide a common definition of a web service, and to define its place within a larger web services framework. It provides a conceptual model and a context for understanding web services and the relationships between the components of this model. The WSA is an interoperability architecture that identifies those global elements of the global web services network that are required in order to ensure interoperability between web services. However, it does not attempt to specify how web services are implemented. The OASIS Reference Model for Service Oriented Architecture [204] is also an abstract framework for understanding significant entities and relationships between these entities within a service-oriented environment, and for the development of consistent standards or specifications supporting that environment. The reference model is not directly tied to any standards, technologies or other concrete implementation details. It seeks to provide a common semantics that can be used unambiguously across and between different implementations. In 2006 the Object Management Group (OMG)[3] issued a request for proposal for an UML Profile and Metamodel for Services (UPMS) [232]. It requests a services metamodel and profile for extending UML with capabilities applicable to modelling services using an SOA. In the current standardization process there are substantially two submissions [236, 235] that provide a metamodel and UML profile for modelling services. Both submission concentrate on defining a core metamodel for modelling service and reference other specifications for related modelling aspects. The submissions reference the Business Process Definition Metamodel (BPDM) [225] for the process modelling aspect and the Business Motivation Model (BMM) [224] to specify service usage and business requirements. The UML Profile and Metamodel for Services – for Heterogeneous Architectures (UPMS–HA) submission additionally provides extensions to more easily apply service-oriented (modelling) techniques for various architectural styles and technologies like web services, service component architectures, peer-to-peer (P2P), Grid, Agents, Event driven architectures and semantic web services.

**Metamodel for Service-oriented Modelling** As one representative of the mentioned approaches, we will shortly introduce PIM4SOA as a metamodel for service-oriented modelling. We describe those parts of PIM4SOA that are essential for the transforma-

[1] BPEL4WS is the predecessor of WS-BPEL. We use BPEL4WS instead of WS-BPEL, whenever work was explicitly developed for BPEL4WS.

[2] `http://www.mid.de`

[3] `http://www.omg.org`

tions described in Section 4.1 and Section 4.2.

In PIM4SOA *Collaborations* are used to specify bi-directional, complex, long-lived and asynchronous interactions between service consumers and providers (see Figure 2.1). A collaboration is a service contract that defines the patterns of interaction between participating roles, i.e. it defines the involved roles and their responsibilities within the collaboration. Collaboration specifications can be nested to arbitrary depth. Collaborations can be defined on the basis of *+subcollaborations* via *CollaborationUses*. A collaboration use specifies a link (*+collaboration*) to the used *Collaborations* definition and a *RoleBinding* (*+bindings*). The *RoleBinding* relates the specific roles within the current collaboration (*+boundRole)* to the roles of the used collaboration (*+role*). A *Role* can be requester or a provider of service. A *ServiceProvider* specifies an entity that provides or consumes services. Therefore it takes on roles through which it *+participates* in collaborations and realizes roles in collaborations. The *RoleBinding* is used to specify which roles of the collaboration are realized by the roles of the service provider. A service provider is *'executable'* or *'abstract'* and owns a *Behaviour* that can be used to describe its internal realization as well as the message exchange with its environment.

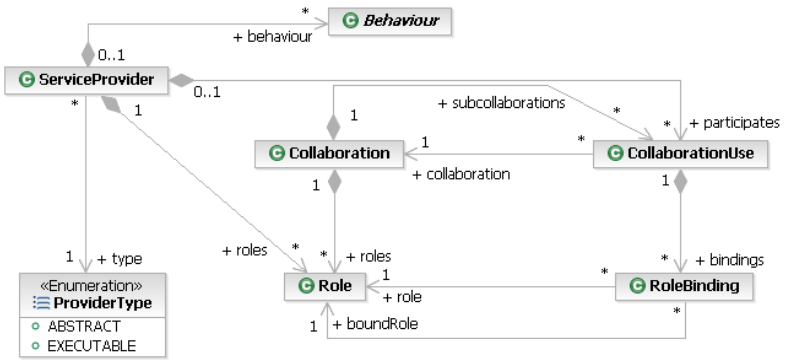


Figure 2.1: PIM4SOA metamodel: service modelling

The communication behaviour as well as the activities, that together realize the provided services, are described by the service provider's behaviour (see Figure 2.2). *Process* implements the behaviour for service providers as a set of tasks and decisions (*Steps*) linked by control flows (*Flows*). *Tasks* represent actions carried out by the process. A task is either an internal processing task that is not further specified and requires implementation beyond the scope of this model, or an interaction with another service provider. In the latter case the (invoked) service is referenced by specifying a *+collaborationUsePath*.[4]. *StructuredTasks* are used to represent subprocesses. Flows provide the links between steps via their interactions. An *Interaction* defines an interface for input or output flows on a step. A *Message* defines a chunk of information sent from one role to another role in a collaboration. Interactions reference the *Messages* they send or receive.

**UML Profile for Service-oriented Modelling** For a clearer description of our usage of PIM4SOA in the following sections we introduce a UML profile for PIM4SOA (see

[4]In order to derive BPEL4WS code from arbitrary platform-independent, service-oriented models automatically, the *collaborationUsePath* would have to reference the respective *RoleBinding* instead to the *CollaborationUse* Examples for this improvement can be found in [23, 232].

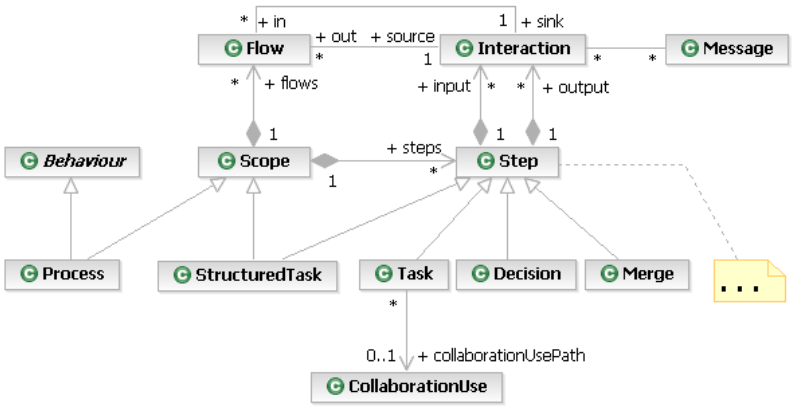


Figure 2.2: PIM4SOA metamodel: process modelling

[1]) that was defined and implemented within the ATHENA IP project [15]. Table 2.1 depicts the definition of the UML profile [1]. The metamodel elements of PIM4SOA became stereotypes in the UML profile. The second column depicts the UML metaclass to which the stereotype adds information.

Figure 2.3 and Figure 2.4 depict simplified excerpts of the Purchase Order example (for more details about the example see [232]) to illustrate modelling with the UML profile for PIM4SOA. In Figure 2.3 one can see two collaborations: *PurchaseOrder* and *Productions*. The *PurchaseOrder* collaboration make use of the *Productions* collaboration via a collaboration use (*productions:Productions*). The roles of the composed collaboration are bound to the roles of the used collaboration, e.g. the role *orderProcessor* is bound to the role *client*. Further the service provider *OrderProcessor* and its role *purchasing* is bound to the role *orderProcessor*, i.e. it participates in the collaboration *PurchaseOrder*.

The behavioural description of the *OrderProcessor* is modelled as a process and activity in UML respectively (see Figure 2.4 for an excerpt). Internal tasks of the *processPurchaseOrder* activity are for example decisions, where the purchase order process has to compute which outgoing control flow has to be followed. The task *requestProductScheduling* is an interaction task that is used to realize an interaction for exchanging messages with the *scheduling* role in the *PurchaseOrder* collaboration. Thus the task references the collaboration use *poParticipation* via which the service provider *OrderProcessor* participates in the *PurchaseOrder* collaboration with a collaboration use path.

## 2.2 Process Modelling and Execution

People with different background, i.e. business or IT, that speak about processes not always mean the same. IT people often refer to the term process in the context of process execution and workflows. Business people use process to describe procedures within and between organisations, that are of coarse granularity at a high level of abstraction and can often not be executed (directly) by workflow engines. Hammer and Champy [123] *"define a business process as a collection of activities that takes one or more kinds of input and creates an output that is of value to the customer"*. The term business process

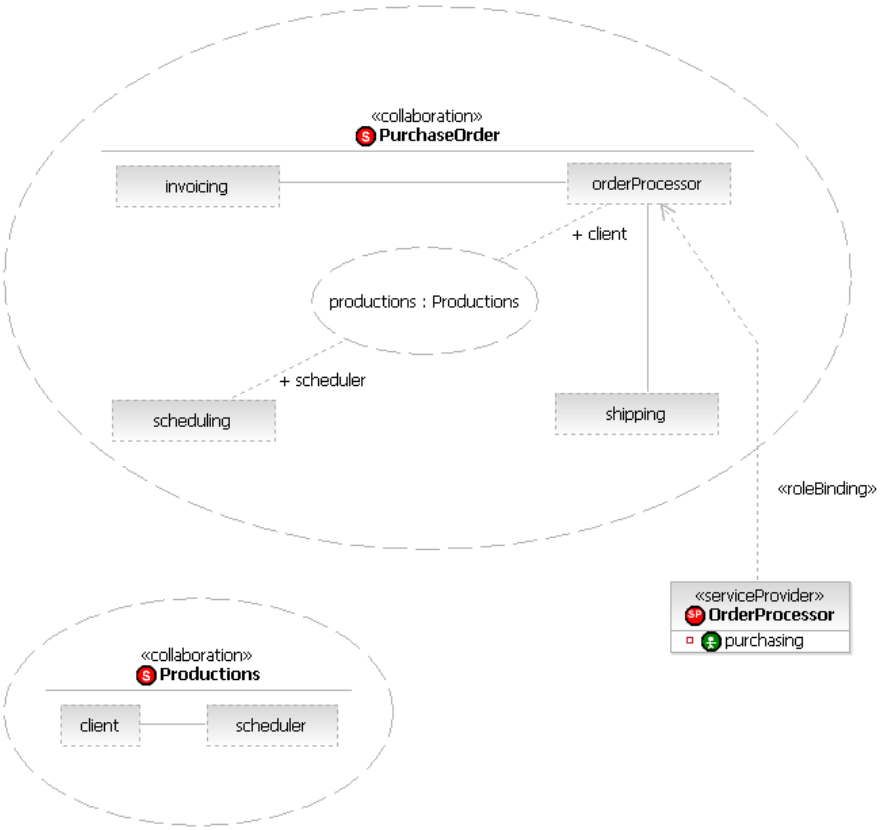


Figure 2.3: UML profile for PIM4SOA: service modelling

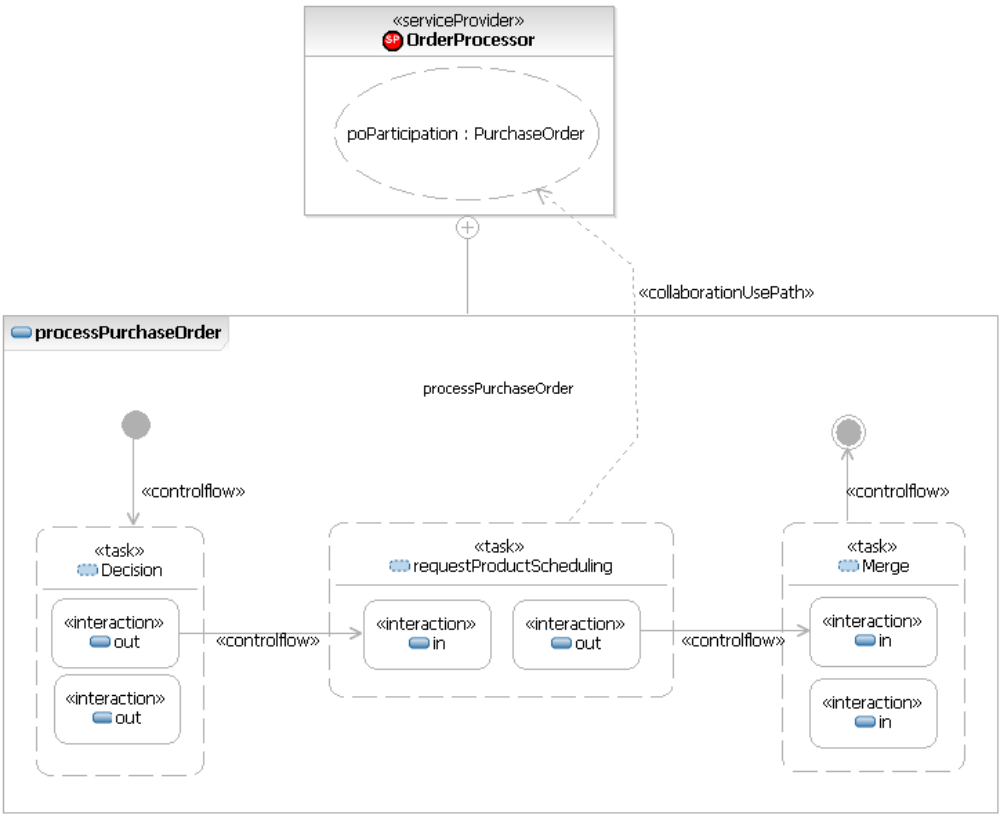


Figure 2.4: UML profile for PIM4SOA: process modelling

| *Stereotype* | *Metaclass* | *Description* |
|---|---|---|
| Collaboration | Collaboration | This element represents the definition of a service. Each service description is viewed as a collaboration between roles. |
| CollaborationUse | CollaborationUse | A collaboration use represents the usage of a service. This service must to be defined previously. CollaborationUse specification must be encapsulated in a composition structure. This structure is provided by a collaboration definition or by a service provider. |
| RoleBinding | Dependency | RoleBinding is used to connect two roles in the structural part of a class or collaboration. |
| ServiceProvider | Class | This element represent the service provider. |
| Role | Role | A role represents a structural part in a collaboration. |
| Process | Activity | A process represents a behavioural description. This behaviour is contained by a service provider or collaboration. |
| Task | StructuredActivity Node | A structured task is related with a collaboration use. This element contains a set of interactions. |
| Interaction | CallBehaviorAction | An interaction defines an interface for input or output flows. |
| Flow | ControlFlow | This flow relates two interactions. |
| CollaborationUse Path | Dependency | This dependency relates a task (source) with a collaboration use (target). |
| Message | Operation | The message concept is related with a role through an interface. |

Table 2.1: UML profile definition for PIM4SOA

is used by both camps. However, business processes often include manual activities and can be related to every kind of resource.

A variety of languages for describing workflows and process executions like Web Services Business Process Execution Language (WS-BPEL) [129, 205], Business Process Modeling Language (BPML) [14], Web Services Flow Language (WSFL) [167], XLANG [286], Web Service Choreography Interface (WCSI) [303], Yet Another Workflow Language (YAWL) [293], or XML Process Definition Language (XPDL) [319] has been developed. Until now several different approaches became apparent, where WS-BPEL and XPDL are the most promising representatives. WS-BPEL workflows are defined in a block-structured way similar to programs written in programming languages like Java or C. XPDL allows users to specify workflows whose control flows represent arbitrary graphs. As we will see in Section 4.2 this difference has important implications on the generation of executable workflow code from higher level descriptions.

Other languages like Business Process Modeling Notation (BPMN) [226], UML 2.0 Activity Diagrams, BPML, or Event-driven Process Chains (EPCs) are used to represent processes at higher level of abstraction. The BPMN has turned out to be a notation that can be easily used by business people. UML activity diagrams are often favoured by technical oriented people. With the Business Process Definition Metamodel (BPDM) [225] the OMG has developed a common metamodel for process modelling that includes also mappings from BPMN to BPDM and mappings from BPDM to BPEL4WS. Mappings to the Web Services Choreography Description Language (WS–CDL) [313] will be incorporated in a future version of the BPDM specification. Concepts represented in UML activity diagrams, BPMN and Enterprise Distributed Object Computing (EDOC)

[220] have also been incorporated into BPDM.

### 2.2.1 Orchestration & Choreography

Orchestration and choreography describe two complimentary notions of a process. In orchestration a central entity coordinates the execution of services involved in a higher-level business process. Only the coordinator of the orchestration is aware of this composition. He defines sequences of activities that are carried out with branching and synchronization of different threads (see Figure 2.5 [148]).

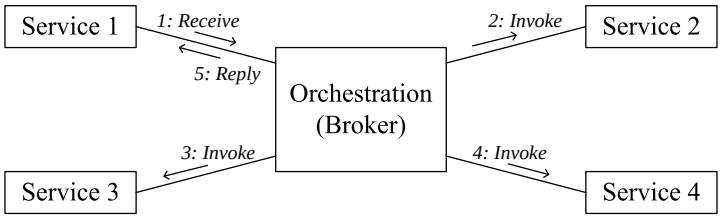


Figure 2.5: Orchestration

Choreography describes the interactions of collaborating entities (e.g. services), each of which may have their own internal orchestration processes (see Figure 2.6 [148]). These interactions are often structured into interaction protocols to represent the conversation between the parties. These protocols usually exist between both internal organisational roles and external stakeholders such as other departments, business units, as well as customers, suppliers and regulatory authorities [225]. An important distinction between orchestration and choreography is the fact that orchestration is generally owned and operated by a single organisation while in a choreography no organisation necessarily controls the collaboration logic [85].

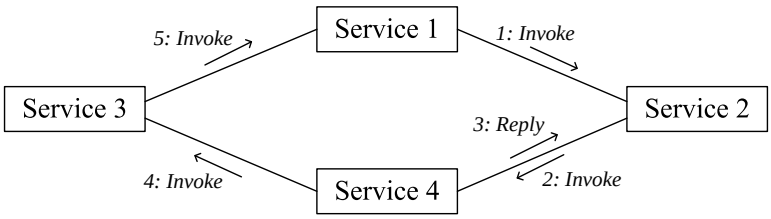


Figure 2.6: Choreography

### 2.2.2 Process Modelling

In process modelling it is common to distinguish between an internal and an external view of business processes. Depending on the viewpoint, a process is described either as an executable, abstract, or collaborative process (see Figure 2.7). [249] distinguishes between these process modelling patterns, which serve as process modelling primitives from which process models are combined.

- *Executable Process*: The internal view models the 'how' of a business process from the modeler's view. Processes that model process flows as a set of partially ordered tasks, are called executable processes [129]. As the flow of an executable

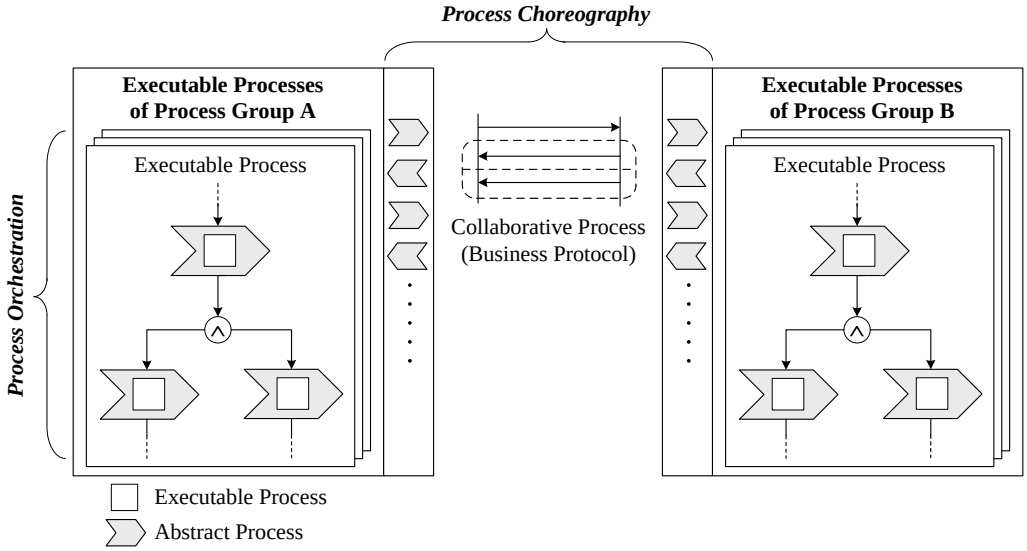


Figure 2.7: Executable, abstract, and collaborative processes

process is described from the viewpoint of a single process coordinating its sub-processes, this is often referred to as process orchestration.

- *Abstract Process*: The external view models the 'what' of a business process. Each process specifies its roles in the collaboration with other processes, but hides the way it is realized. The interfaces of such business processes components are called abstract processes describing the public interactions they perform in relation to their roles in collaborations.

- *Collaborative Process*: A collaborative process describes the collaboration between abstract processes in the case of process choreography. Collaborative processes use abstract processes to model the sequence of the message exchange from the viewpoint of an external observer. The collaborations between the involved parties are modelled as interaction patterns between their roles.

### 2.2.3 Cross-organisational Business Processes

In order to coordinate inter-organisational workflow, Liu and Shen introduced the concept of views, as they are used in database systems, to provide abstract information about internal processes. In [171] they extend their work to Cross-organisational Business Processes (CBPs). Chiu et al. introduce workflow views to control visibility of internal processes and to enable interoperability of e-services, focusing on combining views of different partners to composite e-services CBPs. Schulz et al. use the concept of views, and formalize the dependencies between private processes, process views and CBPs [263]. Adopting the general approach of [263], we distinguish between Private Processes (PPs), View Processes (VPs), and CBPs (according to [170]). Note, the elementary primitives for process modelling are still executable, abstract, and collaborative processes as described in Section 2.2.2. The distinction between PPs, VPs, and CBPs can be used as an additional categorization in process modelling. In a process model, a PP for example is still represented by an arbitrary elementary process modelling primitive

like executable process or a combination of the process modelling primitives introduced in Section 2.2.2.

- **Private Processes (PPs)** are internal to an organisation. They contain data not to be revealed by default. Views on processes provide an abstraction of PPs, which is sufficient to coordinate internal actions with activities of external business partner(s) [263]. A particular interaction may require involved partners to adapt for the purpose of the communication. This adaptation may not necessarily be reflected in the partners' private (internal) business processes without impairing their ability to interact with other partners in a different context.

- **View Processes (VPs)** combine PPs to an abstract level that enables companies to hide critical information from unauthorized partners. The VP connects the PP with the abstract process an organisation provides to a CBP. Based on one PP, different views can be generated, which reflect the specific requirements of different interactions.

- **Cross-organisational Business Processes (CBPs)** define the interactions between two or more business entities. These interactions take place between the defined abstract processes and are defined as a sequence of message and/or other material input/output exchange. Using different views of the same internal processes, organisations are able to interact in a different context without changing the internal process.

To represent these processes in service-oriented models, the PIM4SOA has been extended to model CBPs. Figure 2.8 depicts the extensions of the PIM4SOA metamodel (see Section 2.1) to allow the modelling of PPs that participate in various collaborations through their VPs.

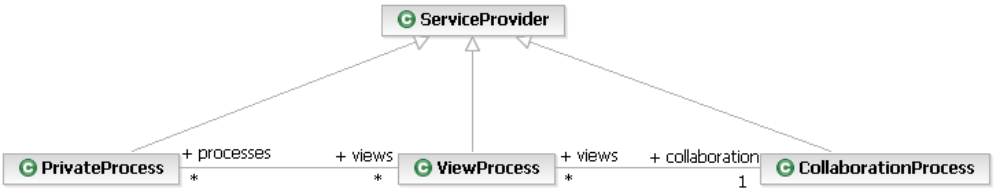


Figure 2.8: PIM4SOA metamodel CBP-extension: Providers

Private processes, view processes and collaboration processes are service providers. A private process is an executable service provider who references view processes. These view processes enact the participation of the private process in external collaborations. Its behaviour is modelled by an executable process. A view process is an executable service provider whose behaviour is a process flow model that may include view tasks. A view task is an activity that abstracts a set of activities of the realizing private processes into a single task. A view process realizes roles in a single collaboration; view tasks are visible in the collaboration (see Figure 2.9). A collaboration process is an abstract service provider whose behaviour is a process flow model. The collaboration process may specify the view processes that together enact the collaboration.

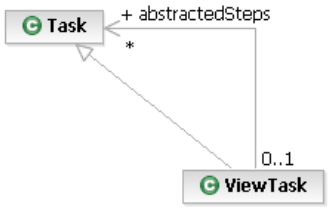


Figure 2.9: PIM4SOA metamodel CBP-extension: ViewTasks

Note, a view process is an executable process that realizes several abstract processes - one for the collaborations it participates in and the others to participate in the implicit collaborations with the private processes it supports. A view process connects the abstract process an organisation provides to a CBP to the private processes of the organisation. Nevertheless a view process is an executable process. Section 4.1.4 will provide more detailed examples for this mechanism.

## 2.3 Architecture Evaluation and Decision Methods

This section introduces architecture evaluation and decision methods, which are further relevant for this thesis.

### 2.3.1 Architecture Evaluation

Scenario-based ICT architecture evaluation is used to determine quality of software architecture. In architecture evaluation methods like ATAM, SAAM, or ARID [21, 57] quality attributes are characterized by scenario descriptions.

Quality attributes are part of the non-functional requirements and therefore properties of a system. They can be broadly grouped into two categories [71]. Qualities like performance, security, availability, and usability are observable via execution (i.e. at *runtime*) and qualities like extensibility, modifiability, portability, reusability, etc. which are not observable via execution but at *build-time* [31].

According to Bass et al. [21], scenario descriptions (see Figure 2.10 [21, p.75]) consist of a *stimulus* (a condition that needs to be considered when it arrives at a system), a *source of stimulus* (some entity that generates the stimulus), an *environment* (the stimulus occurs within certain conditions), an *artifact* (the part of the system that is stimulated), a *response* (the response is the activity undertaken after arrival of the stimulus) and a *response measure* (defines how the result of the response is measured). General scenarios [22] are applicable to many software systems and have architectural implications; they establish sets of scenarios which are configured to the respective application domain (for which evaluation is performed) by varying the expected response value scales of the scenarios.

**Tactics** To be able to decide how well a quality attribute or a scenario is supported by a software architecture pattern and to compare architecture patterns, it is crucial to understand how an architecture influences quality attributes. According to Bass et al.

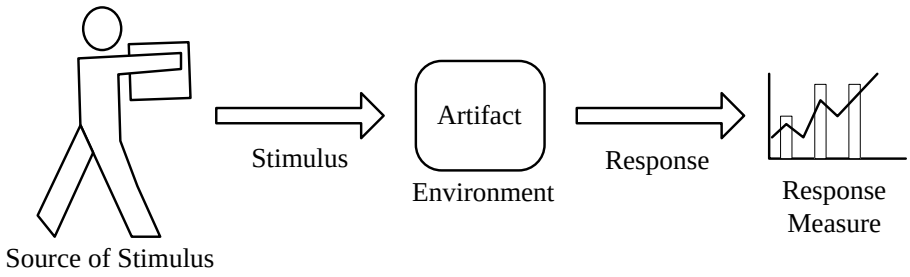


Figure 2.10: Quality Attribute

[21] architects use so-called tactics to achieve quality attributes. A tactic is a design decision that influences the control of a quality attribute. The software architecture patterns described in this thesis make use of the following tactics (non-exclusive list; for detailed description see also [21, p.99ff]): *Maintain semantic coherence*, *anticipate expected changes*, *generalize module*, *restrict communication paths*, *use an intermediary*, *maintain existing interfaces*, and *hide information*.

- *Maintain semantic coherence:* The goal of this tactic is to ensure that the responsibilities among modules work together without excessive reliance on other modules. Patterns and principles for realizing this tactic are abstraction, loose coupling, and orchestration.
- *Anticipate expected changes:* This tactic tries to limit the set of modules that need to be modified in case of certain changes. In contrast to the semantic coherence strategy anticipating expected changes does not concern itself with the coherence of a module's responsibilities but rather with minimizing the effects of the changes. Patterns and principles for realizing this tactic are wrapper, broker, abstraction, loose coupling, and orchestration.
- *Generalize module:* Making a module more general allows it to compute a broader range of functions based on input. The more general a module is, the more likely it is that changes can be made by adjusting the input rather than by modifying the module. Patterns and principles for realizing this tactic are abstraction and orchestration.
- *Restrict communication paths:* This tactic tries to limit modifications to the localized modules by reducing the number of modules a given module shares data with. Patterns and principles for realizing this tactic are broker, loose coupling, and orchestration.
- *Use an intermediary:* Intermediators manage the activities associated with dependencies between modules. A bridge, mediator, etc. pattern can for example convert the syntax of service from one form into another. A broker pattern can be used to mask changes in the identity of interfaces and modules respectively.
- *Maintain existing interfaces (separate interface from implementation):* Maintaining the name and signature of a module's interface, allows other modules using this interface to remain unchanged (this tactic works well at least for syntactic changes). Patterns and principles for realizing this tactic are wrapper and broker.

- *Hide information:* Information hiding is the decomposition of the responsibilities for an entity into smaller pieces and choosing which information to make private and which to make public. Its goal is to isolate data and logic within one module and prevent changes from propagating to other modules. Patterns and principles for realizing this tactic are abstraction and orchestration.

**Architectural Patterns and Principles** Tactics are used by an architect to create a design using design patterns, architectural patterns, or architectural strategies. An architect usually chooses a pattern or a collection of patterns designed to realize one or more tactics. However, each pattern implements multiple tactics, whether desired or not. The following list provides an overview of architecture patterns, design patterns, and design principles used to realize the above described tactics (non-exclusive list compiled from [21], [84], [85], and [101]): *Wrapper*, *broker*, *abstraction*, *loose coupling*, and *orchestration*.

- *Wrapper:* Wrappers encapsulate components like legacy environments and expose (legacy) functionality of these components. Wrappers are often utilized for integration purposes and are a frequently used form of adapters.
- *Broker:* A broker consists of software used for mediation between components. It can be used for integration enabling communication between components requiring different data formats and it can be used to mask (changes in) the identity of interfaces (e.g. by forwarding messages).
- *Abstraction:* Abstraction is a principle to reduce and factor out details so that one can focus on a few concepts. It allows components to act as black boxes hiding their details from the outside world. It can be used for both encapsulating potentially complex processing logic and abstracting from data structures.
- *Loose coupling:* Loose coupling is a principle that aims to decrease coupling and increase independence of components. A component that acquires knowledge of another component still remains independent of that component.
- *Orchestration:* Orchestration describes the automated arrangement and coordination of services and fosters the separation of computation from coordination. Process logic encapsulated by an orchestration can be modified or extended in a central location while still remaining extensible. Orchestration is also a good way to provide composed services.

### 2.3.2 Analytic Hierarchy Process

The Analytic Hierarchy Process (AHP) [256] is a decision making approach, which decomposes a decision problem into a hierarchical network of factors and subfactors. Factor decomposition establishes a hierarchy of first level and second level factors cascading from the decision objective or goal. AHP applies pairwise comparisons to the factors and the alternatives in the decision making process. Pairwise comparisons lend themselves to solving problems with limited number of choices, where each choice has a number of attributes and it is difficult to formalize some of those attributes. Finally the ratings of the second level factors are aggregated to first level factors and the final rating. We illustrate the AHP via an example for *'choosing the best house to buy'* (cp. [257]):

1. In a first step, we construct a hierarchy that represents the decision problem (see Figure 2.11). The overall objective *'satisfaction with house'* is located at the top of the hierarchy, the criteria and alternatives are placed at each descending level of the hierarchy.

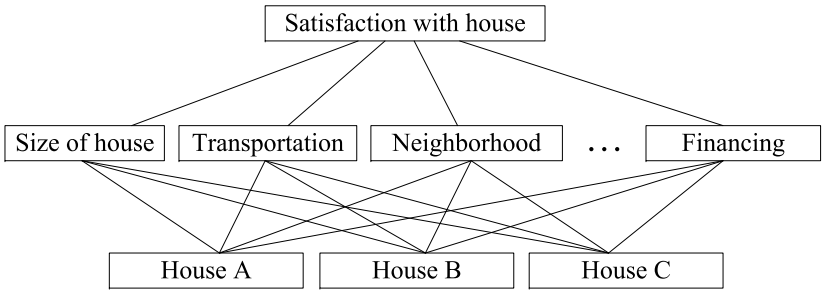


Figure 2.11: AHP example: decomposition tree

2. To apply the principle of comparative judgment, one has to set up a comparison matrix at each level by comparing pairs of criteria, or pairs of alternatives at the lowest level. Table 2.2 depicts the comparison matrix for the criteria *'size of house'*. Since house A is bigger than house B and house C, it gets the higher comparison values (6 and 8).

| Size of house | House A | House B | House C | Priority vector |
|---|---|---|---|---|
| House A | 1 | 6 | 8 | 0.754 |
| House B | $\frac{1}{6}$ | 1 | 4 | 0.181 |
| House C | $\frac{1}{8}$ | $\frac{1}{4}$ | 1 | 0.065 |

Table 2.2: AHP example: rating size of house

3. The last step determines the final rating, which is given through the composite or global priorities of the houses. One can find the local priorities of the houses with respect to each criterion in the matrix depicted in Table 2.3. We multiply each column with the priority of the corresponding criterion and add across each row the results of the multiplication. House A has the highest global priority and is probably the house that is bought.

| | Size of house | Transportation | Neighborhood | ... | Financing | |
|---|---|---|---|---|---|---|
| | 0.173 | 0.054 | 0.188 | ... | 0.333 | |
| House A | 0.754 | 0.233 | 0.754 | ... | 0.072 | 0.396 |
| House B | 0.181 | 0.055 | 0.065 | ... | 0.650 | 0.341 |
| House C | 0.065 | 0.713 | 0.181 | ... | 0.278 | 0.263 |

Table 2.3: AHP example: local and global priorities

### 2.3.3 Contingency Theory

The contingency theory for organisations [72] is used to rationalize how the various aspects of organisations' environment (called *contingency factors*) influence organisation structure. It suggests, that there is no unique or best way to organise an organisation, but the design of an organisation and its systems must 'fit' with its environment. The

*"organizational effectiveness results from the fitting characteristics of the organization, such as its structure, to contingencies that reflect the situation of the organization"* [72, p.1]. *"Contingency theory (...) sees maximum performance as resulting from adopting, not the maximum, but rather the appropriate level of the structural variable that fits the contingency. Therefore, the optimal structural level is seldom the maximum, and which level is optimal is dependent upon the level of the contingency variable"* [72, p.4]. In other words, an organisational structure (e.g. hierarchical, organic, bureaucratic or functional) which fits the contingency factors, such as size of the organisation, environment or organisational strategy, is more effective with regard to efficiency, profitability or innovation rate. [72] distinguishes contingencies within (e.g. task uncertainty, task interdependence) and outside (e.g. environmental uncertainty) of the organisation. If the contingency, its value and its influence on the organisational effectiveness is known to the organisation, the organisation is able to adapt its organisational design (e.g. its structure) and thereby increases its performance. On the other hand, if the organisational structure misfits the contingency, a negative impact on the performance will be the result. [165]

Translating this into the terms of companies and their business systems, a maximum of centralization, decentralization, or some of the ICT system architectural qualities like modifiability, security, etc., will seldom yield maximum performance of an ICT system for the overall business goals. An appropriate level of performance, where an ICT system best supports the overall business goals, can be reached, if the ICT system fits the internal and external contingencies.

## 2.4 Semantic Technologies and Technological Spaces

Semantics is a very important aspect of IT, since it helps the receiver of data, e.g. computers or people, to interpret the data correctly in the way that was intended by the sender of the data. In the recent years a new field of research has emerged that promises to improve current IT solution by a computer-understandable semantics: the Semantic Web [32, 33]. This section introduces the concept of Technological Space and distinguishes model-based from Semantic Web technology. It further discusses the notion of ontology in the IT and gives a definition of syntax, semantics, and ontology that is used throughout this document.

### 2.4.1 Technological Spaces

Kurtev et al. [160] introduce the concept of Technological Spaces (TSs) aiming to improve efficiency of work by using the best possibilities of different technologies. A technological space is in short a zone of established expertise and ongoing research. It is a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities. Initially five technological space, the MDA TS, the XML TS, the Abstract Syntax TS, the Ontology TS, and the DBMS TS, have been presented in [160], of which the MDA TS and the Ontology TS are important for our work. The ontology engineering space performs outstanding in traceability, i.e. in the specification of correspondences between various metamodels, while the MDA TS is much more applicable to facilitate aspects or content separation. The XML TS is based on the XML format, where documents are written in a syntax constrained by well-formedness and validity constraints. The Abstract Syntax TS has its foundations in context-free grammars for specification of language syntax and a number of formalisms for specification

of language semantics (attribute grammars, denotational and action semantics, etc.). The DBMS TS is defined on the basic concepts of data and schema.

### MDA TS

In the MDA TS models are considered as first-class citizens, representing particular views on the system being built. A model is an artifact that conforms to a metamodel and that represents a given aspect of a system. These relations of conformance and representation are central to model engineering. A model is composed of model elements and conforms to a unique metamodel. This metamodel describes the various kinds of contained model elements and the way they are arranged, related and constrained. A language intended to define metamodels and models is called a metametamodel [39]. In the Modelware and Model-based Technology TS (MDA TS) models and their metamodels are commonly arranged in four-level metamodelling hierarchy. More information about MDSD and the MDA TS can be found in Chapter 3.

### Ontology TS

The Ontologyware and Ontology Engineering TS (Ontology TS) can be considered as a subfield of knowledge engineering, mainly dealing with representation and reasoning. The Semantic Web [32, 33] is one way to combine techniques and standards in the Ontology TS. In general, a set of languages is used for knowledge representation. These languages correspond to formalisms (normally logic), that allow to perform reasoning on the data expressed with these languages.

In the Semantic Web languages are based on Extensible Markup Language (XML) [314], which is used as standardized syntactical representation and uniform way to structure data. Uniform Resource Identifiers (URIs) [134] are used to identify abstract and physical resources. XML does not itself imply specific interpretation of data. Other languages, like the Resource Description Framework (RDF) [308, 310] or RDF Schema (RDF(S)) [309] provide the models, that allow machine interpretation of data [111]. RDF gives XML structured data meaning by representing data as triples (statements). A triple consists of a subject (a resource, identified by an URI), a predicate (the property of the subject), and an object (the value of the property) [307]. The value can be a literal or another resource. However, RDF itself is only used to describe instances of ontologies [105]. RDF(S) extends RDF with means to enable the creation of taxonomies and ontologies. RDF(S) provides primitives such as *Class*, *subClassOf*, and *subPropertyOf*. The Web Ontology Language (OWL) [306] is used as a standard language to describe ontologies in the Semantic Web. OWL is a combination of increasingly expressive sublanguages built on top of each other. *OWL Lite* is intended to support the building of simple classification hierarchies and simple constraints. *OWL DL* provides more expressiveness, but also guarantees that all conclusions are computable and reasoning terminates in a finite time. OWL DL can be classified as a $\mathcal{SHOIN}(\mathbf{D})$[5] language [126]. *OWL Full* does provide maximum expressiveness, but does not guarantee computational completeness and decidability. The Semantic Web uses further languages and

[5] $\mathcal{SHOIN}(\mathbf{D})$ classifies the description logic expressivity of OWL DL. $\mathcal{S}$ comprises atomic negation, concept intersection, universal restrictions, limited existential quantification, and complex concept negation. $\mathcal{H}$ stands for role hierarchy, $\mathcal{O}$ for nominals, $\mathcal{I}$ for inverse properties, and $\mathcal{N}$ for cardinality restrictions. (**D**) represents the use of datatype properties, data values, or data types.

techniques to address issues like query (SPARQL [315]), rules (RIF [300]), unifying logic, proof, trust, and crypto [66, 68].

With the Ontology Definition Metamodel (ODM) [234] the OMG issues a specification defining a family of independent metamodels, related profiles, and mappings among the metamodels corresponding to several international standards for ontology definition, as well as capabilities supporting conventional modelling paradigms for capturing conceptual knowledge. It is based on a grounding in formal logic, through standards-based, model-theoretic semantics, sufficient to enable reasoning engines to understand, validate, and apply ontologies developed using ODM. ODM includes a set of metamodels which are grouped logically together according to the nature of the representation formalism that each represents; ODM comprises metamodels for RDF(S), OWL, Common Logic (CL), Topic Maps (TM), and as a non normative part Description Logic (DL). Metamodels for RDF(S) and OWL represent more structural or descriptive representations that are commonly used in the Semantic Web community. ODM further defines transformations between the UML 2 metamodel and different metamodels defined in ODM (e.g. OWL and RDF(S)).

According to [37], the notion of a metamodel is strongly related to the notion of ontology, since a metamodel is a formal specification of an abstraction, usually consensual and normative. Atkinson and Kühne [17] provide a framework that clarifies further the relationship between metamodel and ontologies. They distinguish between the orthogonal dimensions of linguistic *instanceOf* relationships, as they are used in e.g. a four-level metamodelling hierarchy, and ontological *instanceOf* relationships, which are used e.g. for the definition of the user's domain.

### 2.4.2 Ontology

In the field of computer science, ontologies have attracted more and more attention during the last years. The word *"Ontology"* comes from the Greek *ontos*, for "being", and logos, for "word". In Philosophy, *ontology* is concerned with the fundamental questions of what exists. Computer scientists use this idea to enable automated knowledge sharing by the use of ontologies.

Gruber [118] defines an ontology as an *"explicit specification of conceptualization"*. He aims to improve the knowledge exchange between different software systems. If the knowledge base of an intelligent system is to represent (parts of) the world for some purpose, then it must be committed to some conceptualization. Conceptualization stands for an abstract, simplified, view of the world, i.e. a model. An ontology is an explicit and formal description of this model and conceptualization, respectively. Certain concepts of the real world (the "universe of discourse") are linked with formal elements like classes, relations or functions. The formal representation also implies that an ontology should be machine-readable (for more information see [105]).

According to [207] an ontology differs from existing methods and technologies in the following way: (i) the primary goal of ontologies is to enable agreement on the meaning of specific vocabulary terms and, thus, to facilitate information integration across individual languages; (ii) ontologies are formalized in logic-based representation languages. Their semantic are thus specified in an unambiguous way. (iii) The representation languages come with executable calculi enabling querying and reasoning at run time.

Gruber describes criteria [118, p.2ff], which 'good' ontology should meet. First of

all an ontology should communicate the meaning of the defined terms effectively and thus provide *clarity*. The defining axioms should be logically consistent and, hence, contain no contradictions (*coherence*). Due to the frequent modification in knowledge systems, *extensibility* is a crucial feature. For compatibility issues, a *minimal encoding bias* is essential. Specific encoding of a value (e.g. a number as float or decimal with a certain number of digits) should not lead to a "biased" representation of knowledge. An ontology should make as few claims about the world as possible, i.e. a *minimal ontological commitment* is desirable. Finally, suitability for the specific field of application is an important issue for ontology design.

There exist a number of terms that are used to classify ontologies according to their usage (for more information see [207]). *Application ontologies* contain the definitions specific to a particular application [119], while *reference ontologies* refer to ontological theories whose focus is to clarify the intended meaning of terms used in specific domains. *Leightweight ontologies* only consist of concepts and their relations, while *heavyweight ontologies* contain many axioms, additional conditions and restrictions to allow automated checking. Another classification refers to the specificity [120]: *Generic* or *top-level ontologies* define very basic facts like states, processes or components, whereas domain ontologies and application ontologies contain conceptualizations, that are confined to a certain universe of discourse [119]. In the intermediate range *core ontologies*, which refer to several domains. The terms *upper ontology* or *foundation ontology* both have the same characteristics as *top-level ontologies*. [47] provides two more classification criteria for ontological analysis: *Descriptive ontologies* are based on the human perception of the world, even if it is not fully scientific and correct. *Revisionary ontologies* tend to avoid ambiguous concepts or unscientific use of language. Of course, they are less intuitively understandable. A further distinction can be made between multiplicative and reductionist ontologies: *Multiplicative ontologies* allow overlapping of several entities, i.e. light can be classified as wave and as particle, which induces a bigger number of basic concepts. *Reductionist ontologies* dissolve such contradictions by introducing different concepts and allow only one object for each point in the space time continuum.

**Ontologies for Interoperability Solutions** Ontologies are considered a key element for semantic interoperability and act as shared vocabularies for describing the relevant notions of a certain application area, whose semantics is specified in a (reasonably) unambiguous and machine-processable form [47].

The Interoperability Development for Enterprise Application and Software (IDEAS) network stated in its vision for 2010 [132] requirements to enable enterprises to seamlessly interoperate with others. According to this, it is necessary to integrate and adapt ontologies in architectures and infrastructures to the layers of enterprise architecture and to operational models. As enterprises often apply different methodologies, they need to share their enterprise models and knowledge independent of languages and tools. Therefore, one needs to develop mappings between different existing enterprise modelling formalisms based on an enterprise modelling ontology as well as tools and services for translating models (IDEAS analysis - gap 12 [131]).

[83] proposes a rather abstract interoperability framework for MDSD of software systems, which supports the business interoperability needs of an enterprise. Mutual understanding on all levels of integration, conceptual, technical, and applicative level, has to be achieved. One uses the conceptual reference model to address model interop-

erability, whereas metamodels and ontologies are used to define model transformations and model mappings between the different views of an enterprise system.

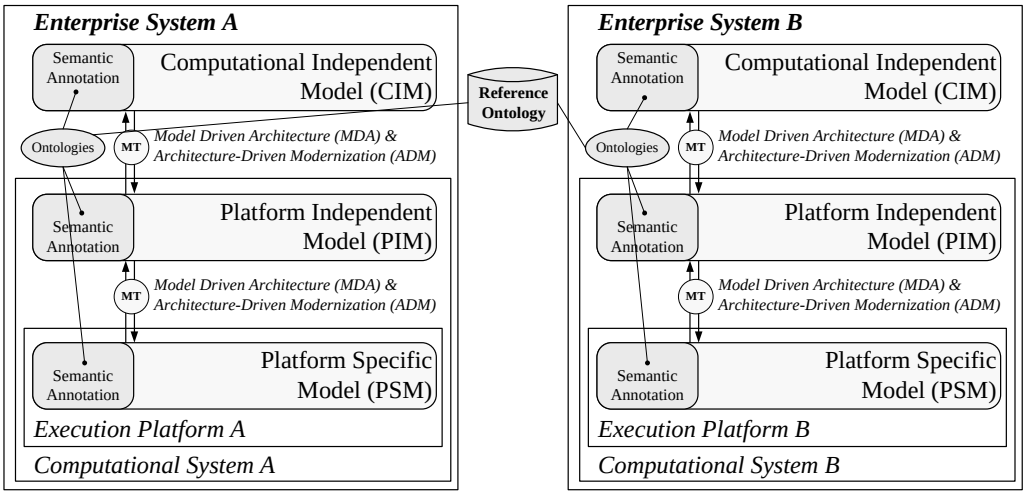


Figure 2.12: Reference model for conceptual integration

In the reference model for conceptual integration (see Figure 2.12) the models at the various abstraction levels (CIM, PIM, and PSM) are semantically annotated using ontologies. This helps to achieve a mutual understanding on all abstraction levels. MDA and the ADM [239] are *top-down* and *bottom-up* approaches to software development and integration. The use of a reference ontology aids to perform model transformations and mappings between and across the three model levels. [83] also proposes the usage of interoperability patterns for horizontal and vertical integration.

### 2.4.3 Syntax, Semantics, and Ontology

The notion of the term semantics differs in the context it is used and by the people using it. As the root of the problem Harel and Rumpe [124] identify insufficient regard for the crucial distinction between syntax and true semantics. Thus we clarify a few terms that have particular significance to this thesis.

- **Syntax:** *Syntax* $N_L$ is the notation of a language $L$. A distinction is made between the concrete syntax, the textual or graphical representation of the language, and an abstract syntax or metamodel, being the machine's internal representation. A metamodel is a way to describe a language's syntax [124].

- **Semantic:** Semantic is the meaning of language, which is expressed by relating the syntax to a semantic domain. The description of a *semantic domain* $S$ (its notation is $N_S$) can vary from plain English to mathematics. Semantics is defined by a semantic mapping $M : L \rightarrow S$ from the language's syntax to its semantic domain [124, 327].

- **Ontological:** According to [230] an ontology defines the common terms and concepts (meaning) used to describe and represent and area of knowledge. Talking about 'ontological' we mean technology of the Ontology TS, i.e. technology based on logics like RDF(S) or OWL used by the semantic web community to describe

e.g. vocabularies or ontologies. Instead of semantic web enabled we also use the term *semantic-enabled* as synonym in this thesis.

## 2.5 Summary

In this chapter we have introduced technologies that are further important in this thesis. The concepts and definitions presented for service-orientation, process modelling, process execution, semantic technologies, and Technological Spaces build a foundation on the basis of which we develop our solutions. It is also important to understand the architecture evaluation and the decision methods described in Section 2.3, since we apply a combination of these methods to the new application area of ICT architecture evaluation. In the next section we introduce the basics of MDSD which is the main topic of this thesis.

# Chapter 3

# Model Driven Software Development

In software engineering one can currently observe a paradigm shift from object-orientation towards models, that may have important consequences on the way information systems are built and maintained [37]. J. Greenfield and K. Short write in [115, p.1]: *"The software industry remains reliant on the craftsmanship of skilled individuals engaged in labor intensive manual tasks. However, growing pressure to reduce cost and time to market and to improve software quality may catalyze a transition to more automated methods. We look at how the software industry may be industrialized, and we describe technologies that might be used to support this vision. We suggest that the current software development paradigm, based on object orientation, may have reached the point of exhaustion, and we propose a model for its successor."* This new global trend is called Model Driven Engineering (MDE). Model Driven Software Development (MDSD) [278], which is sometimes also called Model Driven Software Engineering (MDSE), is the intersection between MDE and software engineering, i.e. it is subset of MDE which is concerned with software production [94].

MDSD is an approach to software development based on modelling and automated transformation of models to implementations [99]. Higher-level models are transformed to more detailed lower-level models. MDSD solutions consist of an arbitrary number of such transformations. Each model may require some human refinement. Finally, code is generated from lower-level models. In MDSD models are more than abstract descriptions of systems, as they are used for model and code generation – they are the key part of the definition of a software system. Largely automated model transformations refine abstract models to more concrete models (vertical model transformations[1]) or simply describe mappings between models of the same level of abstraction (horizontal model transformations). As model transformations play a key role in MDSD, it is important that transformations can be developed as efficiently as possible [103]. With the MOF 2.0 Query, Views, and Transformation (QVT) specification [233] the OMG provides a standard syntax and execution semantics for transformations used in a MDSD tools chain.

This chapter describes the current state of the art in MDSD and sets the context of

[1] In the case of vertical model transformation knowledge of platforms is encoded into transformations or provided by separate platform models, reused for many systems rather than re-designed for each new system.

this thesis. Section 3.1 describes how software development can benefit from MDE. It introduces approaches, tools, initiatives, and projects that realize MDSD. Section 3.2 and 3.3 provide more details about the key concepts of MDSD. They explain concepts and technologies used to realize and implement models and model transformations.

## 3.1 MDE Approaches to Software Development

The MDE approach pursues multiple objectives: Apply models and model technologies to raise the level of abstraction at which developers create and evolve software [116, 122]; use models as primary artifacts, not just as sketches or blueprints, from which efficient implementations are generated [104]; reduce complexity of software artifacts by separating concerns and aspects of a system under development [37, 122]; domain-oriented models expressed with domain-specific languages are the primary focus when developing new software components [37, 104]; use transformation engines and generators to generate code and other target domain artifacts with input from both modelling experts and domain experts [104, 262].

The ideas behind MDE are not totally new. Since the beginning of the computing discipline, software researchers and developers have been creating abstractions both of language and platform technologies. One well-know effort is Computer-aided Software Engineering (CASE), which focused on developing software methods and tools that enabled developers to express their designs in terms of general purpose graphical programming representations. Except a few domains like telecom call processing, CASE was not widely adopted in practice and had very little impact on commercial software development [262]. One reason was that abstractions provided by CASE focused on the solution space, i.e. they were computing-oriented, rather than on the problem space by expressing designs in terms of concepts in application domains. According to Booch et al. [46], to bridge the semantic gap between domain-specific concepts encountered in modern software applications and standard programming technologies is the greater and more important challenge. Furthermore, CASE tools also did not support many application domains effectively because their 'one-size-fits-all' graphical representations were too generic and non customizable.

Having learned lessons from CASE tools in the 1980s, MDE today combines domain specific languages with transformation engines and generators [262]. Currently there are a variety of approaches and standards which realize MDE like Model Driven Architecture (MDA) [227], Software Factories [116], Agile Model-Driven Development [9], Domain-Oriented Programming [287], or Model Integrated Computing (MIC) [284]. In the following, we present with MDA and Software Factories the two most prominent representatives. We have a detailed look at the MDA approach, since its tool chain based on MOF is of particular importance in this thesis.

### 3.1.1 MDA

Model Driven Development (MDD) and Model Driven Architecture (MDA) [208] are both trademarks of the OMG. MDD can be described as an approach to build systems using models. MDD solutions start with a formalized description of the problem in terms of the problem domain and apply transformations, which lead a model that specifies the solution in terms of the solution domain [217]. MDA is a specific MDD deployment effort around a set of OMG standards like MOF, XMI, UML, CWM, QVT, SPEM, etc.

OMG's initial definitions of MDA [158, 212] described MDA as an approach building on a set of distinctive artifacts: Computation Independent Models (CIMs), Platform Independent Models (PIMs), Platform Models (PMs), Platform Specific Models (PSMs), and code. Most current MDA-based solutions make use of the following three models (see [213, 283]):

- A Computation Independent Model is a view of a system from the computation independent viewpoint. A CIM describes the requirements for a system and the business context in which the system will be used. It specifies what a system will be used for, and does not show details of the structure of the system or how it is implemented. The CIM plays an important role in bridging the gap between the experts of the domain and its requirements, and the experts of the design and construction of the artifacts that together satisfy the domain requirements. It uses a vocabulary that is familiar to the practitioners of the domain and is often expressed in a business or domain specific language.
- A Platform Independent Model is a view of a system from the platform independent viewpoint. It describes how a system is constructed, but without reference to the technologies used to implement the model. A PIM exhibits a specified degree of platform independence and may be implemented by one platform rather than another, or it may be suitable for implementation on many platforms.
- A Platform Specific Model is a view of a system from the platform specific viewpoint. It is a model of a solution from a particular platform perspective. It includes both the details from the PIM that describe how the CIM can be implemented, and the details describing how the implementation is realized on a specific platform.

MDA mappings provide specifications that transform PIMs into PSMs for particular platforms, where the PMs determine the nature of the mappings [212, p.3-2]. The MDA specification [227] gives examples for a variety of platforms like object, batch, dataflow, and embedded for generic platforms or AJAX, CORBA, J2EE, and Spring for technology specific platforms. Finally, code is generated from the PSM in the MDA tool chain.

In MDA models and modelling languages should be described by the means of UML, Meta Object Facility (MOF), and UML profiles. The narrow focus on distinctive model types and modelling languages led to a lot of debates and criticism on MDA [61, 94, 121, 114]. Thus, OMG's current official statement about MDA [216], which was approved in June 2004, was put on a broader basis. It says:

*MDA is an OMG initiative that proposes to define a set of non-proprietary standards that will specify interoperable technologies with which to realize model-driven development with automated transformations. Not all of these technologies will directly concern the transformations involved in MDA.*

*MDA does not necessarily rely on the UML, but, as a specialized kind of MDD (Model Driven Development), MDA necessarily involves the use of model(s) in development, which entails that at least one modelling language must be used. Any modelling language used in MDA must be described in terms of the MOF language, to enable the metadata to be understood in a standard manner, which is a precondition for any ability to perform automated transformations.*

The current definition of MDA includes the development and use of arbitrary domain specific languages beneath UML. Modelling languages must be described in terms

of MOF, which is OMG's standard language to define metadata. This ensures interoperability to exchange models and metamodels between MDA tools and the ability to perform automated transformations. The OMG recognizes also the fact that "there are no magic buttons" [114, p.5] and still design decisions must be made. It considers a mixture of manual and automatic transformation [227].

Such a definition of MDA indeed fits very well the IBM view on MDA described by Booch et al. [46]. They explain the three basic and complementary tenets of MDA to be *direct representation*, *automation* and *standards* (see Figure 3.1 [46]).

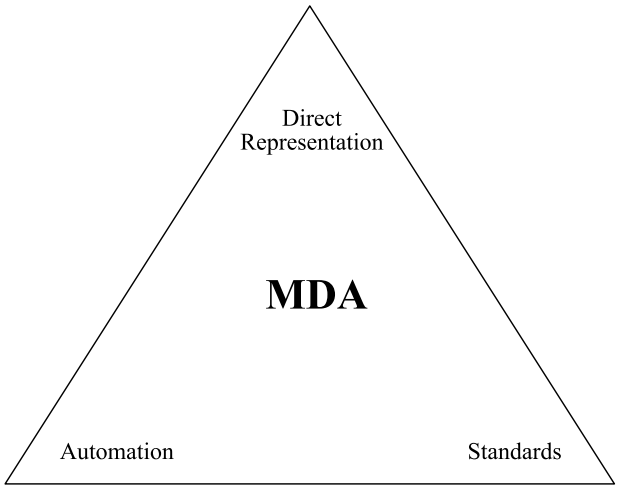


Figure 3.1: Tenets of MDA

- *Direct representation* stands for modelling with languages that map their concepts to domain concepts rather than computer technology concepts. This allows a more direct coupling of solutions to problems, leading to more accurate designs and increased productivity.

- *Automation* aims to increase speed and reduce errors in software development by the automated transformation of domain-specific models into implementation code. Generation tools exploit the domain and technology choices modelled in frameworks and the knowledge built into particular application frameworks.

- *Standards* encourage the emergence of an entire ecosystem of tool vendors addressing many different needs. Open source development ensures that standards are implemented consistently and encourages the adoption of standards by vendors.

Summing up, MDA approaches build models that directly represent domain concepts. DSLs may either be built on top of general languages such as UML or on the metamodelling framework MOF. By using frameworks that explicitly model assumptions about the application domain and the implementation environment, automated tools can transform models into implementations. [46]

### 3.1.2 Software Factories

Software factories are an initiative from Microsoft[2] towards MDE. They promote the use of DSLs, which are software development languages that have been developed for particular problem domains.

*"A software factory is a software product line that configures extensible tools, processes, and content using a software factory template based on a software factory schema to automate the development and maintenance of variants of an archetypical product by adapting, assembling, and configuring framework-based components."* [116]

Software factory schemas define viewpoints, i.e. the kinds of models and the tools used to build them, and computable relationships between viewpoints. Viewpoints describe the artifacts that must be developed to produce a software; these are DSLs used to build artifacts, refactorings that can improve artifacts, or mappings that support transformations between artifacts. Software factory templates implement software factory schemas by defining the DSLs, patterns, frameworks, and tools the software factory schemas describe, packaging them, and making them available to product developers.

Like MDA the software factory approach advocates the direct representation and the automation paradigm by working with application domain concepts and introducing automation into the software life cycle. Both approaches emphasize the importance of visual modelling and capturing expertise through patterns. The main difference between MDA and the software factory approach is the emphasis that is put on open standards, particularly UML and MOF [104].

### 3.1.3 Benefits of Model Driven Engineering

When applying MDE, i.e. the MDA, the software factory, or any other approach, it is crucial to use only those abstractions (models), transformations, frameworks, and tools that are appropriate for the problem and the solution domain. Abstractions and transformations should justify the effort that is necessary to realize them in MDE applications by yielding significant added value for the achievement of the overall objectives. Respecting these aspects, MDE has the potential to greatly improve current mainstream software development practices. The following list, which based on [283], gives an overview of benefits of MDE.

- *Increased productivity*: MDE reduces the costs of software development by generating code and artifacts from models, which increases developer productivity.
- *Maintainability*: MDE facilitates maintainable architectures where changes are made rapidly and consistently, enabling more efficient migration of components onto new technologies. This makes it easier to handle changes in the underlying platform technology and its technical architecture.
- *Reuse of legacy*: Reverse transformations and engineering allows to derive models from components implemented on legacy platforms. These components are brought to a higher level of abstraction. Then one has the option of migrating the components to a new platform or generating wrappers to enable the legacy component to be accessed via integration technologies.

[2] `http://www.microsoft.com`

- *Adaptability*: Adaptability is a key requirement for businesses, and IT systems need to be able to support it. When using a MDE approach, adding or modifying a functionality is quite straight forward since the investment in automation was already made.
- *Consistency*: Applying manually coding practices and architectural decisions is an error prone activity. MDE ensures that artifacts are generated consistently.
- *Repeatability*: The return on investment of MDE from developing the transformations increases each time they are reused. The use of tested transformations increases the predictability of developing new functions and reduces the risk since the architectural and technical issues were already resolved.
- *Improved stakeholder communication*: Models omit implementation detail that is not relevant to understand the logical behaviour of a system. Models are therefore much closer to the problem domain, reducing the semantic gap between the concepts that are understood by stakeholders and the language in which the solution is expressed.
- *Improved design communication*: Models facilitate understanding and reasoning about systems at the design level. This leads to improved discussion and communication about a system.
- *Expertise capture*: Projects or organisations often depend on the knowledge of experts who repeatedly make best practice decisions. The knowledge of an organisation is maintained in models when experts leave the organisation. The application of best practices in general improves the quality of solutions.
- *Models as long-term assets*: In MDE, models are important assets that capture what the IT systems of an organisation do. High-level models are resilient to changes at the state-of-the-art platform level. They change only when business requirements change.
- *Ability to delay technology decisions*: In the MDE approach, early application development is focused on modelling activities. The choice of a specific technology platform or product version can be delayed until a later point when further information is available.

However, there exist several obstacles to realize these benefits of MDE. There is a lack of proven solutions and reusable MDE assets for most application domains. Still, model transformations that bridge the gap between domain-specific concepts and software applications have to be developed for various application domains. Further, there is little support by frameworks and tools that help to reuse models and model transformations. Without appropriate support MDSD is difficult to adopt. However, with the use of DSLs and the constant evolution of models and modelling languages there arise interoperability problems that have to be dealt with. Further open research issues in MDE can be found in [37, p.184ff].

### 3.1.4 MDE Tool Suites and Initiatives

Today, there exist a range of MDE tool suites and initiatives that aim to support software development with MDE tools. Commercial tools suites like the Rational Software

Architect [127, 283] from IBM[3], Innovator [187] from MID, or MagicDraw [173] from No Magic[4] support UML modelling and some model transformations for MDA. For a more complete and current list of companies that support MDA see the OMG website[5]. Other commercial MDE products like the Visual Studio DSL Tools [186] from Microsoft or MetaEdit+ [185] from MetaCase[6] focus more on domain-specific modelling and the software factory approach.

In addition to commercial products there exist various open source approaches and projects dedicated to MDSD. The Eclipse Generative Modeling Tools (GMT) project [78] provides a set of research tools illustrating operations applicable to abstract models. Those tools range from code generation (openArchitectureWare (oAW) [81, 206], MOFScript [196]) over model transformation and weaving Atlas ModelWeaver (AMW) [74] to model management ATLAS MegaModel Management (AM3) [73]. The Eclipse Model Driven Development integration (MDDi) project [80] is dedicated to offer a platform with the integration facilities needed for applying a MDE approach. It provides the ability to integrate modelling tools to create a customizable MDE environment. The Generic Modeling Environment (GME) [133] is a configurable toolkit for creating domain-specific modelling and program synthesis environments. The configuration is accomplished through metamodels representing the modelling languages of the application domains. GME is based on the MIC approach and realizes a four-level metamodelling hierarchy (see Section 3.2.2).

In the last years the European Commission has launched various research projects, which reside in the area of MDE. The MODELWARE project [195] was the biggest European research project solely dedicated to MDE. Its goal was to develop the complete infrastructure required for large scale deployment of MDE strategies to ensure the successful adoption of MDE solutions by industry. The MODELPLEX project [194] is the official follow-up of the MODELWARE project. Its goals are to develop an open solution for complex systems engineering built on MDE improving quality and productivity, and to ensure its successful adoption of the solution by the industry. MDE for software development was applied to specific application domains in a variety of projects: In the ATHENA IP project [15] MDSD was applied for cross-organisational development and the enactment and execution of business processes. The SECSE project [264] focuses on cost-effective development and usage of services. It applies MDE techniques to service development and service centric systems engineering. The AMPLE project [10] supports SPL based software development through a combination of AOSD and MDE. The AGILE project [3] applies MDE techniques to the domain of embedded systems. Other projects like the SERIOUS project [268] and the MOMOCS project [197] focus on the methodological aspect. They study and develop methodologies which apply MDE techniques to various application domains. There are multiple other projects in the field of MDE, which are funded by national institutes, like the ModelCVS project [193] or the AgilPro project [4]. The Networked European Software and Services Initiative (NESSI) [201] is a European Technology Platform (ETP) with the objective of defining medium to long-term research and technological objectives. NESSI aims at shaping a vision and building an ecosystem that together enable the emergence of a service-oriented economy in Europe. At the core of the NESSI vision is the provision of new approaches enabling

[3] http://www.ibm.com
[4] http://www.nomagic.com
[5] http://www.omg.org/mda/committed-products.htm
[6] http://www.metacase.com

the transformation of the European economy through service oriented business models [86]. In this thesis we will mention and describe the relevant projects when necessary.

## 3.2 Models

In his work *The Nature of Modeling* [255] J. Rothenberg gives the following definition of modelling:

> *"Modeling, in the broadest sense, is the cost-effective use of something in place of something else for some cognitive purpose. It allows us to use something that is simpler, safer or cheaper than reality instead of reality for some purpose. A model represents reality for the given purpose; the model is an abstraction of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger and irreversibility of reality."*

The most important ideas of this statements towards software development are that models are a copy with probably smaller scale than the software (*"something in place of something else"*) and that models are a simplified version of the software (*"deal with the world in a simplified manner"*).

When entering the MDE world, one is confronted with terms like model, modelling language, metamodel, metametamodel, and various other kinds of models. Moreover, many definitions vary according to the purpose they are used for. Seidewitz [265] defines a *model as a set of statements about some system under study (SUS)*. The UML standard [238] defines a *model as an abstraction of a physical system, with a certain purpose*. Kleppe, Warmer, and Bast give in [158] a more restrictive definition of models in the context of MDA: *"A model is a description of (part of) a system written in a well-defined language"*. The MOF standard defines the notion of a metamodel as follows: *"A metamodel is a model that defines the language for expressing a model"* [210, glossary p.10]. Seidewitz [265] defines *"a metamodel as a specification model for a class of SUS where each SUS in the class is itself a valid model expressed in a certain modelling language"*.

Section 3.2.1 provides definitions for the concepts used in MDE by the means of a megamodel. Sections 3.2.2–3.2.5 describe various aspects of models like concepts to realize modelling tools, modelling languages, model types, and usage scenarios of models.

### 3.2.1 MegaModel for MDE

A megamodel is *"a model whose elements represent models, metamodels and other global entities"* [36]. A megamodel is used to define the set of entities and relations that are necessary to model some aspect about MDE. The megamodel presented in [93, 94] provides a terminology and specification of the concepts use in MDE that will be used throughout this thesis.

In this megamodel the *system* is the central element when talking about MDE. The megamodel distinguishes further between three kinds of systems: *physical systems* (PS) are observable elements or phenomenons pertaining to the physical world, *digital systems* (DS) are those systems that reside in computer memories and are processed by computers, and *abstract systems* (AS) are ideas and concepts that eventually reside in human mind to be processed by human brains.

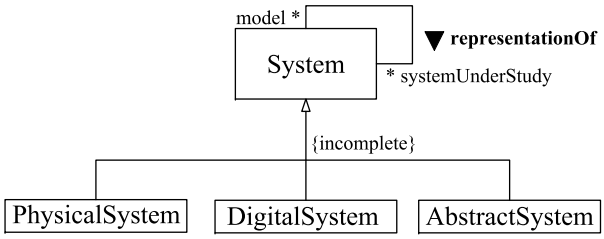


Figure 3.2: Megamodel: system, model, and *representationOf*

[41] defines a *model* as a simplification of a system built with an intended goal in mind. The *model* should be able to answer questions in place of the actual system. In short a model is *representationOf* a system. Figure 3.2 depicts these elements and the *representationOf* relationship [94].

Dealing with more and more systems or models leads to the abstract concept of *set*. The term set refers here to the mathematical concept of the set theory. The relation between the elements of a set and the set will be called *elementOf* in our MegaModel (see Figure 3.3 [93]).

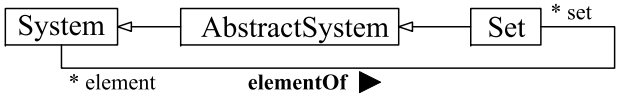


Figure 3.3: Megamodel: set and *elementOf*

A modelling language is a set of models, and models are *elements of* a modelling language. Models *conform to* a model of the modelling language, i.e. a metamodel. Those metamodels can be used to validate models. For one modelling language multiple (meta)models can exist, which can differ in the language they are described in. Figure 3.4 summarizes the basic concepts in the metamodelling context.

For example the English dictionary is a physical model of the English language. If one writes an English text, i.e. a model in the English language, the dictionary can be used to check whether the words in the text exist in the English language. However, there may be words of the English language that are not in the dictionary, i.e. represented in the metamodel. To check the grammar of the English text one would have to use another metamodel, for example an English grammer book.

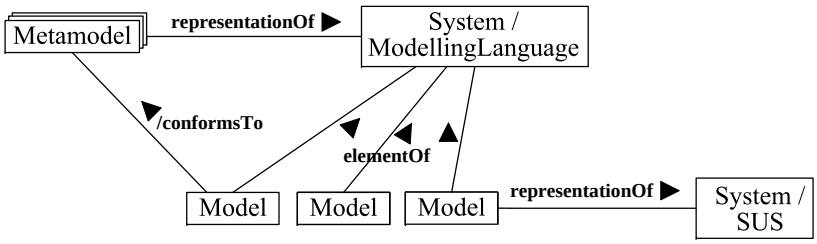


Figure 3.4: Modelling languages, metamodels, models and their relationships

### 3.2.2 Metamodelling Hierarchy

The MDA Guide defines in its glossary metamodel as *"a model of models"* [213, p.A-2]. Since a metamodel itself is a model, *"this structure can be applied recursively many times so that we get a possibly infinite number of meta-layers; what is a metamodel in one case can be a model in another case, and this is what happens with UML and MOF. (...) MOF is commonly referred to as a metametamodel, even though strictly speaking it is a metamodel"* [237, p.16].

The OMG promotes a four-level architecture, which Bézivin [37] more precisely calls a 3+1 architecture (see Figure 3.5). At the bottom level, the M0 layer is the real world system. A model represents this system at level M1. At level M2 metamodels are used to define modelling languages. A model of level M1 conforms to its metamodel defined at level M2. A metamodel itself conforms to the metametamodel at level M3. The metametamodel conforms to itself.

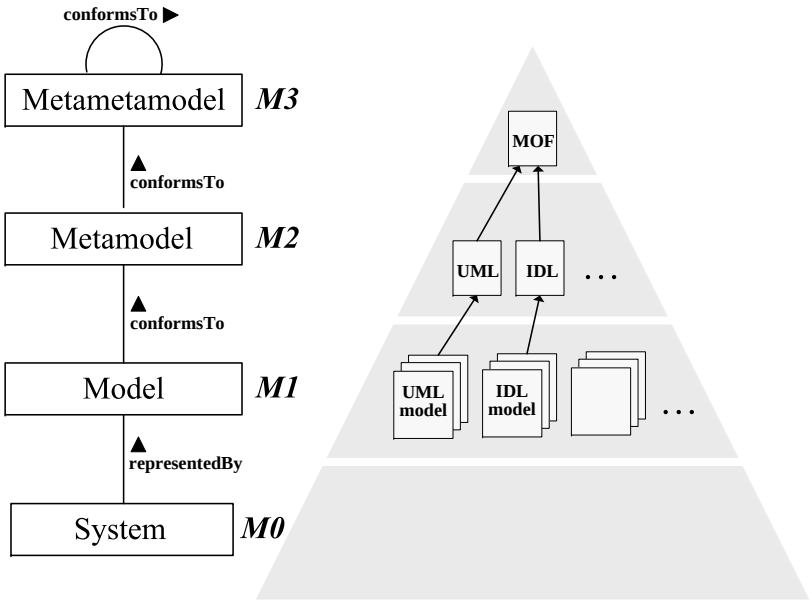


Figure 3.5: The 3+1 metamodelling hierarchy

This metamodelling hierarchy has proved to be extremely valuable in the implementation software engineering and modelling tools. Though many tools have been built to support the Unified Modeling Language (UML) [237, 238], its metamodel, and graphical representation, the real power of the metamodelling hierarchy comes from the existence of level M3 [37]. The existence of the Meta Object Facility (MOF) [228] in MDA at level M3 allows building tools for standardized metamodels like the Common Warehouse Metamodel (CWM) [209], Software Process Engineering Metamodel (SPEM) [223], Enterprise Distributed Object Computing (EDOC) [220], Ontology Definition Metamodel (ODM) [234], etc. Further it allows building coordination between models, based on different metamodels. Examples are the coordination of models though model transformation based on the QVT standard or model exchange based on the XML Metadata Interchange (XMI) standard [221].

A very popular implementation of the described metamodelling hierarchy is the Eclipse Modeling Framework (EMF) [75]. EMF is an implementation of the a Essential

MOF (EMOF) standard, which is the object-oriented subset of MOF, and is a basis for a various (meta)modelling tools that are based on the Eclipse platform. Another implementation of the MOF standard is the Metadata Repository (MDR) [202, 178], which is integrated in the NetBeans Tools Platform. The Kernel MetaMetaModel (KM3) [144] is a DSL for describing metamodels which provides an easy to use textual concrete syntax [18]. The KM3 metamodel is also based on EMOF.

There are further MDE metamodelling environments that do not use MOF or extend MOF as their metametamodel, but are also built on the 3+1 metamodelling hierarchy. The KerMeta [135, 97] language extends MOF with the possibility to specify static and dynamic semantics for metamodels. Other examples are the Generic Modeling Environment (GME) [133, 164], the MetaEdit+ tool [185], or the Visual Studio DSL Tools [186] from Microsoft (a preliminary version of the metametamodel can be found in [42]).

### 3.2.3 UML vs. Domain Specific Languages

The basic use of metamodels and modelling languages is that they facilitate the separation of concerns. When dealing with a given system, one may observe and work with different models of the same system, each one characterized by a given metamodel [37].

MDA treats the UML, which is a general-purpose modelling language for software-intensive systems, as the modelling language of choice for most application modelling, but also provides two possibilities to use Domain Specific Languages (DSLs). A DSL is a modelling language that is designed to express the requirements and solutions of a particular business or architectural domain. The alternative that is commonly recommended to realize DSLs is to use the UML profile mechanism [237], which allows UML to be constrained and customized for specific domains and platforms. The MDA approach also acknowledges the value of non-UML DSLs as a necessary technique and provides the MOF to specify metamodels for DSLs.

In contrast, the software factory approach (see Section 3.1.2) suggests to use UML only for developing sketches, documentation, and conceptual drawings that do not directly relate to code. Non-UML DSLs should be used for developing precise abstractions from which code is generated, mappings between DSLs, and conceptual drawings that have precisely specifiable mappings to other DSLs or to code artifacts [62].

However, decisions of whether to use UML or a DSL are too complex to reduce them to how precise a model is. Based on [283] we present a list of arguments that illustrate further advantages and disadvantages of using UML. Advantages of UML are:

- UML is an open standard modelling language with many available books and training courses. UML is a recognized and transferable skill for software developers.
- UML profiles provide a lightweight approach that is easily implemented using readily available UML tooling. In the future, it may be possible to generate tooling for DSLs, but some customization is still likely to be needed.
- Models with UML profiles applied can be read by all UML tools, even if they don't have any knowledge of the profile.
- Basing all DSLs on UML creates a set of related languages that share common concepts. This makes new profiles more readily understandable and enables mod-

els expressed by different DSLs to be integrated easily. Having a set of models expressed using different DSLs replicates the middleware integration problem at the modelling level.

- UML can be used for both, high-level architectural models and detailed models from which code can be generated. It provides consistency throughout the software life cycle, enabling users to move seamlessly from modelling-in-the-large to modelling-in-the-small.

There are also arguments to use DSLs as they are languages designed for specific usage purposes and application domains, context, and user groups, instead of UML and UML profiles.

- DSLs are adjusted to specific user groups, application domains, and usage context. Thus, for users it is easier to model since the language (exactly) provides the concepts they need for modelling and they have less possibilities for modelling the same thing. UML profiles only permit a limited amount of customization. It is not possible to introduce new modelling concepts that cannot be expressed by extending existing UML elements.

- In DSLs the semantics of the modelling language is better understandable to the users of the application domain. It is easier for people to interpret models right or in the same way.

- The scope of domain specific modelling languages is customized to its application domain and use; the user will be guided by the modelling language towards certain types of solutions. The use of UML does require familiarity with modelling concepts.

- Large monolithic metamodels like UML 2.x have some limitations regarding their usage because of their complexity. Most of UML usages only rely on a small subset of the entire metamodel. Since any well designed process should provide a precise characterization of these subsets, it is necessary to restrict the usage of UML with UML profiles. In general is much more difficult to work by restriction than by extension (developing new DSLs) [37]. This also fosters the possibilities of automating code generation, since code generation does have to take into account less modelling and interpretation possibilities.

### 3.2.4 Types of Models

After describing the concepts of modelling and presenting standards and implementations of those concepts, very few has been said about the actual usage of the models. To understand MDE, one has to know more about the intents and purposes models are build for. Based on [94] we introduce a set of terms that help to differentiate these aspects of models.

The terms *specification model* and *descriptive model* [265] distinguish models on the basis of the purpose they are used for. Specification models are used to specify systems that are built; descriptive models are used to describe existing systems. Another distinction that is based on the usage of models is *sketchy models*, *blueprint models*, and *executable models*. A sketchy model is not precise or complete. The purpose of such

models is typically to try out an idea when the model is a specification or to simplify communication and understanding when the model is descriptive. Blueprint models are more precise and can be used as specifications to build a systems. Normally they describe only some aspect of these systems. Executable Models contain enough information to be directly interpreted by a processor or to derive an executable system. The Executable UML approach [180] is one representative that uses executable models.

In MDE various models at various abstraction levels are used to develop systems. As described in Section 3.1.1, MDA distinguishes between *computation independent models*, *platform independent models*, and *platform specific models*. A similar distinction to specify models at different abstraction levels are *conceptual models*, *specification models*, and *implementation models*. This kind of classification emphasises the fact that each kind of models can be described using the same modelling language.

Another distinction between models is the difference between *product models* and *process models*. Product models are used to specify software artifacts; process models describe the process of software development. For example, the OMG provides with UML a software product metamodel, while Software Process Engineering Metamodel (SPEM) [223] is a metamodel to describe software development processes [37]. One can further distinguish between static and behavioural models. In general semantics specification, checking, verification, code generation, etc. is far easier for static models. Hence, there exist already a variety of practical solutions for static models. In the field of behavioural models these tasks get more complicated. Up to now there is few support for behavioural models except for state machines.

Bézivin postulates in his article *"The unification power of model"* [37] that *"everything is a model"*. To complete this section about models, we provide an non-exclusive list of concepts and artifacts related to the information technology domain which can be seen as models.

- *Programs as models*. Programs are expressed in a programming language. If one makes explicit the correspondence between a grammar and a metamodel, programs can be converted into equivalent MDA-models. For example, MDA models can be converted into Java programs or XML documents. The challenge is to be able to implement agile bridging between these different TSs.

- *Traces as models*. A trace is a model of the dynamic execution of a program. It expresses the specific events traced (object creation, process activations, method calls, exception events, etc.) during program execution.

- *Platforms as models*. In order to generate Platform Specific Models (PSMs) from Platform Independent Models (PIMs) with sufficient automation, precise models of the targeted platforms, i.e. Platform Models (PMs), are necessary.

- *Legacy as models*. To integrate applications and functionality developed for platforms of the past into today's and future platforms, MDE-models have to be extracted from legacy systems. In service-oriented environments one can concentrate on extracting interfaces and protocol specifications by regarding the legacy application as a black box. However, the extraction of MDE models from legacy systems is a great challenge, especially when also complex behaviour has to extracted.

- *Transformations as models*. Model transformations are a kind of metaprogramming, which allow building coordination between models based on different metamodels. Languages to specify model transformation are implemented as DSLs, like the QVT standard in MOF, so that model transformations themselves are models.

- *Verification as models*. Many operations on models may be seen as special cases of transformations. A refactoring, an improvement, or a verification could be viewed as regular operations on models. To enable verification, the semantics of metamodels has to be rigorously like it is done in the UML 2 Semantics Project [292] for UML.

Other forms of models are processes, systems, metamodels, model-elements, measures, tests, aspects, or patterns [36].

### 3.2.5 Models as Assets of Organisations

Organisations which develop and use ICT systems to support their business face challenges like to bridge the semantic gab between the problem space (business requirements) and the solution space (ICT systems) [46], an increase in the complexity of ICT infrastructures and solutions, applications that may be difficult to use, and continued pressure to achieve tight time-to-market timelines [162]. MDE is an approach to address these problems by using models. Models can embody critical solutions and insights and thus can be seen as assets for an organisation.

Larsen [162] describes an asset as *"a collection of artifacts that provides a solution to a problem. The asset has instructions on how it should be used and is reusable in one or more contexts, such as a development or a runtime context. The asset may also be extended and customized through variability points."*.

The main intend behind treating models as assets is to reuse the knowledge and solutions captured in models and transformations (which are also models in MDE). This requires the ability to discover, understand, and customize models for the relevant context in a timely manner. If it takes an organisation longer than what seems reasonable, the organisation will search elsewhere or stop search and re-create the content themselves [162]. The benefits of treating models as assets in MDE are realized through cost reduction, time-to-market reductions, and quality improvements. This is achieved not only through automating the development process but also by capturing and reusing knowledge and critical solutions.

## 3.3 Model Transformation and Code Generation

Model transformations play a key role in MDSD. Vertical model transformations refine abstract models to more concrete models while horizontal model transformations describe mappings between models of the same level of abstraction. Thus, it is important that transformations can be developed as efficiently as possible [103]. With the MOF 2.0 Query, Views, and Transformation (QVT) specification [222, 233] the OMG provides a standard syntax and execution semantics for model-to-model transformations used in a MDSD tools chain. For model-to-text transformations the OMG issued a separated request for proposal in 2004 (MOF Model to Text Transformation

Language [219]) that will eventually lead to a standard for mapping MOF-based models to text. Applications scenarios for which model transformations are developed are manifold: generating lower-level models, and eventually code, from higher-level models [158, 266], mapping and synchronizing among models at the same level or different levels of abstraction [136], creating query-based views of a system [54, 271], model evolution tasks such as model refactoring [282, 330], or reverse engineering of higher-level models from lower-level models or code [92]. The following introduction to model transformations and overview of related terms, approaches, tools and standards is mainly based on [65, 103, 182].

Model Transformations (MTs) are a kind of metaprogramming. They are used to write or manipulate other programs, i.e. models. Thus, a model transformation is defined with respect to the metamodels (see Figure 3.6). When executing a model transformation *Mt*, models like *Ma* conforming to the source metamodel *MMa* are transformed to target models like *Mb* conforming to the target metamodel *MMb*. In general, model transformations can have multiple source and target models. Applying principle 'Everything is a model', the model transformation *Mt* itself is a model [37]. As a consequence, different transformation languages can be defined on the basis of MOF – one of them is QVT. The model transformation $Mt : Ma \rightarrow Mb$ (i.e. the transformation program itself) conforms to a metamodel *MMt* which represents a model transformation language.

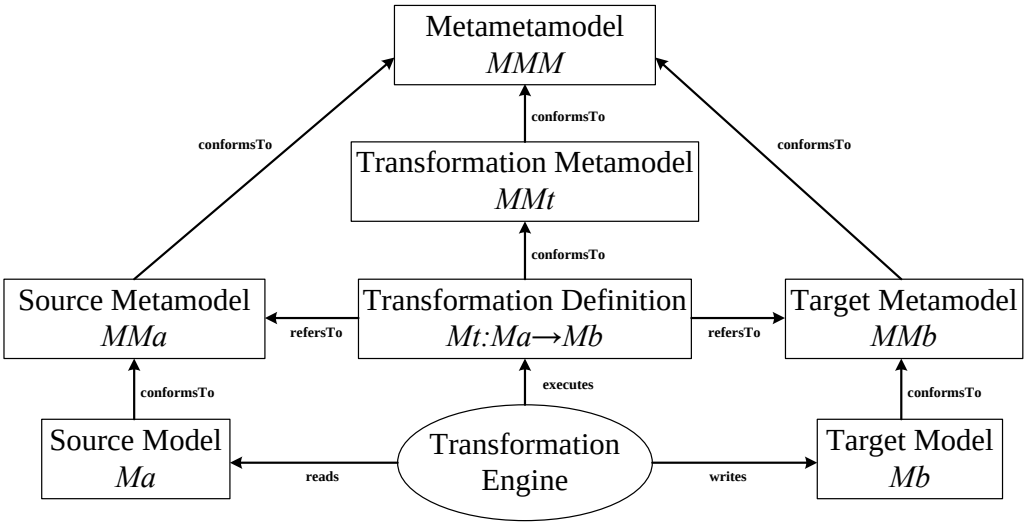


Figure 3.6: Basic concepts of model transformation

One consequence of this organisation is that higher-level transformations are possible, i.e. transformations taking other transformations as input and/or producing transformations as output [37]. Higher-level transformations, which are also referred to as higher-order transformations in literature [38, 182], can be for example applied to refactor a given set of transformations (e.g. a family of code generators) to reduce the amount of code duplication in these transformations. Examples of higher-order transformations can be found in [38] and [295].

Model transformations are characterized through many aspects. Mens and Van Gorp [182] provide a list of dimensions which allows to characterize model transformations.

- *Endogenous vs. exogenous:* Endogenous transformations are transformations between models expressed in the same metamodel. Endogenous transformations

are also called rephrasing. Transformations between models expressed using different metamodels are referred to as exogenous transformations or translations [298]. Typical examples of endogenous transformations are optimization, refactoring, simplification, and normalization of models. Typical examples of exogenous transformations are synthesis of a higher-level specification into a lower-level one, reverse engineering, and migration from a program written in one language to another.

- *Horizontal vs. vertical:* A horizontal transformation is a transformation where the source and target models reside at the same abstraction level. Typical examples are refactoring (an endogenous transformation) and migration (an exogenous transformation). A vertical transformation is a transformation where the source and target models reside at different abstraction levels. A typical example is refinement, where a specification is gradually refined into a full-fledged implementation, by means of successive refinement steps that add more concrete details.

- *Level of automation:* The level of automation is the grade to which a model transformation can be automated. When applying MDSD in practice, one can often find transformations that need to be performed manually (or at least need a certain amount of manual intervention).

- *Complexity:* Model transformation also differ in their complexity. Simple transformations can be for example mappings for identifying relations between source and target model elements. More complex transformations are needed to specify for example synthesis, where higher-level models are refined to lower-level models. The difference in complexity of transformations can require an entirely different sets of techniques and tools.

- *Preservation:* Each transformation preserves certain aspects of the source model in the transformed target model. The properties that are preserved can differ significantly depending on the type of transformation. For example, with refactorings the (external) behaviour needs to be preserved, while the structure is modified. With refinements, the program correctness needs to be preserved [20].

In literature one can find many requirements that are necessary for successful application of model transformations in practices [65, 103, 182, 266]. In [103] Gardner et al. provided a review of submissions to the OMG's QVT-RfP [211] and made suggestions about which concepts and requirements they thought important for a successful adoption of QVT. The following non exhaustive list, which was compiled from the mentioned work, provides an overview of requirements and applications scenarios that model transformation approaches should fulfill.

- Transformations should be able to *handle string expressions* in the source and target model.

- A transformation should support the *propagation of incremental changes* occurring in one model to the other model. Thereby, the changes the user has made to the target models shall be maintained.

- The proposed languages should support the *traceability of transformation executions*. Transformations with a transactional character should be definable (commit,

rollback), which would prevent an invalid (not well-formed) model resulting from a transformation that has failed during execution.

- A transformation language should support the *reuse and extension of generic transformations*. It should provide mechanisms to inherit and override transformations and the ability to instantiate templates or patterns.
- The use of additional transformation data that is not contained in the source model but that *parameterizes the transformation process* should be possible.
- Transformations should be able to *implement updates of models*, i.e. the target model is the same as the source model.
- A transformation language should allow *grouping, composing, and decomposing transformations*.
- Transformation execution should be *resilient to errors*. The occurrence of an exception during transformation execution should not halt the transformation.
- It should be possible to specify *transformations from partial source models*.
- It should be possible to specify symmetrical, *bidirectional transformations*.
- A transformation language should allow to specific *many-to-many transformations* with multiple source and target models.

Beneath the MOF 2.0 QVT specification, which provides a standard for Core, Relational, and Operational languages, and the MOF Model to Text Transformation Language [231] a number of other transformation approaches has be published in literature and implemented in open-source and commercial tools. The following non exhaustive list provides an overview over them:

- *Published in literature:* VIsual Automated model TRAnsformations (VIATRA) framework [296, 295], Kent Model Transformation language [8, 7], Tefkat [107, 163], Graph Rewriting and Transformation language (GReAT) language [5], Atlas Transformation Language (ATL) [40, 145], UMLX [324], A Tool for Multi-formalism and Meta-Modeling (AToM$^3$) [19, 67], Bidirectional Object-oriented Transformation Language (BOTL) [50, 177], MOdel transformation LAnguage (MOLA) [149], Attributed Graph Grammar (AGG) [285], Atlas ModelWeaver (AMW) [43, 44], triple-graph grammars [159], Model Transformation Language (MTL) [299], Yet Another Transformation Language (YATL) [244], Kermeta [199], Constraint-Specification Aspect Weaver (C-SAW) [113], and MT Model Transformation Language [290].
- *Implemented within open-source tools:* AndroMDA [13], openArchitectureWare (oAW) [81, 206], From UML to Java And Back Again (Fujaba) [100], Java Model Driven Architecture (Jamda) [137], Java Emitter Templates (JET) [79], Model Transformation Framework (MTF) [130], MOFScript [196], and StringTemplate [281].
- *Implemented within commercial tools:* XMF-Mosaic [328], OptimalJ [60] MetaEdit+ [185, 288], ArcStyler [110], and Codagen Architect [176].

In the following, we introduce a taxonomy for model transformations (Section 3.3.1) and a classification of model transformation approaches (Section 3.3.2). In the classification section the different transformation approaches are also compared via their strengths and weaknesses. Furthermore, we provide a more detailed introduction to the QVT standard in Section 3.3.3.

### 3.3.1 Features of Model Transformations

In [64] one can find a taxonomy for model transformations published by Czarnecki in 2003, that describes and clarifies terms and concepts relevant for model transformation. The taxonomy was revised in [65] and is represented with feature diagrams [150]. The feature diagram in Figure 3.7 provides an overview of the key features of model transformation languagess. Model-to-model and model-to-text approaches are treated uniformly by the feature diagrams. We distinguish them in the classification Section 3.3.2.

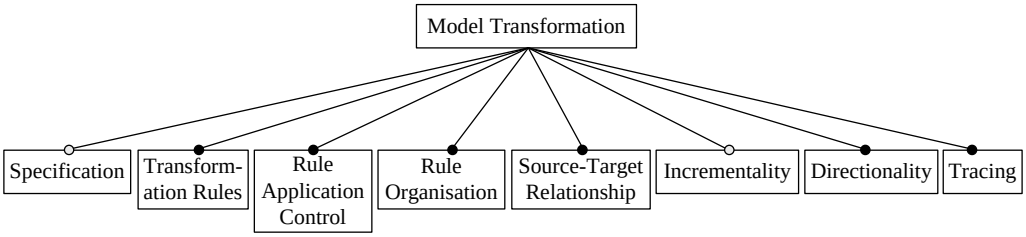


Figure 3.7: Feature diagram representing the top-level areas of variation

- *Specification:* Some transformation approaches provide a dedicated specification mechanism, such as preconditions and postconditions expressed in Object Constraint Language (OCL) [229]. A particular transformation specification may represent a function between source and target models and be executable.

- *Transformation rules:* A transformation rule consists of two parts: a left-hand side (LHS) and a right-hand side right-hand side (RHS). The LHS accesses the source model, whereas the RHS expands in the target model.

  A *domain* is the part of a rule responsible for accessing one of the models on which the rule operates. Rules usually have a source and a target domain, but they may also involve more than two domains.

  The *body* of a domain can be devided into three subcategries: variables, patterns and logic. *Variables* may hold elements from the source and/or target models (or some intermediate elements). *Patterns* are model fragments with zero or more variables. Sometimes, such as in the case of templates, patterns can have not only variables embedded in their body, but also expressions and statements of the metalanguage. *Logic* expresses computations and constraints on model elements.

  The transformations variables and patterns can be typed. In the case of syntactic *typing*, a variable is associated with a metamodel element whose instances it can hold. Semantic typing allows stronger properties to be asserted, such as well-formedness rules (static semantics) and behavioral properties (dynamic semantics). A type system for a transformation language could statically ensure for a transformation that the models produced by the transformation will satisfy a

certain set of syntactic and semantic properties, provided the input models satisfy some syntactic and semantic properties.

- *Rule application control:* For rule application control one can distinguish between location determination and scheduling. Location determination is the strategy for determining the model locations to which transformation rules are applied. Scheduling determines the order in which transformation rules are executed.

- *Rule organisation:* Rule organisation is concerned with composing and stucturing multiple transformation rules by mechanisms such as modularization and reuse.

- *Source-target relationship:* This is concerned with issues such as whether source and target are one and the same model or two different models. Some approaches, such as ATL, mandate the creation of a new target model that has to be separate from the source. However, *in-place* transformation can be simulated in ATL through an automatic copy mechanism. In some other approaches, such as VIATRA and AGG, source and target are always the same model; that is, they only support in-place update. Yet other approaches, for example, QVT Relations and MTF, allow creating a new model or updating an existing one. QVT Relations also support in-place update. Furthermore, an approach could allow a destructive update of the existing target or an update by extension only, that is, where existing model elements cannot be removed.

- *Incrementality:* This refers to the ability to update existing target models based on changes in the source models. The basic feature of all incremental transformations is target-incrementality, that is, the ability to update existing target models based on changes in the source models. This basic feature is also referred to as change propagation in the QVT final adopted specification [222]. A target-incremental transformation creates the target models if they are missing on the first execution. A subsequent execution with the same source models as in the previous execution has to detect that the needed target elements already exist. When any of the source models are modified and the transformation is executed again, the necessary changes to the target are determined and applied. At the same time, the target elements that can be preserved are preserved.

- *Directionality:* Transformations may be unidirectional or multidirectional. Unidirectional transformations can be executed in one direction only, in which case a target model is computed (or updated) based on a source model. Multidirectional transformations can be executed in multiple directions, which is particularly useful in the context of model synchronization. Multidirectional transformations can be achieved using multidirectional rules or by defining several separate complementary unidirectional rules, one for each direction.

- *Tracing:* This is concerned with the mechanisms for recording different aspects of transformation execution, such as creating and maintaining trace links between source and target model elements. Traceability links can be established by recoding the transformation rule and the source elements that were involved in creating a given target element. Trace information can be useful in performing impact analysis (i.e. analyzing how changing one model would affect other related models),

determining the target of a transformation as in model synchronization, model-based debugging (i.e. mapping the stepwise execution of an implementation back to its high-level model), and in debugging model transformations themselves.

### 3.3.2 Classification of Model Transformation Approaches

After introducing a terminology for model transformations and the necessary concepts for model transformations, we now provide a classification of model transformation approaches which is also based on [65]. We distinguish two broad categories of model transformations: model-to-model and model-to-text transformations. While a model-to-model transformation creates its target as an instance of the target metamodel, the target of a model-to-text transformation is just strings [65]. Model-to-text transformation approaches are also often referred to as code generation approaches.

#### Model-to-model Approaches

Several model transformation approaches can be distinguished: direct-manipulation, structure-driven, operational, template-based, relational, graph-transformation-based, and hybrid approaches. In the description of each approach we also provide examples of concrete realizations.

- *Direct manipulation approach:* This category of approach offers an internal model representation and some APIs to manipulate it, such as Java Metadata Interface (JMI). It is usually implemented as an object-oriented framework, which may also provide some minimal infrastructure to organise the transformations (e.g. abstract class for transformations). However, users usually have to implement transformation rules, scheduling, tracing, and other facilities, mostly from the beginning, in a programming language such as Java.

- *Structure-driven approach:* Approaches in this category have two distinct phases: The first phase is concerned with creating the hierarchical structure of the target model; whereas, the second phase sets the attributes and references in the target. The overall framework determines the scheduling and application strategy; users are only concerned with providing the transformation rules. An example of the structure-driven approach is the model-to-model transformation framework provided by OptimalJ.

- *Operational approach:* Approaches that are similar to direct manipulation but offer more dedicated support for model transformation are grouped in this category. A typical solution in this category is to extend the utilized metamodelling formalism with facilities for expressing computations. An example would be to extend a query language such as OCL with imperative constructs. The combination of MOF with such extended executable OCL becomes a fully-fledged object-oriented programming system. Examples of systems in this category are QVT Operational mappings, XMF-Mosaic's executable MOF, MTL, C-SAW, and Kermeta. Specialized facilities such as tracing may be offered through dedicated libraries.

- *Template-based approach:* Model templates are models with embedded metacode that compute the variable parts of the resulting template instances. Model templates are usually expressed in the concrete syntax of the target language, which

helps the developer to predict the result of template instantiation. The metacode can have the form of annotations on model elements. Typical annotations are conditions, iterations, and expressions, all being part of the metalanguage. An obvious choice for the expression language to be used in the metalanguage is OCL. A concrete model-template approach is given by Czarnecki and Antkiewicz [63].

- *Relational approach:* This category groups declarative approaches in which the main concept is mathematical relations. In general, relational approaches can be seen as a form of constraint solving. Examples of relational approaches are QVT Relations, MTF, Kent Model Transformation Language, Tefkat, AMW, and mappings in XMF-Mosaic. The basic idea is to specify the relations among source and target element types using constraints. In its pure form, such a specification is non-executable (e.g. relations [8, 215] and mapping rules [214]). However, declarative constraints can be given an executable semantics, such as in logic programming. All of the relational approaches are side-effect-free and, in contrast to the imperative direct manipulation approaches, create target elements implicitly. Relational approaches can naturally support multidirectional rules. They sometimes also provide backtracking. Most relational approaches require strict separation between source and target models; that is, they do not allow in-place update.

- *Graph-transformation-based approach:* This category of model transformation approaches draws on the theoretical work on graph transformations. In particular, this category operates on typed, attributed, labeled graphs [12], which can be thought of as formal representations of simplified class models. Examples include AGG, AToM$^3$, VIATRA, GReAT, UMLX, BOTL, MOLA, and Fujaba.

  Graph transformation rules have an LHS and an RHS graph pattern. The LHS pattern is matched in the model being transformed and replaced by the RHS pattern in place. The LHS often contains conditions in addition to the LHS pattern. Some additional logic, for example, in string and numeric domains, is needed to compute target attribute values such as element names. Graph patterns can be rendered in the concrete syntax of their respective source or target language (e.g. in VIATRA) or in the MOF abstract syntax (e.g. in BOTL and AGG). The advantage of the concrete syntax is that it is more familiar to developers working with a given modelling language than the abstract syntax.

- *Hybrid approach:* Hybrid approaches combine different techniques from the previous categories. The different approaches can be combined as separate components or, in a more fine-grained fashion, at the level of individual rules. QVT is an example of a hybrid approach with three separate components, namely Relations, Operational mappings, and Core. Examples of the fine-grained combination are ATL and YATL.

  A transformation rule in ATL may be fully declarative, hybrid, or fully imperative. The LHS of a fully declarative rule (so-called source pattern) consists of a set of syntactically typed variables with an optional OCL constraint as a filter or navigation logic. The RHS of a fully declarative rule (so-called target pattern) contains a set of variables and some declarative logic to bind the values of the attributes in the target elements. In a hybrid rule, the source or target patterns are complemented with a block of imperative logic which is run after the application of the target

pattern. A fully imperative rule (so-called procedure) has a name, a set of formal parameters, and an imperative block, but no patterns. Rules are unidirectional and support rule inheritance.

- *Other approaches:* There are two more approaches which do not fit in the described categories: transformation implemented using Extensible Stylesheet Language Transformation (XSLT) [301] and the application of metaprogramming to model transformation. Because models can be serialized as XML using the XMI [221], model transformations could be implemented with Extensible Stylesheet Language Transformation (XSLT), which is a standard technology for transforming XML. Unfortunately, the use of XMI and XSLT has scalability limitations. Manual implementation of model transformations in XSLT quickly leads to non-maintainable implementations because of the verbosity and poor readability of XMI and XSLT. A more promising direction in applying traditional metaprogramming techniques to model transformations is a domain-specific language for model transformations embedded in a metaprogramming language [290].

Garnder et al. [103] made a recommendation about which model transformation approach to use. They based their statement on a quote of Adam Bosworth [48].

> *"Alan Kay is supposed to have said that simple things should be simple and hard things should be possible. It has been my experience over 25 years of software development that for most software products, simple things should be declarative and/or visual and hard things should be procedural. Declarative languages have an unfortunate tendency to metastasize because people need to do things that are hard."*

In general, for simple transformations and for identifying relations between source and target model elements a relational appoach approach should be used. An operational approach is preferable for the definition of complex many-to-many transformations that involve detailed model analysis. In practice, a number of transformation languages like QVT offer the possibility to combine elements of both approaches in transformation specifications.

### Model-to-text Approaches

Model-to-text transformation corresponds to the concept of 'pretty printing' in program transformation. Model-to-text approaches are useful for generating both code and non-code artifacts such as documents. If one provides a metamodel for the target programming languages, one can view transforming models to code as a special case of model-to-model transformations. However, for practical reasons of reusing existing compiler technology and for simplicity, code is often generated simply as text, which is then fed into a compiler [65].

- *Visitor-based approach:* A very basic code generation approach consists in providing some visitor mechanism to traverse the internal representation of a model and write text to a text stream. An example of this approach is Jamda – an object-oriented framework providing a set of classes to represent UML models, an API for manipulating models, and a visitor mechanism (CodeWriters) to generate code. Jamda does not support the MOF standard to define new metamodels;

however, new model element types can be introduced by subclassing the existing Java classes that represent the predefined model element types.

- *Template-based approach:* The majority of currently available MDA tools support template-based model-to-text generation (e.g. oAW, JET, Codagen Architect, AndroMDA, ArcStyler, MetaEdit, and OptimalJ). AndroMDA reuses existing open-source template-based generation technology: Velocity [297] and XDoclet [329].

  A template usually consists of the target text containing slices of metacode to access information from the source and to perform code selection and iterative expansion. According to our terminology, the LHS uses executable logic to access source, and the RHS combines untyped string patterns with executable logic for code selection and iterative expansion. Furthermore, there is no clear syntactic separation between the LHS and RHS. Template approaches usually offer user-defined scheduling in the internal form of calling a template from within another template.

  The LHS logic accessing the source model may have different forms. The logic could be simply Java code accessing the API provided by the internal representation of the source model such as JMI, or it could be declarative queries, for example, in OCL or XML Path Language (XPath) [316]. The oAW Generator Framework propagates the idea of separating more complex source access logic, which might need to navigate and gather information from different places of the source model, from templates by moving the logic into user-defined operations of the source-model elements.

Compared with a visitor-based transformation, the structure of a template resembles more closely the code to be generated. Templates lend themselves to iterative development as they can be easily derived from examples.

### 3.3.3 OMG Standard: Query/View/Transformation

The OMG adopted the *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification* (QVT) [233] as standard for model transformations. The QVT specification defines a hybrid transformation language. The three transformation languages *Relations*, *Core*, and *Operational Mappings* provide declarative and imperative transformation constructs. As one can see in Figure 3.8, the declarative part is split into a two-level architecture. The languages *Relations* and *Core* can be used to specify declarative transformations at different levels of abstraction. QVT provides two options to extend declarative specifications with imperative transformation constructs, the *Operational Mappings* language and *Black Box* operations.

The *Relations* language allows a declarative specification of the relationships between MOF models. The Relations language supports complex object pattern matching. Relations can assert that other relations also hold between particular model elements matched by their patterns. The pattern matching results are used to instantiate model elements in new models and to apply changes to existing models. The semantics of Relations is defined through a combination of English, first order predicate logic, and a transformation *RelationsToCore* to the Core language. The *Core* language is a more elementary declarative language than the Relations language. Traceability links are treated as ordinary model elements like the source and target model. The Core language is

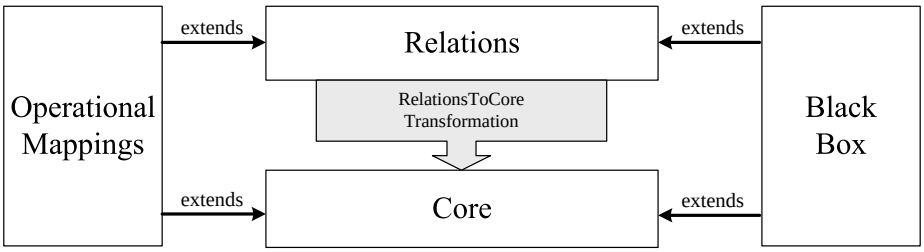


Figure 3.8: QVT languages architecture

equally powerful to the Relations language, and because of its relative simplicity, its semantics can be defined more simply. One purpose of the Core language is to provide a reference semantics of the Relations language. The *Operational Mappings* language extends the Relations language with imperative constructs and OCL constructs with side effects. Transformations can be defined purely in the Operational Mappings language (operational transformations) or through a combination of Relations and Operational Mappings constructs (hybrid approach). The *Black Box* mechanism allows the use of code in model transformations. Complex algorithms can be coded in any programming language and existing libraries can be reused.

### QVT Relations

As the QVT Relations language is further important in this thesis, we provide more detailed information. The examples and the description is based on the QVT standard specification [233].

**Transformations and Model Types** In the Relations language a transformation (*Transformation*) between models is specified as a set of relations (*Relation*) that must hold for the transformation to be successful. A transformation can be applied to models that conform to a model type (*TypedModel*), which is a specification what kind of model elements any conforming model can have, similar to a variable type specifying what kind of values a conforming variable can have in a program. The types of elements these models can have are restricted to those within a set of referenced packages which are in most cases metamodels. The models for which a transformation is specified are parameters (+*modelParameter*) of the transformation.

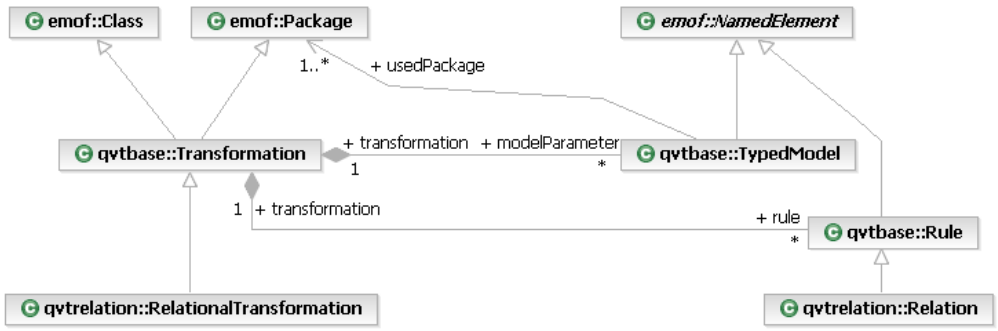


Figure 3.9: Relations metamodel: transformation and model types

Listing 3.1 specifies a transformation named *UmltoRdbms* between the models *uml* and *rdbms*. The model named *uml* declares the *UmlMM* package as its metamodel, and the *rdbms* model declares the *RdbmsMM* package as its metamodel.

Listing 3.1: Transformation example

```
1 transformation UmlToRdbms(uml:UmlMM; rdbms:RdbmsMM) {
2   ...
  }
```

**Relations and Domains** Relations (*Relation*) in a transformation declare constraints that must be satisfied by the elements of the models referenced by the transformation. A relation is defined by two or more domains (*RelationDomain*). A domain is a distinguished typed variable that can be matched in a model of a given model type. A domain has a pattern (*DomainPattern*) which can be viewed as a graph of object nodes, their properties, and association links originating from an instance of the domain's type. Alternatively, a pattern can be viewed as a set of variables and a set of constraints that model elements bound to those variables must satisfy to qualify as a valid binding of the pattern. Variables (*Variable*) are specified within the relations. A relation domain has a distinguished typed variable called the root variable (+*rootVariable*) that can be matched in a model of a given model type.

A transformation invoked for enforcement is executed in a particular direction by selecting one of the models of the transformation as the target. The target model may be empty or may contain existing model elements to be related by the transformation. Whether or not the relationship is enforced is determined by the target domain, which may be marked as *checkonly* or *enforced*.

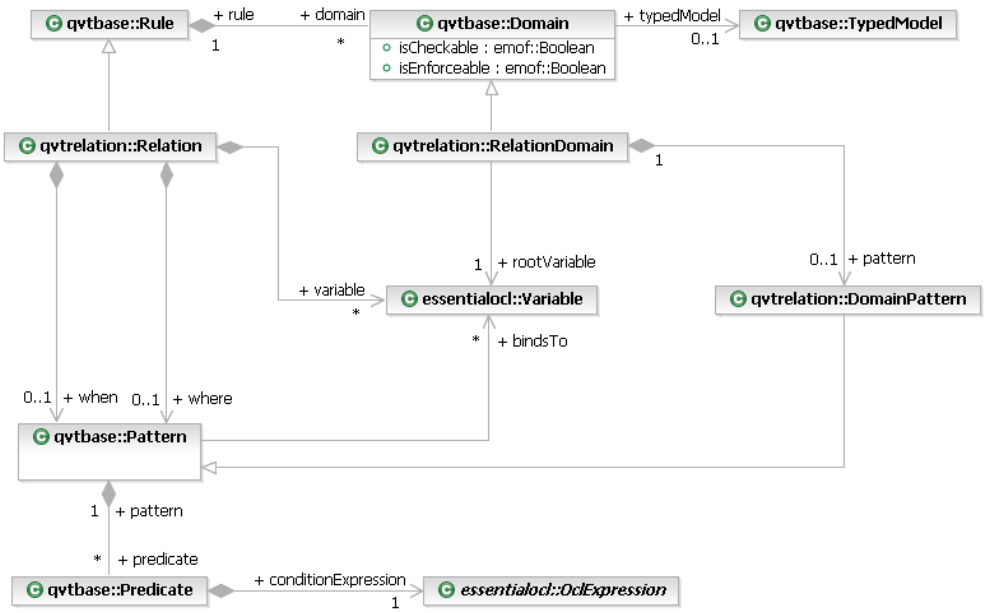


Figure 3.10: Relations metamodel: relations and domains

In Listing 3.2 two domains are declared which will match elements in the *uml* and *rdbms* models respectively. Each domain specifies a simple pattern - a *Package* with a *name* and a *Schema* with a *name*. Both *name* properties are bound to the same variable *pn* implying that they should have the same value. The domain for the *uml* model is marked *checkonly* and the domain for the *rdbms* model is marked *enforce*.

Listing 3.2: Relation and domain example

```
top relation PackageToSchema {
  pn : String;
  checkonly domain uml p:Package {
    name=pn
    };
  enforce domain rdbms s:Schema {
    name=pn
    };
}
```

A relation can also define a *when* clause, that specifies the conditions under which the relationship needs to hold, and a *where* clause, that specifies the condition that must be satisfied by the model elements that are being related. The *when* and *where* clauses are patterns (*Pattern*). A pattern is a set of variable declarations (+*bindsTo*) and predicates (*Predicate*), that must evaluate to true for a binding of the variables of the pattern (cp. Figure 3.10).

Listing 3.3 specifies a mapping between a *Class* and a *Table*. The *when* clause specifies that the relation *ClassToTable* needs to hold only when the *PackageToSchema* relation holds between the *Package* containing the *Class* and the *Schema* containing the *Table*. Whenever the *ClassToTable* relation holds, the relation *AttributeToColumn* must also hold.

Listing 3.3: When and where clauses example

```
relation ClassToTable {
  checkonly domain uml c:Class {
    namespace = p:Package{},
    ...
    };
  enforce domain rdbms t:Table {
    schema = s:Schema{},
    ...
    };
  when {
    PackageToSchema(p,s);
  }
  where {
    AttributeToColumn(c,t);
  }
}
```

**Patterns and Templates** A domain pattern (*DomainPattern*) can specify an arbitrarily complex pattern graph consisting of template expressions, Object Template Expressions (OTEs) (*ObjectTemplateExp*), and Property Template Items (PTIs) (*PropertyTemplate-Item*). A template expression specifies a pattern that matches model elements in a model of a transformation. The matched model element may be bound to a variable and this variable may be used in other parts of the expression. A template expression may match

either a single model element or a collection of model elements depending on whether it is an object template expression or a collection template expression. An OTE can have other template expressions nested inside it to an arbitrary depth. An OTE specifies a pattern that may match only single model elements. OTEs have a type specified by the referred class. An OTE is specified by a collection of PTIs, each corresponding to different attributes of the referred class. PTIs are used to specify constraints on the values of the slots of the model element matching the container OTE. The constraining expression is given by the *value* expression and the constraint is on the slot that is an instance of the referred property. A relation call expression (*RelationCallExp*) specifies the invocation of a relation. A relation may be invoked from the *when* or *where* clause of another relation.

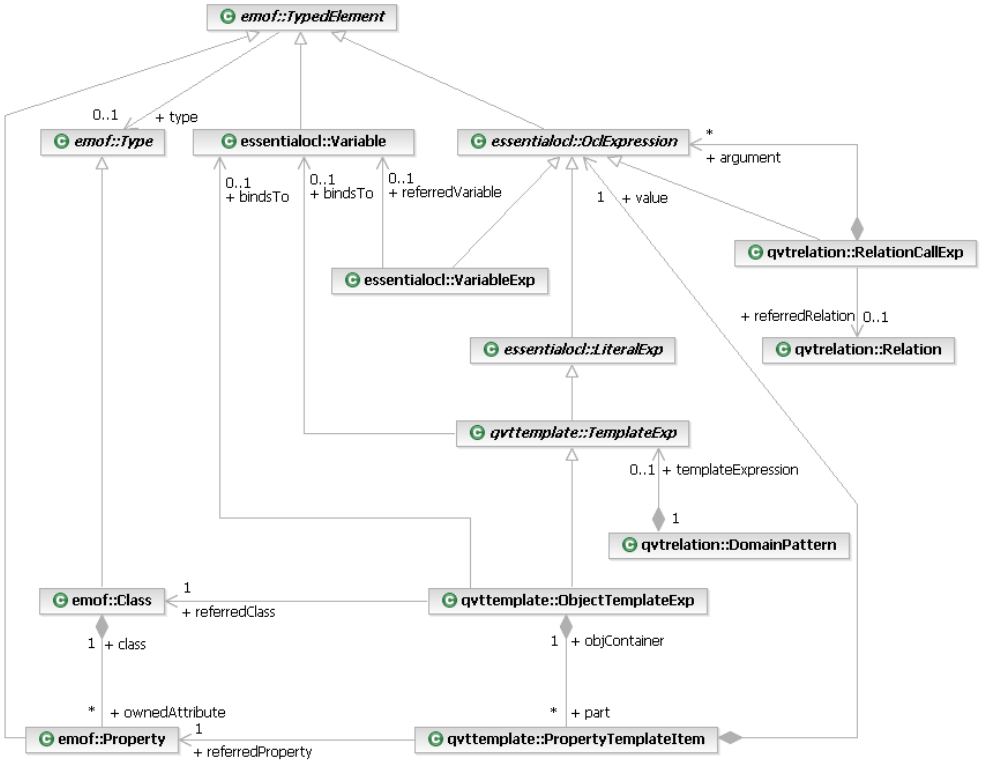


Figure 3.11: Relations metamodel: patterns and templates

The *ClassToTable* relation in Listing 3.4 defines several OTEs which are used to match patterns in models. For example one OTE is associated with the domain of *uml*. Pattern matching will bind all the variables in the expression (*c*, *p*, and *cn*), starting from the domain's root variable *c* of type *Class*. In this example the variable *p* would already have a binding resulting from the evaluation of the when clause expression *Package-ToSchema(p,s)*. The matching proceeds by filtering all of the objects of type *Class* in the *uml* model, eliminating any which do not have the same literal values for their properties as the template expression. Any *Class* with its *kind* property not set to *'Persistent'* is eliminated.

Listing 3.4: Pattern and template example

```
1  relation ClassToTable {
2    cn : String;
     checkonly domain uml c:Class {
4      namespace = p:Package{},
       kind='Persistent',
6      name=cn
       };
8    enforce domain rdbms t:Table {
       schema = s:Schema{},
10     ...
       };
12   when {
       PackageToSchema(p,s);
14   }
   }
```

For properties that are compared to variables, such as *name=cn*, two cases arise. If the variable *cn* already has a value binding, then any class that does not have the same value for its name property is eliminated. If the variable *cn* is free, as in the example, then it will get a binding to the value of the name property for all classes that are not filtered out due to a mismatch with other property comparisons. The value of *cn* will be either used in another domain, or can have additional constraints placed on it in the where expression of the domain or its owning relation.

The matching proceeds with properties whose values are compared to nested template expressions. For example, the property pattern *namespace = p:Package* will match only those classes whose *namespace* property has a non-null reference to a *Package*. At the same time, the variable *p* will be bound to refer to the Package. Since in the example *p* is already bound in the *when* clause, the pattern will only match those classes whose *namespace* property has a reference to the same package that is bound to *p*.

**Keys and Object Creation** OTEs also serve as templates for creating objects in a target model. When creating objects one wants to ensure that duplicate objects are not created when the required objects already exist. In such cases the existing objects shall just be updated. The Relations metamodel introduces the concept of *Key*, which defines a set of properties of a class that uniquely identify an object instance of the class in a model.

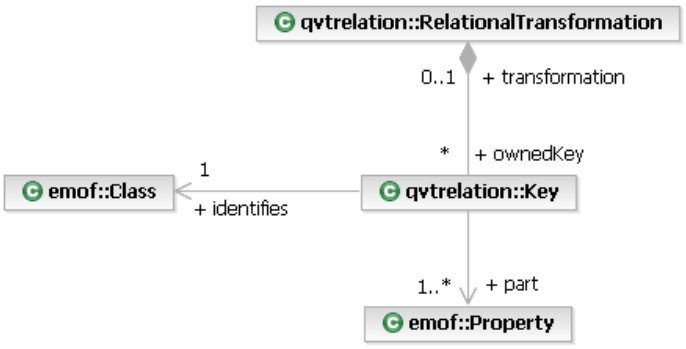


Figure 3.12: Relations metamodel: keys and object creation

Listing 3.5 depicts a key for the class *Table*. A *Table* is uniquely identified by the

properties *schema* and *name*. Keys are used at the time of object creation. If an OTE has properties corresponding to a key of the associated class, then the key is used to locate a matching object in the model; a new object is created only when a matching object does not exist.

Listing 3.5: Keys and object creation example

```
1  key Table {schema, name};
2  relation ClassToTable {
     cn : String;
4    checkonly domain uml c:Class { ... };
     enforce domain rdbms t:Table {
6      schema = s:Schema{},
       name = cn,
8      ...
       };
10 }
```

**Execution Semantics** QVT Relations allows to specify model transformations in a declarative way based on a powerful pattern matching mechanism and OCL constraints on the models. This facilitates developing a consistent transformation for the user, but at the same time involves complex execution semantics with nested loops of object tuples for the execution engine [233]. QVT Relations uses implicit rule scheduling which is based on the dependencies among the relations.

The execution semantics first conducts a checking step, where it is checked, whether there exists a valid match in the target model that satisfies the relationship with the source model. Second, the enforcement step modifies the target model so that it satisfies the relationship to the source model. This is done on the basis of the checking results.

## 3.4 Summary

Up to now a wide range of concepts and sophisticated techniques have been developed, that provide a good foundation for applying MDSD to real, industrial software developments projects. In practice however, (software development) projects not only depend on production techniques, i.e. model and code generation. Other factors like the people involved in these projects, the relationships between collaborating organisations, or market barriers are at least equally important for the success of a software develop projects.

In larger software development projects it is necessary to guide people, define development artifacts and their correlations, specify project control techniques, etc. MDSD can be used to improve project quality and development time, but does not achieve these goals on its own. Though some work has already been done to use MDSD techniques in software development (e.g. the MOMOCS project), the study of the complementary notion of MDSD and methodologies in software development projects is an open research area.

MDSD development fosters the reuse of development artifacts and its solutions. There will be whole libraries of MDSD artifacts from which IT architectures and developers can choose. However, the efficient selection of adequate artifacts, that are combined to the best solution for a given problem, requires highly complex decisions. Today, there is a lack of appropriate decision support for MDSD that helps IT architects and developers to make good choices on the basis of a variety of influence factors.

MDSD is not a single-user desktop application or applied in closed teams. When organisations collaborate, also their IT departments have to collaborate to provide integrated ICT solutions. Similar to the integration of systems at code level, there arise interoperability issues in MDSD at higher abstraction levels. History shows, that standardization of e.g. communication protocols or data exchange formats solve interoperability problems only in small or very specific domains. Hence, MDSD has to be supported by techniques that increase the interoperability of MDSD tools and artifacts.

Today, there still exist barriers to the application of MDSD. The combination and integration of MDSD tools as well as the development of MDSD artifacts, like transformations and generators, are highly complex tasks. Organisations have to be supported by existing, reusable, and adaptive solutions that can be easily adapted to their problems.

In the next chapters we describe such solutions, which are the main contributions of this thesis. In Chapter 4 we facilitate the generation of executable code from high-level business process descriptions by providing architecture patterns, model transformations, and a code generation framework for model-driven CBP modelling and enactment. In Chapter 5 we develop a new decision method and guidelines for selecting appropriate ICT architectures. Chapter 6 introduces a solution the facilitates the exchange of models between different enterprises as well as the reuse and evolution of model transformations.

# Chapter 4

# Enacting Cross-organisational Business Processes with MDSD

Enterprises need to adapt their constantly evolving business relationships, cross-organisational value chains and business systems in order to remain competitive. Since IT is a key enabler of cross-organisational value chains, we apply MDSD to specify changes of business processes at the business level and propagate them down to the level of ICT systems.

In MDSD high-level models are refined to more detailed models via model transformations. Like in the MDA, the ATHENA IP framework uses three abstraction levels for modelling and executing CBPs (cp. Figure 1.1 and Figure 4.1). High-level CIMs are used at business expert level to model and agree on CBPs. PIMs, that represent a model of the ICT and contain information about architecture and technological paradigms, are of interest to the IT expert. Finally, executable PSMs are generated for the IT system abstraction level. From these PSMs code can be derived automatically.

However, additional information is needed to implement the model transformations that narrow the gap between the different abstraction levels. Transformations between CIMs and PIMs commonly encode knowledge about software architecture or best practices. The solutions implemented in these transformations also heavily depend on the technological paradigms (service-orientation, monolithic, peer-to-peer, etc.) the ICT systems under development is based on. PIM to PSM transformations are governed by PMs, whereas the term platform is often interpreted in a way that is appropriate for the software development context (see Section 3.1.1).

To facilitate an efficient and effective generation of executable code from high-level, domain-specific models, there is a strong need for frameworks, tools, and model transformations that support the application MDSD (cp. Challenge 1 in Chapter 1). In this chapter we develop and apply MDSD concepts and techniques to provide end-to-end support for the design and execution of CBPs. In Section 4.1 we refine CBPs to concrete service-oriented implementations by largely automated model transformations that encode software architecture patterns. A second transformation is triggered by a platform model that contains knowledge about the target process execution platform (WS-BPEL, XPDL, etc.) and execution environment (see Section 4.2). This transformation additionally encodes quite complex knowledge about graph and process flow transformations.

In this chapter we develop the following solutions that facilitate a model-driven development and execution of CBPs by bridging the gap between the different abstraction

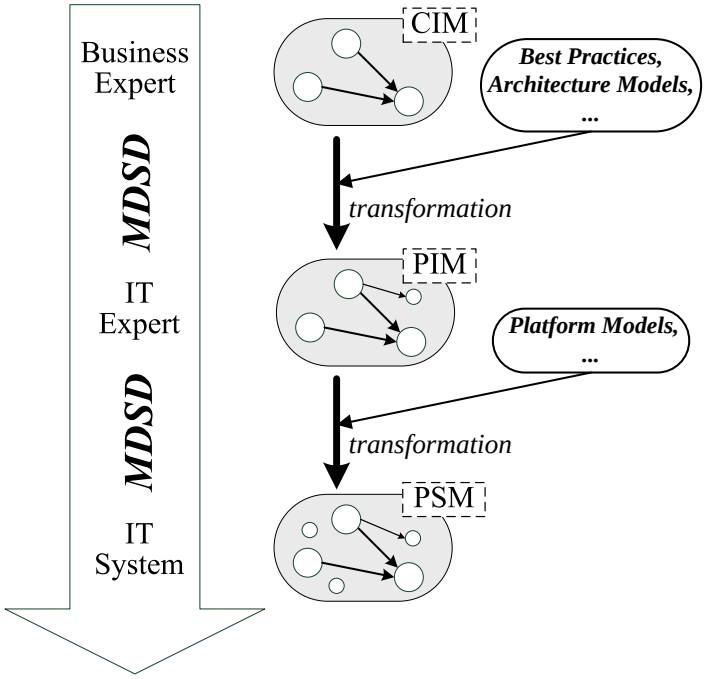


Figure 4.1: MDSD steps

levels:

- Software architecture patterns that enable ICT system coordination in a service-oriented environment.
- Implementation of a set of model transformations that are based on software architecture patterns and allow automated generation of service-oriented PIMs from computational independent CBP models.
- A model and code generation framework that facilitates the generation of executable workflow code from higher-level process models.
- A case study in which the model and code generation framework is applied to generate WS-BPEL code from arbitrary higher-level process descriptions.

## 4.1 Transforming CIM to PIM

To close the semantic gap between business level models (CIMs) and platform-independent models of ICT systems (PIMs), knowledge about the software architecture and best practices has to be encoded into the respective model transformation. Hence, we first develop software architecture patterns for ICT system coordination in Section 4.1.2, before we implement the respective model transformations (see Section 4.1.4). In Section 4.1.3 we introduce a case study, which we use to illustrate how service-oriented models of ICT systems can be derived from business process descriptions via model transformations.

### 4.1.1 Problem Description

Enterprise Application Integration (EAI) projects serve the need of organisations to form and work together in networks by coupling arbitrary ICT systems. These coupled sys-

tems for example facilitate cross-organisational business processes. EAI solutions face, like the organisations themselves, the challenge to adapt their processes with increasing speed and to respond quickly to changing requirements. The main objective of most integration initiatives is to achieve a new level of interoperability while minimizing the impact on existing ICT environments. According to Erl [84] this means:

- avoiding the creation of a fragmented environment through the introduction of business logic that resides outside of established application boundaries.
- avoiding tightly bound integration channels between applications that are easily broken if either application is modified.
- minimizing redevelopment of applications affected by the integration.

Business process level integration has been accomplished through a variety of point-to-point integration models, most commonly facilitated by broker and orchestration components. In recent years, a new generation of EAI solutions has been developed under the service-oriented paradigm, which lends itself to develop highly adaptable solutions and to reuse existing applications. In a service-oriented world, sets of services are assembled and reused to quickly adapt to new business needs. However, *service-orientation does not provide an integration solution by itself*. Service-oriented integration introduces the concept of service to establish a platform-independent model with various integration architectures.

To supporting EAI for service-oriented systems and the enactment of CBPs with MDSD poses two main challenges:

- The development of appropriate software architecture coordination patterns, that can be applied for the integration of cross-organisational systems and the coordination of CBPs.
- Implementation of model transformations encoding the developed software architecture coordination patterns. ICT system models that are based on the coordination patterns can be automatically derived from higher-level CBP descriptions.

### 4.1.2 Software Architectures for ICT System Coordination

Service-oriented integration solutions can be categorized by their topology (see Figure 4.2). In a purely decentralized peer-to-peer topology services of the participating organisations implicitly establish the collaborative process through direct message exchange; this is a realization of choreography. In a hierarchical topology a controller service defines the steps necessary to achieve the overall goal and maps these steps to services provided by the contributing organisations; this is a realization of orchestration. However, in many cases, a mixture of hierarchical and decentralized peer-to-peer topology, i.e. a heterogeneous topology, is used to realize complex multipartner collaborations [168].

Based on these abstract topologies for CBP enactment, we now have a closer look at these coordination architectures and how they can be applied to realize service-oriented integration solutions. These architectures are used to control the conversation flow between the participating organisations. For the description of the coordination architecture we assume, that each organisation willing to participate in a cross-organisational

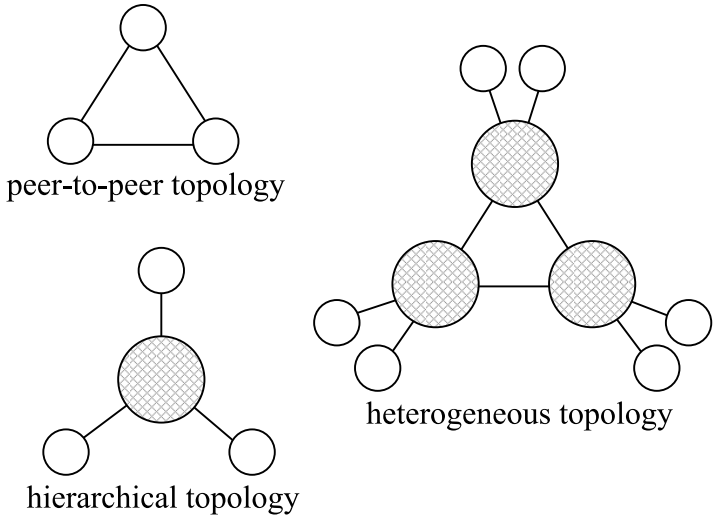


Figure 4.2: Coordination topologies

collaboration supported by ICT systems, has a set of Elementary Services (ESs). These ES are application, business, or hybrid services. In our descriptions we also assume without loss of generality, that the ESs are realized as process services, so that we can use the distinction between executable and abstract process. Nevertheless, ESs could be realized by arbitrary code fragments. An ES can only be a controller service with regard to the organisations' internal service composition, but not with regard to the collaboration process. Cross-organisational Business Processes (CBPs) represent the conversation flow and message exchange between the organisations participating in the collaboration (in particular in an agent communication language).

- *Brokerless architecture*: A brokerless coordination architecture (see Figure 4.3) can be used to realize the decentralized peer-to-peer topology, where messages are exchanged directly between the ESs of the participants as usual in an agents world. Due to the mutual exchange of messages the ESs depend on each other. Control flow logic of CBPs is realized by the executable process of the participants' ESs. Changing the business protocol would result in changing multiple ESs, i.e. their executable processes. Further, the abstract process of the ESs are directly exposed to the collaboration space and therefore are directly accessible by entities outside enterprise boundaries.

- *Central broker architecture*: Figure 4.4 depicts the central broker coordination architecture. Messages are no longer exchanged directly between the ESs, but over a central broker component, which is realized by a controller service. The controller service is a process that orchestrates the ESs of the participating organisations. It acts as a global observer process coordinating the partners as well as making decisions on the basis of data used in the CBP. In the case of a change to the CBP protocol's messages and semantics, only the broker process needs to be modified. Since the broker process is not necessarily owned by one of the participating partners, organisations may hide their elementary services from their collaborators. However, they have to reveal them to a third party instead.

- *Decentral broker architecture*: The *decentral broker architecture* introduces ele-

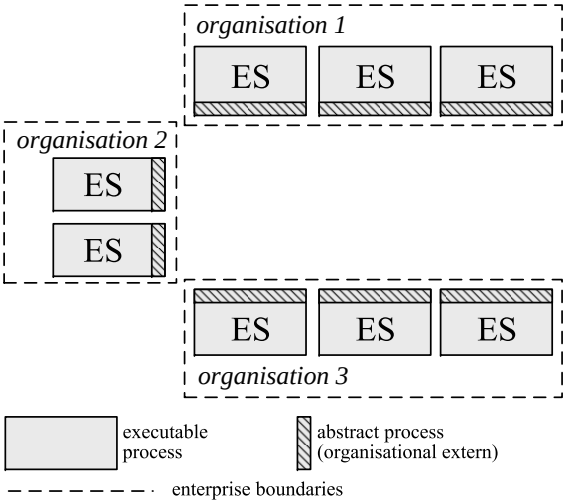


Figure 4.3: Brokerless architecture

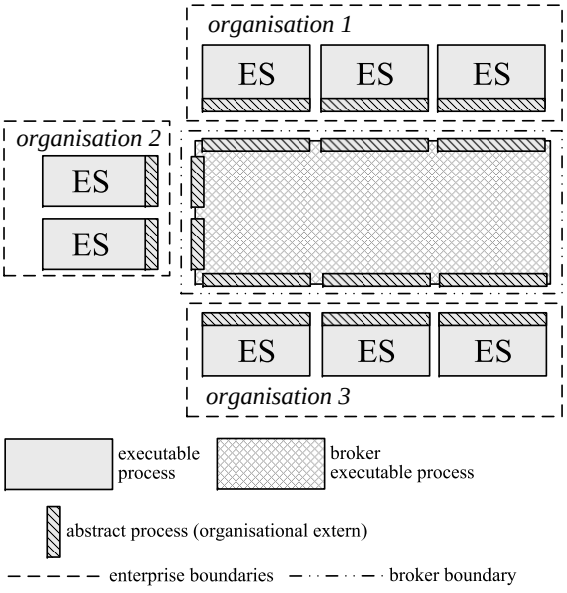


Figure 4.4: Central broker architecture

ments of the decentralized peer-to-peer topology in the hierarchical topology of the central broker architecture. It splits the single broker component into several controller processes jointly providing the broker functionality (note the boundaries in Figure 4.5). Each organisation provides one controller service, also called View Processes (VPs) (cp. Section 2.2.3), which orchestrates the organisation's internal ESs. Messages across organisational boundaries are only exchanged by the VPs, which encapsulate the ESs. In this architecture the elementary service can be seen as kind of Private Processes (PPs).

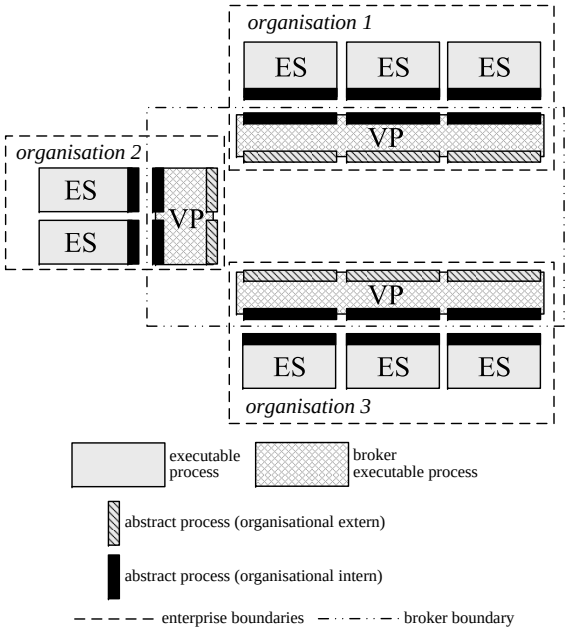


Figure 4.5: Decentral broker architecture

### 4.1.3 Case Study

This case study shows an example of the automotive industry. In this industrial sector it is unusual that automotive manufacturers change their component suppliers often or in short terms[1]. In most cases contracts with suppliers have medium-term to long-term contract periods and high contract volumes. For this reason the process of choosing the suppliers has special significance. Admittedly, ICT systems cannot automate the whole process yet, but enable a better coordination of the activities between the Original Equipment Manufacturer (OEM), the Purchasing Organisation (PO), and the Suppliers (SUs) in terms of resource and applications involved in this process. ICT systems can also provide monitoring and coordination of the solicitation of quotations and check constraints regarding the composition of the suppliers or the combination of suppliers' components.

The example of this case study comprises the solicitation of quotations and the choice of component suppliers by an automotive manufacturer. Although the processes of the example are shortened compared to the reality, they reflect the essential activities

[1]Cp. ATHENA IP internal document: *As-Is Automotive Scenario*, WD.B4.4.1.

of the solicitation of quotations and the choice of suppliers quite well. Three roles are involved in the CBP:

- **Original Equipment Manufacturer (OEM):** An OEM is the automotive manufacturer planning to produce a new automobile type.
- **Purchasing Organisation (PO):** The PO can be an independent company or department of the OEM conduction the solicitation of quotations and the final selection of the suppliers.
- **Supplier (SU):** The SU is a component supplier for the automotive industry aiming to place contracts with the OEM and PO.

In Figure 4.6 we can see an overview of the process steps, which are conducted by these three roles. The roles OEM, PO and SU are described by swimlanes. A process step is modeled within the swimlane of the role conduction the step.

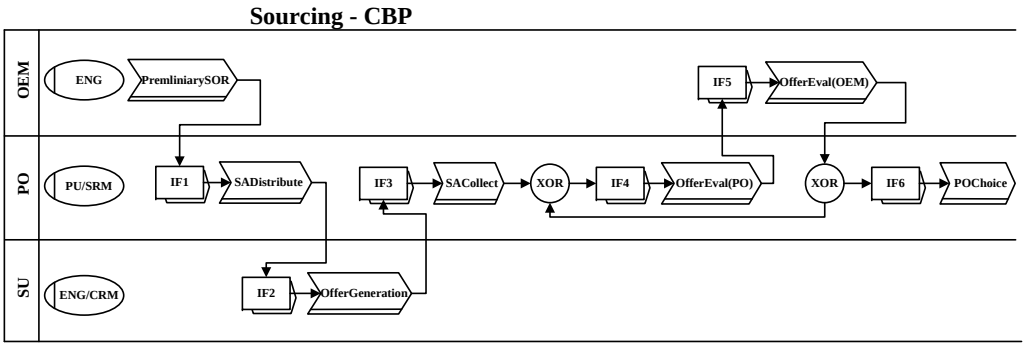


Figure 4.6: Case study: process overview

The CBP starts with the process *PremiliarySOR* conducted by the OEM. After the construction of the new automobile type is finished the requirements for the production are identified in *PremiliarySOR*. The result of *PremiliarySOR* is a Statement of Requirements (SOR). The OEM makes the SOR available to the PO, which conducts the process *SourcingAction*. *SourcingAction* consists of two subprocesses, *SADistribute* and *SACollect*. In *SADistribute* an action plan is generated out of the SOR, containing information about the parts which have to supplied be the suppliers (e.g. object descriptions, types, maximal costs or the time, in which the parts have to be made available by the supply chain). This is followed by the generation of offer requests out of an action plan and the sending of the offer requests to appropriate SUs. A SU evaluates in the process *OfferGeneration* to which price it can supply certain parts. The SU creates an offer and sends it back to the PO. In the process *SACollect* the PO collects the incoming offers and saves these in a uniform format in a database. After all offers have arrived, the evaluation of these offers starts. The evaluation is conducted simultaneously by the OEM and the PO. The OEM checks the offers in the process *OfferEval(OEM)* from a technical point of view that is e.g. whether the quality of the supplied parts is sufficient for the production. The PO analyzes the offers in the process *OfferEval(PO)* from an economic point of view, like which combinations of suppliers are possible or from where the supplier parts have to be procured. Finally the PO compares its results with

the results of the OEM and starts the process *POChoice*. In *POChoice* those SUs are chosen with which contracts shall be placed.

The example of this case study is used in the next section to illustrate the transformation of high-level CBP descriptions modelled in CIMs to service-oriented PIMs.

### 4.1.4 Implementation and Execution CIM to PIM Model Transformations

This section presents the implementation of the software architectures coordination patters, which we introduced in Section 4.1.2, in model transformations. These model transformations take EPCs as input and produce service-oriented models (PIM4SOA) as output.

#### Brokerless Architecture

Using the brokerless architecture (see Section 4.1.2 and Figure 4.3), ESs are derived from the CBP descriptions that interact through collaborations. These binary collaborations are composed to more sophisticated collaborations and protocols. We have described and implemented the transformations rules for the brokerless architecture in [248] for EPC to BPDM model transformations. This version of BPDM [98, 218] is a predecessor of PIM4SOA. Hence, we will describe the transformation process for PIM4SOA via the *Sourcing* process of the case study's example.

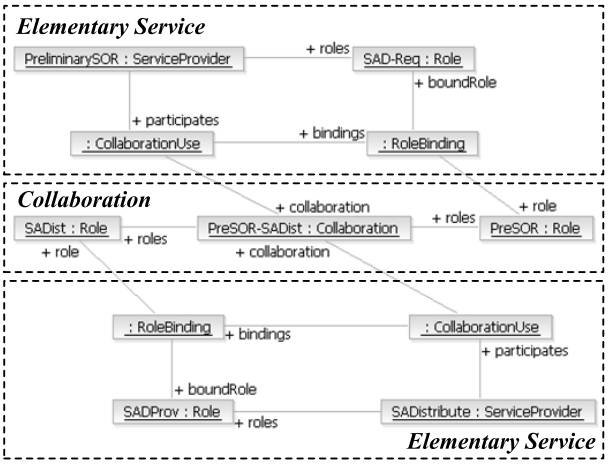


Figure 4.7: PIM4SOA model for brokerless architecture

From each PP of the *Sourcing*-CBP description, one ES is derived. As one can see in Figure 4.7 two service provider instances for *PreliminarySOR* and *SADistribute* are generated. For each pair of service providers that communicate, a collaboration is instantiated (*PreSOR-SADist*). The service providers participate in this collaboration via collaboration uses.

The collaborations of the ESs are realized via multiple binary collaborations. To avoid loss of information (especially about the complete *Sourcing* CBP description), these collaborations are grouped to composite collaborations (see Figure 4.8). In the example, the *Sourcing* composite collaboration is composed of the binary collaboration patterns between the ESs.

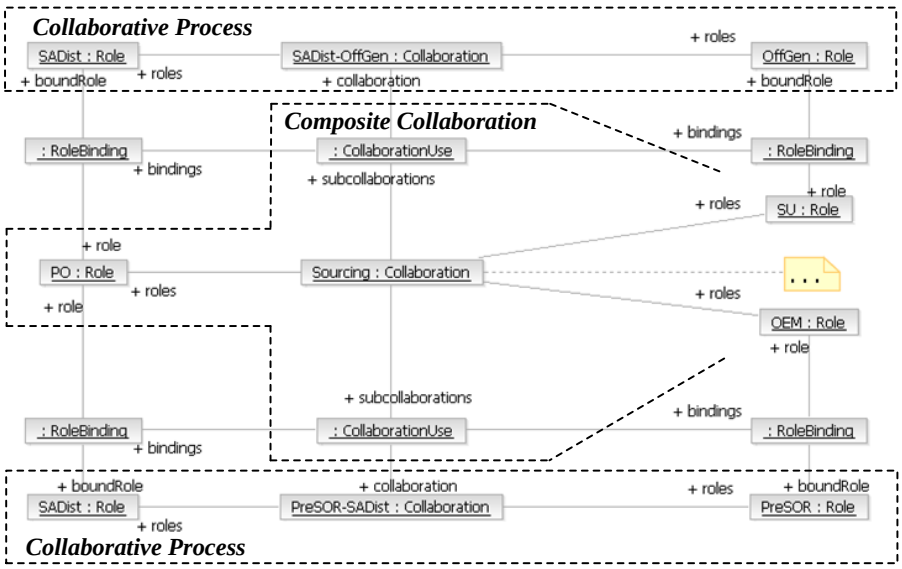


Figure 4.8: PIM4SOA model using composite collaboration

### Central Broker Architecture

Like for the brokerless architecture, we encoded the central broker architecture in transformations (cp. Section 4.1.2) from EPC to BPDM models (see [24, 25]). In this section we illustrate the transformation process via a PIM4SOA model that is generated from the *Sourcing* CBP model of the case study's example.

The central broker architecture (see Figure 4.4) is realized by a coordinating executable broker process and several ESs in PIM4SOA. Like in the brokerless architecture, for each PP of the *Sourcing* CBP description one ES is generated, e.g. for the processes *PremliniarySOR* and *SADistribute*. These ESs do not directly communicate but exchange messages with a central broker component. The ESs communicate with a broker service provider via separate collaborations (*PreSOR-Borker* and *SADist-Broker*). For the *Sourcing* CBP one executable service provider is instantiated that fulfils the centralised broker's functionality (see the *SourcingBroker* in Figure 4.4). This broker service provider implements the coordination of the message exchange described by the *Sourcing* CBP description.

### Decentral Broker Architecture

To represent the decentral broker architecture (see Figure 4.5) with PIM4SOA models, one has to make use of the CBP extension of PIM4SOA described in Section 2.2.3. We present a model transformation from EPC cross-organisational business process description to PIM4SOA. This model transformation realizes and implements the decentral broker architecture [26]. It is described by rules consisting of source and target patterns.

**Transforming the CBP Structure** The following rules (Rule 1.1-1.3) describe the procedure that transforms the structural part of the CBP description to PIM4SOA models, e.g. the definitions of the ESs.

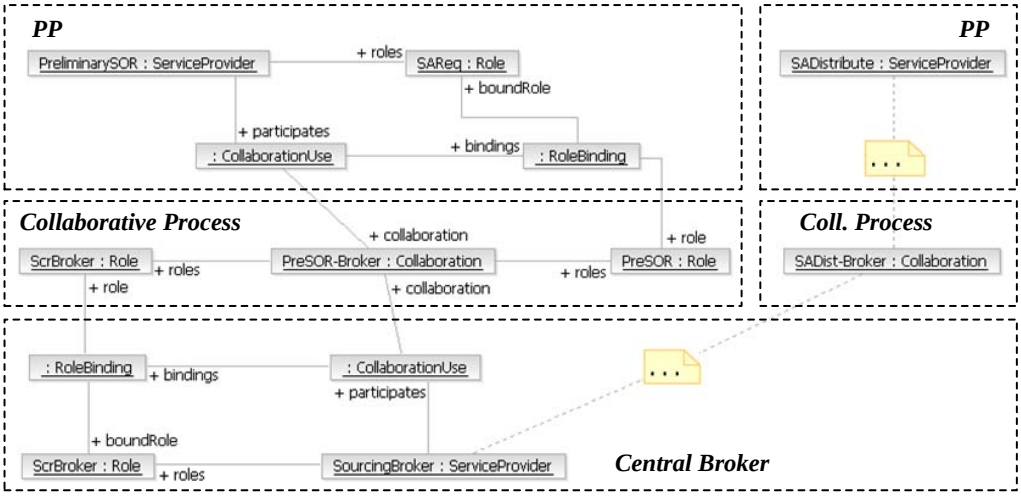


Figure 4.9: PIM4SOA instance central broker

***Rule 1.1***

**Src:** A CBP is modelled in an EPC with row display containing a swimlane concept.

**Trg:** A collaboration process, which is an abstract service provider, is instantiated.

***Rule 1.2***

**Src:** The EPC describing the CBP is structured by swimlanes separating the process modules of the different participants.

**Trg:** For each swimlane a VP is instantiated and connected to the collaboration process it participates in. The name of the VP, i.e. of the service provider, is the name of the department participating in the CBP and realizing roles of the collaboration.

***Rule 1.3***

**Src:** In the case the source and target process module of a control flow edge lie in different swimlanes, there is a collaboration between the two roles represented by swimlanes.

**Trg:**

**(a)** For each pair of roles that collaborates according to the source pattern, one collaboration and the two collaborating roles are instantiated. The two roles are assigned to the collaboration. For each pair of role that collaborates, one and only one collaboration is instantiated.

**(b)** Two VPs, which were derived from the swimlanes (Rule1.2), belong to the two roles participating in the collaboration. For each of the two VPs a collaboration use is instantiated referencing the collaboration.

**(c)** Bindings are instantiated that specify for the collaboration uses, which roles of VPs (boundRole) realize the roles of the collaboration (role).

In Figure 4.10 one can see the target PIM4SOA model that is generated by applying the transformation rules 1.1-1.3 to the EPC model of the case study example (cp. Section 4.1.3). A collaboration process is instantiated for the *Sourcing* CBP modelled in Figure 4.6. The collaboration process is an abstract service provider. By applying Rule 1.2 three VPs are generated, one for each organisation that takes part in the CBP, i.e. the engineering department of the OEM (*Org1-ENG*), the supplier relationship management of the PO, which is often the OEM (*Org2-PU/SRM*), and the customer relationship management of the SU (*Org3-ENG/CRM*).

An association connects the view process (+*views*) to the collaboration process (+*collaboration*). For each pair of roles that collaborate in the *Sourcing* CBP, one collaboration and one role for each of the collaborating roles are instantiated for the PIM4SOA model (Rule 1.3a); in Figure 4.10 these are the roles *OEM* and *PO*. The participation of

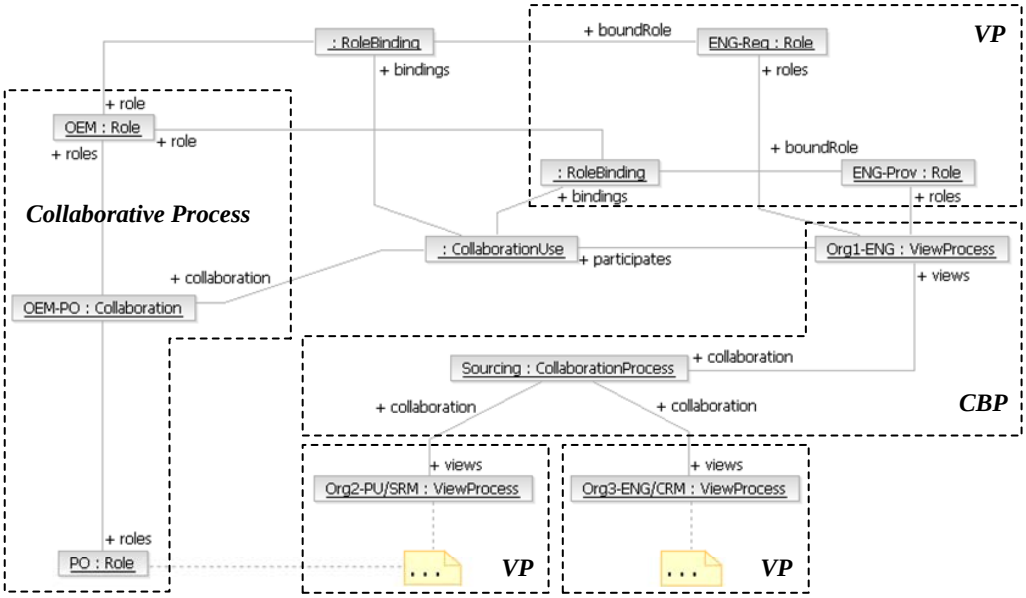


Figure 4.10: Application of transformation rules 1.1-1.3

the organisations' departments in the collaboration is represented by collaboration uses that connect the VPs with the collaboration (Rule 1.3b). A binding (Rule 1.3c) is used to specify by which role (+*boundRole*) a service provider realizes a role (+*role*) (*OEM*) in collaboration.

Considering the decentral broker architecture depicted in Figure 4.5 the collaboration process of the PIM4SOA CBP-extension is a collaborative process as defined in Section 2.2.2. It represents the protocol description (message exchange) between the publicly visible abstract processes. These abstract processes are realized by the VPs that are executable service providers. The collaboration process is an abstract service provider and groups the VPs that belong to one CBP.

**Transforming the CBP Behaviour** Rule 2 describes the transformation of the behavioural part of the CBP description to PIM4SOA models, i.e. the process flow of the CBP.

***Rule 2***

**Src:** A CBP is modelled in an EPC with row display based on a swimlane concept.

**Trg:** For each VP, which has been derived from the CBP, a process is instantiated describing the VPs behaviour. The control flow of the CBP description can be taken over with a few modifications to the VP's behaviour description:

1. Each process module, belonging to the swimlane of the VP, is replaced by a view task.
2. 'Send' and 'receive' tasks are added to the control flow of the VP at the points, where the VP interacts with other VPs (i.e. where the source and the target process module of a control flow edge lie in different swimlanes).
3. The 'send' and 'receive' tasks reference the appropriate collaboration use with their collaborationUsePath. With the interactions represented by the tasks the VP takes apart in the collaboration.
4. All process modules which do not belong to the swimlane of the VP are removed from the process flow.
5. All interfaces are removed from the process flow.

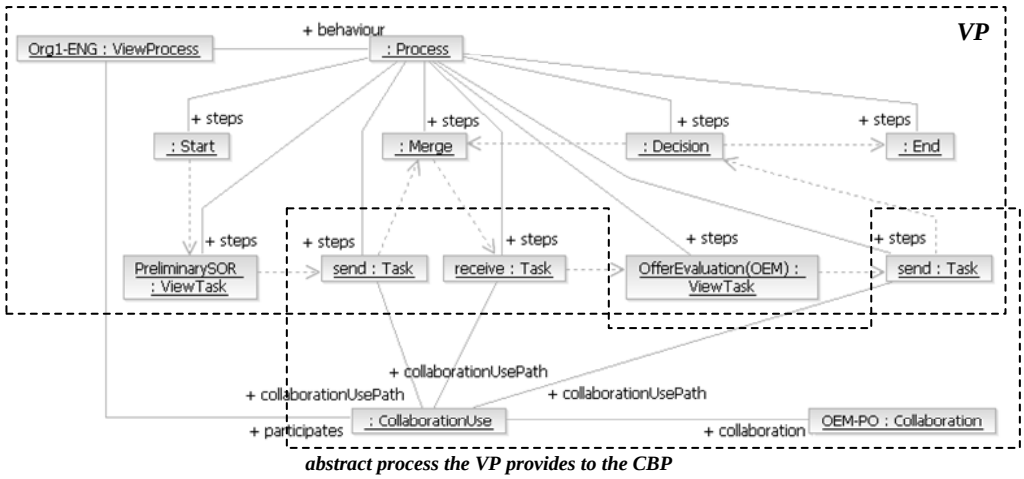


Figure 4.11: Application of transformation rule 2

Figure 4.11 shows the generation of VPs' behaviour description at the example of the *Org1-ENG* view process. The behaviour of the view process, i.e. the service provider, is described by a process. This process consists of steps which are derived from the *Sourcing* CBP according to the algorithm described in Rule 2. Two view tasks *PreliminarySOR* and *OfferEvaluation(OEM)* are instantiated for the corresponding process modules of the *Sourcing* CBP. Since the *Org1-ENG* VP has three interactions with other VPs, two times it invokes another process and one time it is invoked, two *send* and one *receive* tasks are added to the control flow of the VP that participates in the collaborations. In case of the *Org1-ENG* VP all tasks reference the same collaboration use, since the VP only participates in the *OEM-PO* collaboration. In Figure 4.11 the control flow is depicted in a simplified version for clarity reasons as arrows with dashed lines. It shows the complete description of the VP's executable process. Those parts of the executable process that are relevant for publicly visible abstract process, are bound to the respective collaborative process (see Figure 4.10).

**Connecting Private Processes to View Processes** Rule 3.1 and Rule 3.2 connect the PPs and the VPs that are generated from ESs. This has to be done both for the structural and the behavioural part of the PIM4SOA model.

***Rule 3.1***

**Src:** VPs abstract from PPs and PP offer functionality to cross-organisational collaborations with the help of VPs.

**Trg:** A VP references all PPs it abstracts. A PP references all VP over which it offers func-tionality to cross-organisational collaborations.

***Rule 3.2***

**Src:** View tasks are used to describe the behaviour of a VP. They encapsulate tasks, which describe the more detailed behaviour of PPs.

**Trg:** For each view tasks all tasks are reference the view task encapsulates.

The *Org1-ENG* VP is bound to two processes, *PreliminarySOR* and *OfferEvaluation(OEM)*, by applying Rule 3.1 (see Figure 4.12). In addition the view tasks of the VP's behaviour description reference the tasks of the PPs they abstract from (abstract-

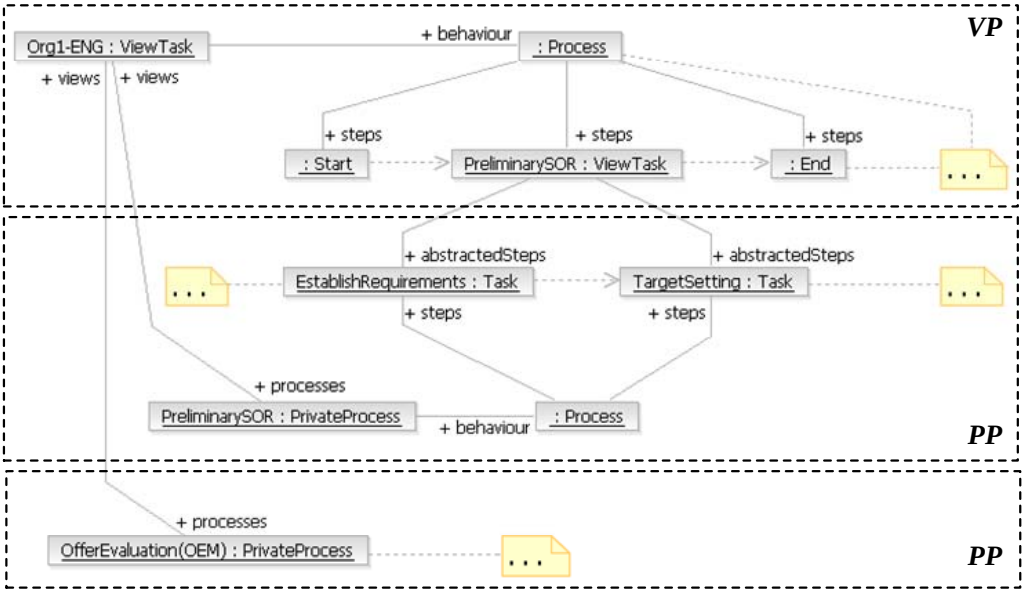


Figure 4.12: Application of transformation rules 3.1-3.2

edSteps). For example the *PreliminarySOR* view tasks abstracts from the *EstablishRequirements* and *TargetSetting* tasks.

### 4.1.5 Discussion

In this section we have developed and compared potential target architectures of a model-driven transformation of CBPs from the computational independent down to the platform independent level. We showed how the information necessary to automatically create platform independent ICT-level models for a service-oriented environment can be derived from high-level business process models. We identified a number of modelling constructs allowing us to derive platform-independent architectures that can be mapped to different ICT architectures.

Having implemented both central, decentral and brokerless approaches, the major insights we gained are as follows: While all three approaches can be derived from a CIM description without an explicit description of the CBP, we found it important that CBPs be explicitly modelled; otherwise, model transformation results are likely to be of poor quality. The decentral broker architecture relies on the existence of a CBP model to a higher degree than the central broker architecture and the brokerless architecture do: the latter can be derived more easily from the process flow; in the former, the appropriate grouping of processes to view processes in a decentral broker must be specified explicitly.

## 4.2 Transforming PIM to PSM

In MDSD platform knowledge is encoded in PIM to PSM transformations. In the context of business process development and enactment the platform is given through the respective process execution environment. In this section we present a model and code generation framework, that fosters the generation of executable workflow code from

higher-level processes. Its main achievements are to improve the reuse and composition of parts of generation solutions and to decouple domain aspects from computational aspects in workflow code generation. The generation framework aims to enable people with no or little experience in code generation and workflow technology to generate workflow code from higher-level models in reasonable time.

After the introduction of an example and context information in Section 4.2.1, we describe the main problems one meets by generating executable workflow code from higher-level process models (see Section 4.2.2). The model and code generation framework solves these problems separately and integrates the respective solutions in a single framework (see Section 4.2.3). The case study in Section 4.2.4 shows the application of the workflow code generation to the example introduce in Section 4.2.1.

### 4.2.1 Context and Example

Before we provide the concrete problem description in Section 4.2.2, we present an example for the integrated application of process modelling and workflow execution techniques. In the AgilPro project we applied process modelling, workflow execution, and service-oriented concepts to support agile business process through flexible, service-oriented IT systems.

During the last decade in the Enterprise Resource Planning (ERP) domain agile processes got more and more important. In order to improve existing products or customize them to the needs of the end-user, most changes need to be done on the flow of services but not on the offered services. This results from changes in jurisdiction, new products, standards, or requirements of the customers. In most organisations there is a need for loosely-coupled components. SOA and Web service technology as an implementation of SOA are one way for achieving this. AgilPro is a tool-suite and process integration framework based on SOA. It allows the user to model their business processes, preview and execute them on a process engine.

The core of the AgilPro solution is a Domain Specific Model (DSM) that conforms to a DSL for process modelling and contains predefined model elements (e.g. applications, services, etc.) for a particular domain. The AgilPro modelling tool further provides the possibility to define multiple concrete syntaxes, for example one that especially suits the ERP domain. The AgilPro modelling tool offers (at least) two views on the DSM, a business view and a technical view. The business view abstracts from technical details such as which web services are invoked, how the data mapping between different applications works, etc. This is part of the technical view where an IT expert can specify the relevant data for an execution of the process - if not already predefined in the DSM. The model and code generation framework is used to generate executable workflow code.

The AgilPro metamodel is graph-based as most business process languages and rests upon the UML 2 metamodel for activity diagrams. It extends it with information about responsibilities or functions like in Architecture of Integrated Information Systems (ARIS) [261] or data and events similar to BPMN [226]. Henceforth, it tries to combine the best practices of the currently existing process modelling languages. To enable domain-specific modelling for e.g. ERP, CRM, or financial service applications, the metamodel and the AgilPro modelling tool allows to define model templates. These model templates contain predefined elements of and information about the application domain, but also syntactical information. These are for example data types or applications with specific (execution) information or icons from the ERP domain. These prede-

fined elements become automatically part of AgilPro modeller's modelling palette and complete together with the AgilPro metamodel the specific DSL. Since such DSLs cannot be executed on current process engines directly, one needs to transform them to an executable language. We use BPEL4WS [129], which is the quasi standard for orchestrating web services and supported by several process engines, for illustration purposes in the rest of this section.

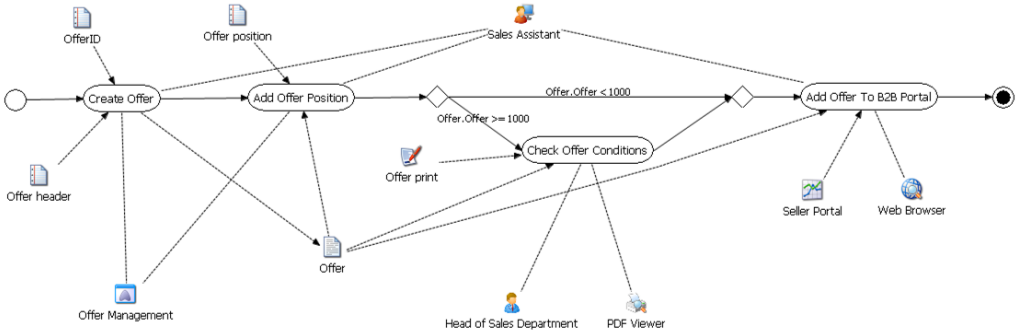


Figure 4.13: Create offer process modelled with AgilPro Light Modeller

Figure 4.13 shows a *Create Offer* process that is modelled with the AgilPro Light Modeler (LiMo). A complete process model comprises processing steps, input and output data, applications that are used to execute the processing steps, and roles that perform the processing steps. In the *Create Offer* process the processing step *Create Offer* has an *OfferID* and *Offer header* as input data and produces an *Offer* as output. *Create Offer* is performed by the *Sales Assistant* and makes use of the *Offer Management* system for execution. If the offer has a value that is greater than or equal to 1,000, the *head of the sales departmen*t has to check the offer. Finally, the offer is added to a Business-To-Business (B2B) portal by the *sales assistant*. What is not shown in the process diagram is the additional information of the predefined elements that complete the DSL. In the *Create Offer* process, these are for example the attributes of the data types like *'dioParameter'* for *OfferID* or information about the AgilPro integration framework Java adapters, which are called for the applications like *'eu.emundo.agilpro.fw.fe.intf.GenericUi'* for *Offer Management*. This additional information is used by the code generation to directly generate executable BPEL4WS code.

### 4.2.2 Problem Description

To develop a code generation that generates executable workflow code directly from processes described at a higher abstraction level like in PIMs, one has to deal with a variety of requirements and challenges.

#### Process (Graph) Transformation

An important challenge is the translation of the higher-level processes into constructs that are provided through the target process execution language. WS-BPEL for example is a so-called block-structured language. There, language elements that represent the control flow are composed in a cycle-free tree-structure without goto-statements. [181, 241] describe approaches of how to translate process graphs into block-structured WS-BPEL code. The process graph has to be analyzed and even restructured in order to map

the graph to the WS-BPEL elements. These approaches are based on the identification of Single-Entry-Single-Exit Component (SESE) components in the control flow. To identify SESEs, algorithms like [142] from compiler theory or the token flow algorithm [112, 226] can be used.

As a result, one has to implement quite complex graph transformation algorithms to realize (e.g. WS-BPEL) workflow generation from higher-level process graphs. It is certainly possible to implement these transformations (some good examples can be found in [241]) with nearly any model-to-model or model-to-code transformation approach. However, most model-to-model and model-to-code transformation approaches qualify themselves a lot better for describing relationships between elements and implementing generation patterns than for implementing complex graph transformation algorithms. To realize preprocessing and graph transformation algorithms like [112, 142, 241] common programming languages like Java or C are better suited.

#### Usage of Process Execution Environments and Engines

Though process execution languages have a well-defined syntax and semantics, different process execution programs can be used to achieve the same external behaviour (effects) of a process (execution) engine. Hence, code generation templates have to be adjusted in the way process engines are used and implement the particular execution patterns. Moreover, code generation templates also encode domain knowledge, like the specific data types. The following examples illustrate two different ways of using BPEL4WS to execute the invocation of a *CreateOffer* service. While the first example represents code generation from process models whose semantics is similar to BPEL4WS code, the second example demonstrates code generation from higher-level process models. The second example demonstrates quite well, that for a range of realistic application scenarios sophisticated execution and invocation patterns have to be encoded into code generations.

**Example 1** The BPEL4WS generation developed in the Service Modeling Language for MID innovatorAOX (SPL4AOX) project [235, p.78ff] was based on the action semantics of UML. Similar BPEL4WS code generations have been described for example in [129]. Listing 4.1 depicts the BPEL4WS code that is generated to invoke the *CreateOffer* service, that gets an *OfferID* as input and provides an *Offer* as output (see Figure 4.13 in Section 4.2.1).

Listing 4.1: BPEL code generated in SPL4AOX

```
1 <invoke name="CreateOffer"
2   partnerLink="CreateOffer_Prov" portType="CreateOffer"
    operation="in" inputVariable="OfferID" outputVariable="Offer">
4 </invoke>
```

**Example 2** For our usage of the JBoss workflow engine in the AgilPro project multiple BPEL4WS instructions are necessary to obtain the same computational result as in example 1. Listing 4.2 depicts the sample code that is necessary to invoke the *CreateOffer* service depicted in Figure 4.13. The full version of the BPEL4WS code can be found in Appendix A. One important issue to recognize is, that the process execution engine has an execution context in which information about the process execution can transiently

be stored. *Variables* like *nextActionReq* are containers in this execution context which consist of attributes (*parts*) like *ticketnumber* or *NameIN*. We use the *ticketnumber* for correlation in the JBoss engine. *DataTypeIN*, *ValueIN*, and *NameIN* contain the information that is also used by the data object of the AgilPro integration framework. The code of Listing 4.2 is generated as follows:

1. Before the processing step *CreateOffer* is started, the input data *OfferID* is copied to the execution context. This is done by an assign and an invoke for each input data (line 193-226).
2. In the lines 227-240 the execution of the processing step *CreateOffer* is started.
3. The receive statement (line 241-246) waits for the completion of the processing step, which can also be human interaction input. Line 247-259 stops the task execution in the process engine.
4. Finally, the result data *Offer* (line 260-290) is fetched from the process execution context.

Listing 4.2: BPEL code generated in AgilPro

```
196 <assign name="set_DTO_OfferID">
      <copy>
198     <from part="Ticketnumber" variable="nextActionReq"/>
        <to part="Ticketnumber" variable="setValueToObjectReq"/>
200   </copy>
      <copy>
202     <from expression="string('ID')"/>
        <to part="DataTypeIN" variable="setValueToObjectReq"/>
204   </copy>
      <copy>
206     <from expression="string('0')"/>
        <to part="ValueIN" variable="setValueToObjectReq"/>
208   </copy>
      <copy>
210     <from expression="string('OfferID')"/>
        <to part="NameIN" variable="setValueToObjectReq"/>
212   </copy>
      <copy>
214     <from expression="number('-1')"/>
        <to part="HashCodeIN" variable="setValueToObjectReq"/>
216   </copy>
    </assign>
218 <invoke name="setValueToObject_OfferID"
        portType="agi:AgilproIssuer"
220     operation="setValueToObject"
        partnerLink="agilpro"
222     nputVariable="setValueToObjectReq">
      <correlations>
224     <correlation pattern="out" set="atmInteraction"/>
      </correlations>
226 </invoke>
    <assign name="startAction_CreateOffer">
```

```
231   ...
232 </assign>
    <invoke name="startAction_CreateOffer"
234     portType="agi:AgilproIssuer" operation="startAction"
        partnerLink="agilpro" inputVariable="startActionReq">
236   <correlations>
        <correlation pattern="out" set="atmInteraction"/>
238   </correlations>
    </invoke>
240 <receive>
        partnerLink="atm" variable="nextActionReq"
242     portType="atm:FrontEnd" operation="nextAction">
      <correlations>
244     <correlation set="atmInteraction"/>
      </correlations>
246 </receive>
    <assign name="endAction_CreateOffer">
```

```
251   ...
252 </assign>
    <invoke name="endAction_CreateOffer"
254     portType="agi:AgilproIssuer" operation="endAction"
        partnerLink="agilpro" inputVariable="endActionReq">
256   <correlations>
        <correlation pattern="out" set="atmInteraction"/>
258   </correlations>
    </invoke>
260 <assign name="get_DTO_Offer">
```

```
279   ...
280 </assign>
    <invoke name="getValueFromObject_Offer"
282     portType="agi:AgilproIssuer"
        operation="getValueFromObject"
284     partnerLink="agilpro"
        inputVariable="getValueFromObjectReq"
286     outputVariable="getValueFromObjectRes">
      <correlations>
288     <correlation pattern="out" set="atmInteraction"/>
      </correlations>
290 </invoke>
```

#### Requirements and Challenges Summary

Most people and organisations aiming to develop model or code transformations only want to be concerned with those parts of their solution that are really specific to their usage scenario. They want to be able to reuse parts of existing solutions and compose them with minimal effort. As described in the previous sections, there arise a variety of challenges when people want to derive executable process descriptions (code or models) from higher-level process models. In Section 4.2.3 we present a model and code generation framework that allows us as far as possible to address the various challenges separately. The following list summarizes the requirements for the framework.

- Since there exists a huge diversity of modelling languages and projects providing means to model processes (like UML activities, BPDM, PIM4SOA [235, p.51ff],

AgilPro, etc.), the framework shall allow to decouple code generation from the format of the input models.

- It is necessary to apply more or less complex graph transformation algorithms to translate higher-level process models to workflow executable code like WS-BPEL or XPDL [319].

- Depending on the process execution environment the code generation has to encode complex invocation patterns that include knowledge about the respective workflow execution engine. The framework has to provide means to easily describe, maintain and reuse these generation patterns independent of other information that is necessary for the code generation.

### 4.2.3 Model and Code Generation Framework

To deal with the challenges described in Section 4.2.2 we applied the separation of concerns paradigm to the model and code generation framework. Solutions of the described challenges can be integrated in the framework from separate components, which overlap as little as possible. This allows flexible reuse and combination of components in the model and code generation framework.

We made the observation that solutions for some of the described challenges highly depend on the application domain, the modelling context, and the execution environment, while others are independent of the application domain. Hence, we introduced a common process modelling format and divided the framework into a domain-specific and a domain independent part like in compiler theory [6], where an intermediate language is used to allow language independent code optimization. However, it is not totally correct to identify a front end and a back end like in compiler construction, since the front end (the *adapter for DSL model (I)*) and the code generator (the *generation templates (IV)*) both depend on the DSL specific process format and thus are not independent from each other. Figure 4.14 depicts a structural view on the generation framework.

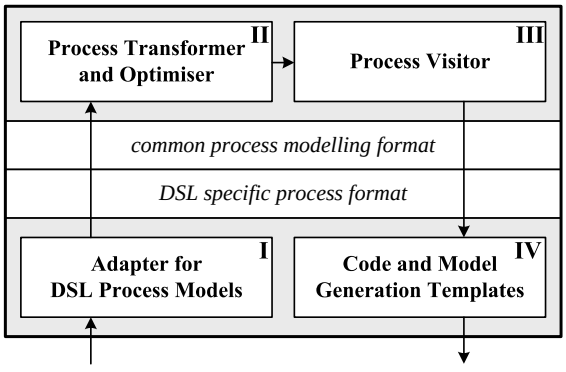


Figure 4.14: Model and code generation framework

- The *process transformer and optimiser* (II) and the *process visitor* (III) are domain independent. They address general graph transformation problems and graph traversing independent to any concrete process modelling languages. These components access process descriptions that are represented in a common process

modelling format. The framework makes use of a process modelling format that was derived from the Standard Workflow Models [156]. For block-structured graphs it provides a common process modelling format that is based on WS-BPEL.

- The other two components of the framework, the *adapter for DSL process models* (I) and the *code and model generation templates* (IV), directly access the process modelling format of the DSL that is used for modelling the input model. Hence, adapters and generation templates have always to be used in combination, i.e. they must use the same DSL specific process format.

The model and code generation framework not only allows to plug together components via common interfaces, but also provides a workflow that composes these components. Once the framework is configured, i.e. the components are registered and plugged in, the user only has to provide the input model and start the generation workflow of the framework. Figure 4.15 shows this generation workflow. It depicts the four states of the workflow execution (a)-(d) and the transitions.

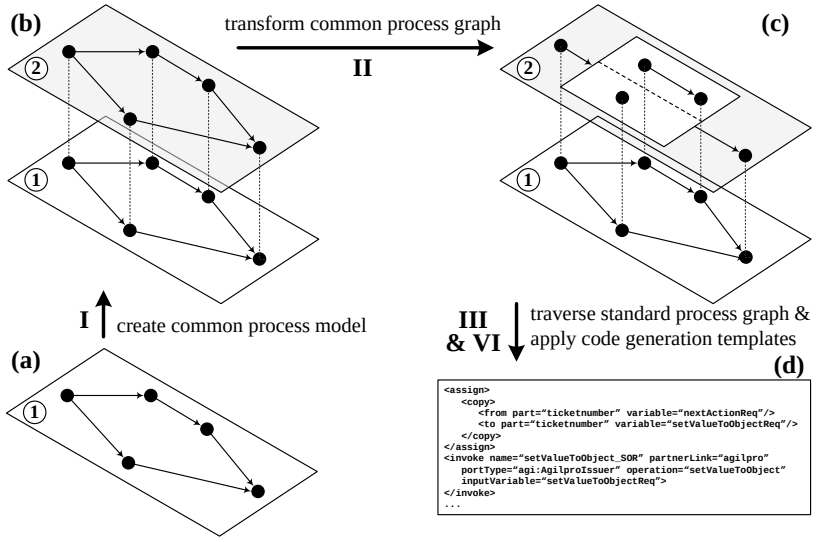


Figure 4.15: Code generation

(a) The first state of the workflow comprises the input process model that is represented in a format specific to the DSL used for modelling the input process.

(b) The second state is reached by applying the adapter on the input model. The adapter creates a representation of the input process in the common process modelling format. To ensure traceability between the processing steps of the two process models, it further links the processing steps in the common representation format to the processing steps of the input model.

(c) The process transformer restructures and optimizes the process represented in the common process modelling format. For example it generates a block-structured graph. Though in state (c) the control flow of the two process representations differs, their processing steps are still linked to the respective processing steps of the other representation format.

(d) The last transition is used for model and code generation and ends in the final state (d) of the framework's workflow. Therefore the process visitor traverses the process of state (c) that is represented in the common process modelling format. The code or model generation templates are called via a notification mechanism provided by the framework.[2] Like we can see in Figure 4.15 the workflow terminates with state (d) when the code or a new model was generated.

The main power of the model and code generation framework lies in the *process transformer and optimiser (II)* and the combination of a visitor-based and a template-based code generation approach (*(III) and (IV)*). (II) identifies SESEs in the process descriptions [112, 142]. SESEs are the basis for graph transformations that allow to generated block-structured (WS-BPEL) code [241]. SESEs are also used to test the soundness of the process's control flow in reasonable time [294]. (III) and (IV) combine the advantages of visitor-based and template-based code generation approaches (see also Section 3.3.2). The process visitor traverses the process flow of the input model and calls templates for workflow code generation. The visitor allows to generate the workflow code in the sequence that is given be the process's control flow. This is especially important for e.g. *Sequences* in WS-BPEL, where the process steps are performed according to the order they have in the WS-BPEL text file. The template mechanism has the advantage that generation templates can easily be derived from examples [65].

The framework allows graph transformation from flexible higher-level process descriptions into constructs that are provided through the target process execution language and supports users to easily implement code generation for process execution via complex invocation patterns.

### 4.2.4 Case Study

This section presents a case study that illustrates the code generation for the *Create Offer* process introduced in Figure 4.13 of Section 4.2.1.

#### Configuration of the Generation Framework

To generate BPEL4WS code the generation framework has to be configured first. Hence, the process transformer that transforms arbitrary processes into BPEL4WS and the process traverser that can process common block-structured process models are registered at the framework. While the process traverser is already provided by the framework, we use the Token Analysis component [274] as a process transformer. The adapter for AgilPro LiMo models and the respective BPEL4WS code generation templates for the AgilPro JBoss workflow engine (jBPM) are also registered at the framework. Now, the workflow of the generation framework can be executed as shown in Figure 4.15.

#### Creation of Common Process Model

In the first step of the framework's workflow, the adapter for AgilPro LiMo process models generates a representation of the input process in the common process modelling format. The resulting process is depicted in Figure 4.16 in UML concrete syntax (the common process modelling format itself has no concrete syntax representation). Further the adapter connects the respective processing steps of the two process representations.

[2] The framework implements the notification mechanism with the publish-subscribe pattern [101].

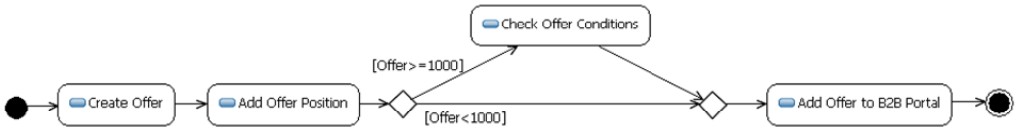


Figure 4.16: Create offer process as common process model

### Transformation of Common Process Graph

The process transformer transforms the process generated by the adapter into a block-structured process. It implements the algorithm described in [112]. Figure 4.17 depicts this block-structured process. As we can see, an alternative block was generated from the decision and merge gateways.


Figure 4.17: Block-structured create offer process as standard process model in UML syntax

### Generating the BPEL Code

In a last step, the block structured process is traversed and the BPEL4WS code generation templates are applied. In the following we present some code excerpts generated by the code generation for the *Create Offer* process. The full version of the generated BPEL4WS code can be found in Appendix A.

Before the process visitor starts, statical information like *partnerLinks*, *variables*, and *correlationSets* is generated for the BPEL4WS process. Further, some initial invocations on the process engine are made (see line 1-129 in the BPEL4WS process in Listing A.1). Then the visitor starts to traverse the block-structured process description. For each node a *scope* is generated that gets the name of the node and contains the further processing instructions that are necessary for this node.

The first node the visitor accesses is the start node. For this node an *empty* process instruction is generated within a *scope*. Since the start node has no name the scope gets the default name (see code excerpt in Listing 4.3).

Listing 4.3: Generated BPEL code for start node

```
130 <scope name="Defaultname">
      <sequence>
132     <empty/>
      </sequence>
134 </scope>
```

Next, a *scope* that contains the further processing instructions for the *Create Offer* node is generated in the BPEL4WS code (see line 135-292 in Listing A.1). The process execution engine has an execution context in which information about the process execution can transiently be stored. The input data of each process node has to be copied

to the execution context. Listing 4.4 depicts respective BPEL4WS code for the *OfferID* input of the *Create Offer* step.

Listing 4.4: BPEL instructions for *OfferID* input data of *Create Offer* node

```
196 <assign name="set_DTO_OfferID">
      <copy>
198     <from part="Ticketnumber" variable="nextActionReq"/>
        <to part="Ticketnumber" variable="setValueToObjectReq"/>
200   </copy>
      <copy>
202     <from expression="string('ID')"/>
        <to part="DataTypeIN" variable="setValueToObjectReq"/>
204   </copy>
      <copy>
206     <from expression="string('0')"/>
        <to part="ValueIN" variable="setValueToObjectReq"/>
208   </copy>
      <copy>
210     <from expression="string('OfferID')"/>
        <to part="NameIN" variable="setValueToObjectReq"/>
212   </copy>
      <copy>
214     <from expression="number('-1')"/>
        <to part="HashCodeIN" variable="setValueToObjectReq"/>
216   </copy>
    </assign>
218 <invoke name="setValueToObject_OfferID"
        portType="agi:AgilproIssuer"
220     operation="setValueToObject"
        partnerLink="agilpro"
222     inputVariable="setValueToObjectReq">
      <correlations>
224     <correlation pattern="out" set="atmInteraction"/>
      </correlations>
226 </invoke>
```

To process the *Create Offer* step, again several BPEL4WS instructions are necessary. In line 227-239 the execution of *Create Offer* is started and the receive statement in line 240-246 waits for a notification that the processing step is completed. Finally the execution of *Create Offer* is stopped (line 247-259). Listing 4.5 shows some excerpts of the necessary processing instructions. Finally the result data *Offer* (line 260-290) is fetched from the process execution context.

Listing 4.5: BPEL instructions for execution of *Create Offer* node

```
233 <invoke name="startAction_CreateOffer"
234     portType="agi:AgilproIssuer" operation="startAction"
        partnerLink="agilpro" inputVariable="startActionReq">
236   <correlations>
        <correlation pattern="out" set="atmInteraction"/>
238   </correlations>
    </invoke>
240 <receive
        partnerLink="atm" variable="nextActionReq"
242     portType="atm:FrontEnd" operation="nextAction">
      <correlations>
244     <correlation set="atmInteraction"/>
      </correlations>
```

```
246 </receive>
    ...
```

```
252 ...
    <invoke name="endAction_CreateOffer"
254     portType="agi:AgilproIssuer" operation="endAction"
        partnerLink="agilpro" inputVariable="endActionReq">
256   <correlations>
        <correlation pattern="out" set="atmInteraction"/>
258   </correlations>
    </invoke>
```

Another interesting part of the generated code is how the alternative is translated into BPEL4WS code. The block-structure that represents this alternative in the process flow is mapped onto a switch-statement in the process code. Listing 4.6 depicts the respective BPEL4WS code.

Listing 4.6: BPEL instructions generated for processing alternative

```
458 <switch name="Defaultname">
      <case condition="bpel:getVariableData('checkGuardRes',
460                                          'ValueOUT')=1">
        <sequence>
462       <scope name="CheckOfferConditions">
            ...
```

```
607         ...
608       </scope>
          <empty/>
610     </sequence>
      </case>
612   <case condition="bpel:getVariableData('checkGuardRes',
                                           'ValueOUT')=0">
614     <sequence>
          <empty/>
616     </sequence>
      </case>
618 </switch>
```

### 4.2.5 Discussion

In the context of executable WS-BPEL workflow code, there exist a variety of solutions that provide hardly more than another concrete syntax (graphical instead of textual) for WS-BPEL (cp. UML profile for BPEL [11], Oracle BPEL Process Manager [240], or the ActiveBPEL designer [179]). These solutions do not narrow the gap between higher-level process descriptions and workflow execution. Tool chains that allow model-driven development and the generation of WS-BPEL code like the IBM WebShpere Tool Suite [128], still have restrictions that prevent all process models from being fully transformed [317, p.109]. To ensure the execution through workflow engines they impose restrictions on the types of modelling elements that can be used, such as loops [320, p.18]. These approaches further require manual model refinement at multiple abstraction levels. [147] describes a generic mapping approach of business process models to other process-oriented representations by the means of XPDL. To our experience a generation of XPDL from higher-level process models does not have to deal with the

same challenges than a generation of WS-BPEL, since the sequence of model elements in a XPDL model does not determine the process's control flow and XPDL does not require block-structured processes.

We have implemented workflow code generations in various projects. In the SPL4-AOX project (see Example 1 in Section 4.2.2) we had an effort of about 2.5 man month to specify and implement BPEL4WS code generation with the oAW Xpand [81, p.87ff] and the oAW Xtend language [81, p.77ff]. The input models for this transformation are constrained to block-structured process models. Our second implementation was done for a prototype of the AgilPro project (see Example 2 in Section 4.2.2). We used JET templates combined with Java code. The solution is able to deal with a limited set of cycles in the control flow. The effort was approximately 1 man month, where most of the time was spent on constraining and transforming the control flow of the input models. However, the parts of this implementation did not lend themselves for reuse, since graph transformation and code generation was combined. Based on these experiences we developed the generation framework and a graph transformation component with an effort of 4 man months[3]. Finally, another person, which *had no experience* with code generation and workflows, implemented the code generation for AgilPro in 3 days with our generation framework. This person simply had to copy the process graph of the input model in the AgilPro adapter and derive the generation templates from examples.

The model and code generation framework allows to achieve time savings compared to other approaches. However, these time savings depend on the complexity of the input graphs (e.g. with or without cycles) and on the experience of the generation developer. Our model and code generation framework makes it possible for people with no or little experience in code generation and graph transformation to produce workflow code at reasonable time. In [253] we describe more experiences about the application of this framework.

## 4.3 Conclusions

In this chapter we applied MDSD to the development of concrete service-oriented implementations via largely automated model transformations. We refined high-level cross-organisational business process descriptions to code that is executable by workflow engines. Model transformations that refine models of three abstraction levels of MDSD have to encode additional information (see Figure 4.1). We identified crucial information for this transformation process and developed solutions that facilitate the generation of executable workflow code from abstract cross-organisational business process descriptions.

For the CIM to PIM transformation we developed software architecture patterns for ICT system generation that can be applied to enact cross-organisational business processes. We implemented these patterns in the CIM to PIM transformations that allow automated generation of service-oriented, platform independent models (PIM4SOA) from computational independent CBP models. Though capturing knowledge about architecture in separate models or implementing them directly in model transformations - a model transformation itself is a model (see Section 3.3) - our solutions stands out from other solutions presented under the MDSD label. It not only converts the representation format of cross-organisational business process from EPC to PIM4SOA and leaves the

[3]The implementation is part of the Workflow Generation Framework [275].

implementation of the software architecture as a manual task to the system architect. Our solutions narrow the gap between high-level business models and ICT system models significantly by encoding software architecture patterns in the model transformations. We develop and implemented such patterns for three different ICT system coordination architectures to enact cross-organisational business processes.

Our solutions of PIM to PSM transformations and code generations also have the principal objective to narrow the gap between the different abstraction levels - executable workflow code is directly generated from higher-level process descriptions. The presented generation framework decouples components that depend on the domain and the execution environment from components that deal with computational aspects like control flow analysis and transformation. This allows better reuse of the knowledge encoded into adapters, graph-transformation algorithms and code generation templates. If e.g. WS-BPEL code in another version (2.0 instead of 1.1) shall be generated from a model, only the respective code generation templates have to be adjusted. The generation framework can be applied to any process modelling language or DSL that is concerned with process description. For the respective models only the respective adapter(s) have to be implemented. The generation framework can even be applied, when the source model does not contain all necessary information to generate executable workflow code. In this case the generation templates are replaced with code that constructs a refined model.

# Chapter 5

# ICT Architectures for CBP Enactment: Applicability Criteria and Evaluation

Peer-to-peer systems, like Multi-agent Systems (MAS) [139], have been proposed in the literature as a suitable architectural and implementation approach for cross-enterprise collaboration [140, 141], due to their support for decentral decision-making and peer-to-peer coordination, loosely coupled coordination, modelling support for the notion of electronic institutions [102], and built-in adaptability. However, different application domains and different market constellations require different types of system architecture.

In MDSD IT architects have to be able to reuse knowledge about ICT solutions by understanding and selecting adequate ICT system architectures in a timely manner. We have made the observation that making the right decision about ICT architecture depends, like organisational structure, on a number of environmental characteristics (called contingencies in [72]).

In this chapter we present our investigations about how these environmental characteristics influence the selection of a most efficient and effective software architecture for ICT system coordination. The contributions of this chapter are as follows:

- A model for decision support suitable for IT architects to derive an appropriate architecture paradigm for a given use case or application domain. The decision model combines the AHP with scenario-based architecture evaluation techniques.
- Scenario descriptions that allow the evaluation and selection of appropriate ICT system coordination architecture paradigms for CBP enactment.
- A set of guidelines of how contingencies influence ICT system coordination architecture based on our experiences in model-driven CBP modelling and enactment.
- We validate our observations (the guidelines) by applying our decision support method, the scenario descriptions for CBP enactment, and the guidelines about contingency influence to application scenarios with differing characteristics.

In Section 5.1 of this chapter we provide an example and problem description for the selection of appropriate ICT system architectures. Section 5.2 presents a method for the evaluation of ICT architecture applicability. In Section 5.3 we apply this method to

the application scenario introduced in Section 5.1. Section 5.4 provides discussion and conclusions.

## 5.1 Example and Problem Description

Let us look at the following scenario of collaborative product development in the automotive sector (see [277]). In Section 4.1.3 we already introduced the *Sourcing* process for this scenario. To choose an appropriate IT architecture, we will have to have a look at the collaboration topology. The scenario (see Figure 5.1) describes the interaction between an Automotive Original Equipment Manufacturer (OEM) and its supplier network consisting of multiple tiers of suppliers, during the process of Strategic Sourcing. Strategic Sourcing is an early step within cooperative product development, where OEMs set up strategic partnerships with the larger (so-called first-tier) suppliers with the aim of producing specific subsystems (e.g. powertrain, safety electronics) of a planned car series. In the use case considered for this chapter, the OEM shares Requests for Quotations with its first-tier suppliers (1). First-tier suppliers serve as gateways to the supplier network; specifications are reviewed and conditions negotiated with second-tier suppliers (2), and feasibility of the requests are checked. First-tier suppliers then issue quotes or suggest changes to the OEM (3). This cycle is repeated until all parties agree on a feasible specification. Finally, first-tier suppliers submit quotes to the OEM.

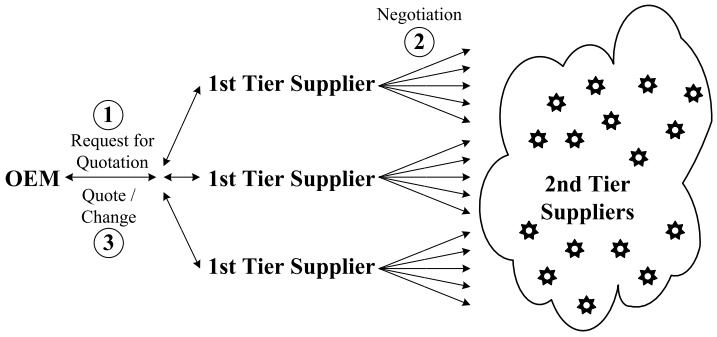


Figure 5.1: Application example: collaborative product development

Figure 5.1 gives a high-level (business-level) model of the CBPs involved in this scenario. Trying to map this model into an executable ICT model, several non-trivial questions need to be answered regarding the ICT architecture. In particular, the question that we address in this chapter is the following: What is the most appropriate architectural choice to model the interactions between the OEM and the 1st tier suppliers? A decentral, peer-to-peer messaging architecture where each role in each process instance is mapped into an agent-like entity to run and control it? An architecture with a central broker (e.g. located at the OEM) that centrally enacts and controls the CBP? Or a mixture of both, a decentral broker architecture where each enterprise provides a publicly visible instance to control and coordinate their business process roles while hiding other, private, elements.

Thus, we can see that there are different architectural choices / paradigms possible for underlying ICT system design. Intuitively, none of these choices is per se better than any other; making the right decision depends on a number of environmental characteristics, i.e. contingencies.

## 5.2 A Method for Evaluation of ICT Architecture Applicability

This section presents an evaluation and decision method that helps to select appropriate ICT architectures for CBP enactment. The evaluation method takes into account the trade-offs between coordination structures, which are implemented by the ICT system architectures in terms of coordination costs and vulnerability costs (see [174]). As visualized in Figure 5.2 the evaluation model distinguishes between quantitative factors, that are measurable by concrete figures (objective factors), and qualitative (subjective factors), which are difficult or impossible to measure. Coordination costs to establish and maintain communication links between collaborating patterns are included as quantitative factors in terms of software, hardware, and labor in the evaluation model. Coordination costs like costs for exchanging messages between collaborating partners are taken into account by qualitative factors. Vulnerability costs, which are "the unavoidable costs of a changed situation that are incurred before the organisation can adapt to a new situation" [174], are qualitative factors in the evaluation model.

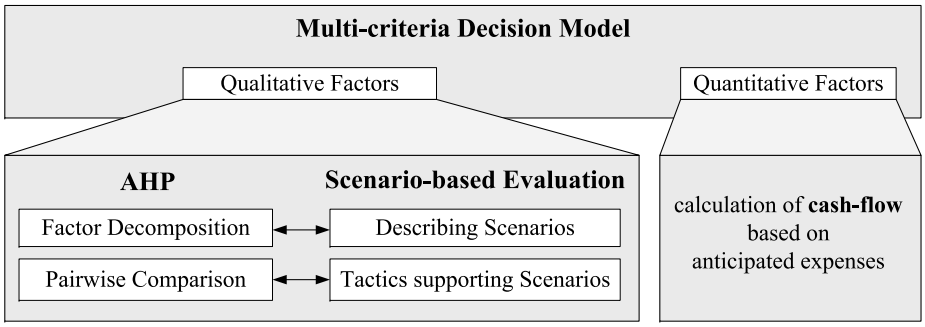


Figure 5.2: Multi-criteria decision model for ICT architectures

To be able to compare the architectural CBP approaches in the face of architectural decisions, it is necessary to get a quantitative measure from qualitative factors. Thus, we apply, extend and customize the multi-criteria decision model of Ghand-foroush et al. [108], which is a modified version of Brown and Gibson's model [53]. As it is a quantitative model, it is useful for selecting one alternative from a given set of alternatives based on quantitative and qualitative factors. Figure 5.2 depicts the design of our multi-criteria decision model developed to evaluate ICT architectures for CBP enactment. The rating of the quantitative factors is determined by the means of cash-flow analysis of the predicted costs. For rating the qualitative factors we combine AHP and scenario-based software architecture evaluation methods. First, based on the AHP, the factors, which have to be considered in the evaluation, are determined by decomposing the evaluation problem and arranged in a hierarchy decomposition tree. The factors are described by the means of quality attributes and scenarios. Rating the scenarios and the alternatives is done by pairwise comparison. The ratings, i.e. the pairwise comparisons, are based upon how good the alternatives realize tactics supporting the respective scenarios.

### 5.2.1 Methodological Issues

Evaluations conducted with the proposed decision method follow a nine step procedure (see Figure 5.3). The first three steps are necessary to set up the decision model for the

specific decision problem, e.g. for CBP architecture evaluation. The selection (Step 1 and Step 2) of the quantitative and qualitative factors is optional, since the predefined factors for CBP architecture evaluation described in this thesis can be used. In Step 3 the prioritization of the subjective factors is determined by pairwise comparisons; this step is specific to the collaboration and organisation(s) for which the evaluation is performed. Steps 4 through Step 7 are used for rating the possible alternatives. In Step 4 one chooses the alternatives that shall be evaluated. The quantitative factors are rated in terms of money and person months in Step 5. The qualitative factor ratings are determined by pairwise comparison of the alternatives (Step 6). Step 7 aggregates the factor ratings quantitative and qualitative factor measures. Before one alternative is chosen in the last step, a sensitivity analysis is performed by varying the importance of the quantitative and the qualitative factor measure (Step 8).

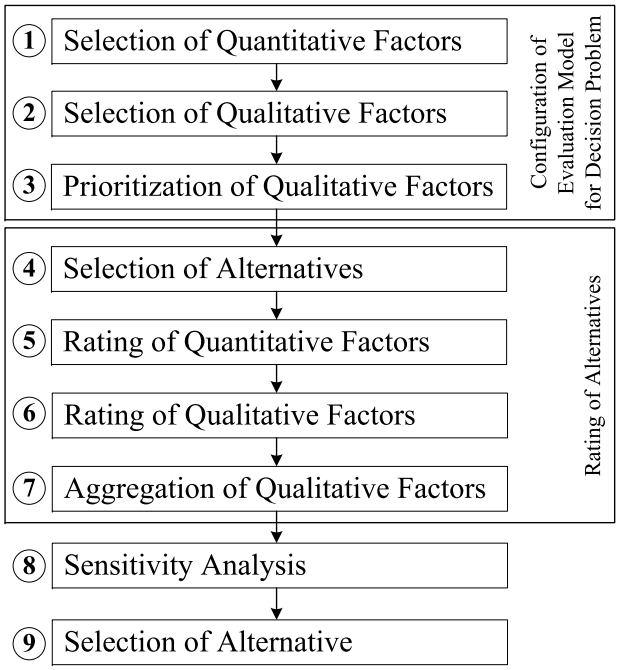


Figure 5.3: Methodology for evaluation and decision model

### 5.2.2 Multi-criteria Evaluation and Decision Model

The multi-criteria evaluation and decision distinguishes between objective (quantitative) factors and subjective (qualitative) factors.

- *Objective factors* are evaluated in monetary terms, and as such are easily quantifiable. Our quantification is based on the cash flow approach and therefore on the discounted present value. The evaluation model considers costs for software, hardware and labor.[1]

[1]The focus of the evaluation model is on the viewpoint of an integrator. The integrator takes into account purchase, licensing, set up, and maintenance costs for hardware and integration and maintenance costs for software. Development of software itself plays a secondary role, since the service or agent software has to be developed independent of the chosen architecture.

- *Subjective factors* are characterized by the fact that they are qualitative measures that typically cannot be quantified. When evaluating software architecture, quality attributes and scenarios are measured in qualitative terms.

The underlying principle of the model is to combine the two evaluation factors into a common evaluation measure. This requires that quantitative considerations and qualitative considerations, where the latter have to be transformed in common measurable units. The model allows to select one software architecture pattern from a given set of alternatives. Following [108], for each software architecture pattern $i$ an architecture evaluation measure $AEM_i$ is defined:

$$AEM_i = X \cdot OFM_i + (1 - X) \cdot SFM_i \tag{5.1}$$

where

$$\begin{aligned}
AEM_i &= \text{architecture evaluation measure, } 0 \leqslant AEM_i \leqslant 1 \\
OFM_i &= \text{objective factor measure, } 0 \leqslant OFM_i \leqslant 1 \text{ and } \sum_{i=1}^{n} OFM_i = 1 \\
SFM_i &= \text{subjective factor measure, } 0 \leqslant SFM_i \leqslant 1 \text{ and } \sum_{i=1}^{n} SFM_i = 1 \\
X &= \text{weight assigned to the objective factor, } 0 \leqslant X \leqslant 1 \\
n &= \text{total number of software architecture patterns evaluated, } 1 \leqslant i \leqslant n
\end{aligned}$$

$AEM_i$ is a measure between 0 and 1 for a particular software architecture pattern, where software architecture patterns with a higher measure score better than patterns with a lower measure. The measure depends to a large extent on the choice of the weight $X$ assigned to the objective factors $OFM_i$ and the subjective factors $SFM_i$. This parameter can be used for sensitivity analysis.

Objective factors are quantified in terms of monetary units. In order to make them comparable to subjective factors, the objective factors have to be converted to a dimensionless index, i.e. an index with the dimension of one:

$$OFM_i = \frac{1}{OFC_i \cdot \sum_{i=1}^{n} \left( \frac{1}{OFC_i} \right)}, i = 1, 2, \ldots, n \tag{5.2}$$

where

$$OFC_i = \text{total objective factor costs for software architecture pattern } i$$

Brown and Gibson [53] ensure through three principles that the objective factor measure is compatible with the subjective factor measure: the software architecture pattern with the highest cost will have the minimum $OFM_i$, the relationship of $OFC_i$ for each pattern relative to all other patterns is preserved, and the sum of all $OFM_i$ is equal to 1.

The subjective factor measure $SFM_i$ is defined as follows:

$$SFM_i = \sum_{j=1}^{m} \left( SFW_j \cdot \sum_{k=1}^{o_j} \left( SSW_{k_j} \cdot SAW_{ik_j} \right) \right) \tag{5.3}$$

$$SFW_j = \frac{SFW'_j}{\sum_{j=1}^{m} SFW'_j} \tag{5.4}$$

$$SSW_{k_j} = \frac{SSW'_{k_j}}{\sum_{k=1}^{o_j} SSW'_{k_j}} \tag{5.5}$$

$$SAW_{ik_j} = \frac{SAW'_{ik_j}}{\sum_{i=1}^{n} SAW'_{ik_j}} \tag{5.6}$$

where

$SFW_j$ = normalized weight value of first level factor $j$
$SFW'_j$ = weight of first level factor $j$ to each first level factor
$SSW_{k_j}$ = normalized weight value of 2nd level factor $k_j$ for one 1st level factor $j$
$SSW'_{k_j}$ = weight of second level factor $k_j$ to all second level factors in first level factor $j$
$SAW_{ik_j}$ = normalized rating of architecture variant $i$ for subjective factor $k_j$
$SAW'_{ik_j}$ = rating of architecture variant $i$ for subjective factor $k_j$
$m$ = total number of first level factors among the subjective factors
$o_j$ = total number of second level factors in a specific first level factor $j$

The subjective factors can be grouped into a hierarchy of factors. A first level factor is an aggregation of a set of second levels factors. Within one first level factor the relative importance of a second level factor is rated by assigning a weight $SSW_{k_j}$ to each of the second level factors. Similar the weight $SFW_j$ specifies the relative importance of one first level factor to the other first level factors. Both factors weights depend on the *organisational context and the collaboration* for which the software architecture patterns are evaluated. The factor weights are independent of software architecture patterns, and can also be used for sensitivity analysis.

All, the first level factor weight $SFW_j$, the second level factor weight $SSW_{k_j}$, and the architecture variant rating $SAW_{ik_j}$ are normalized measures and sum up to one. Thus also the subjective factor measure $SFM_i$ sums up to one and is represented in the same numerical scale as the objective factors.

### 5.2.3 Measuring Qualitative Factors

The part of the evaluation and decision model concerned with measuring qualitative factors is supposed to deal with two main challenges. First it has to provide concepts to evaluate software architecture patterns with respect to organisations' demands. Second the model has to provide means to support people using the model by rating factors and alternatives in order to achieve reasonable and consistent measurements throughout the evaluation process.

We use scenario-based evaluation for software architecture patterns, which is a good way to determine quality attributes of software architecture. The AHP first decomposes a decision problem into a hierarchical network of factors and subfactors before it aggregates second level factors to first level factors. In scenario-based evaluation, first level

factors are represented by quality attributes and second level factors are represented by scenario descriptions.

Since it is problematic to provide sensible scales for measuring the response value of our high level software architectural patterns, we make use of pairwise comparison (see AHP) to rate the qualitative factors and the evaluated software architecture patterns. The decisions for the comparisons are made on the basis of which tactics the evaluated software architecture patterns support and the contingency factors influencing organisations and the collaboration.

### Scenario-based ICT Architecture Evaluation

Scenario-based ICT architecture evaluation (cp. Section 2.3.1) is used to determine quality of software architecture. Hence desired architectural quality attributes are refined by general usage scenarios. These allow a detailed rating of how good quality attributes are supported by software architecture pattern. Quality attributes and scenarios descriptions are used to determine the qualitative factors measure.

**Quality Attributes** Our evaluation model considers the strategic quality attributes modifiability, privacy, reusability and interoperability. For the quality attribute privacy we evaluate the privacy of corporate data and knowledge, which has to be exposed by the enterprises due to the applied software architecture pattern. We do not consider execution related topics like intrusion, denial of service attacks, etc. In the case of interoperability, which can be observed both at execution and build time, we only consider strategic issues like change and reuse of functionality or interaction protocols; we do not consider e.g. conversion of message data at runtime. Furthermore, the evaluation model addresses some more run-time related issues like efficiency and manageability of process execution.

**Scenario Descriptions** The evaluation model is supposed to be suitable for a diversity of systems supporting businesses collaborations. Thus, general scenarios have to be developed, which can be applied to classes of systems rather than to one concrete system. Scenarios represent the characteristics of quality attributes and are used to determine how good quality attributes can be satisfied by systems realizing certain software architecture patterns. The following list gives an overview of the quality attributes (printed in boldface) and the associated scenarios defined for our evaluation and decision model.

- ***Modifiability***
  - *Scenario 1:* Modification of CBPs
  - *Scenario 2:* Change of partners in CBP
  - *Scenario 3:* Incremental development of CBPs
  - *Scenario 4:* Change of elementary services
  - *Scenario 5:* Development of CBP variants
- ***Privacy***
  - *Scenario 6:* Privacy of internal ESs related data
  - *Scenario 7:* Privacy of internal CBPs realizations

- ***Reusability***
  - *Scenario 8:* Reuse of CBPs
  - *Scenario 9:* Reuse of elementary services
- ***Interoperability***
  - *Scenario 10:* Change of CBP protocol specification
  - *Scenario 11:* Change of ES's interfaces
- ***Efficiency***
  - *Scenario 12:* Bottle-neck
  - *Scenario 13:* Security overhead
- ***Manageability***
  - *Scenario 14:* Versioning
  - *Scenario 15:* Monitoring

Table 5.1 depicts the description of the *'Modification of CBPs'* scenario. Descriptions of the other scenarios can be found in Appendix B.1.

| **Scenario 1 – Modification of CBPs** | |
|---|---|
| *Source* | Management |
| *Stimulus* | Due to the constant and rapid change in business existing CBPs have to be adapted to the new business models. |
| *Environment* | Design-time |
| *Artifact* | Cross-organisational business process |
| *Response* | The necessary changes in order to enact the new CBP affect a minimal number of existing modules. Necessary change of existing modules should have no side-effects on other processes (e.g CBPs). |
| *Response Measure* | *Brokerless:* up to n ESs of the partners are affected<br>*Central broker:* the central broker is affected<br>*Decentral broker:* VPs of the respective partner(s) are affected |

Table 5.1: Scenario 1: modification of CBPs

### Factor Decomposition and Pairwise Comparisons

Factor decomposition and pairwise comparisons of our evaluation model are based on the AHP.

**Factor Decomposition** Factor decomposition establishes a hierarchy of first level and second level factors cascading from the decision objective or goal. The hierarchy for our decision method is structured as follows (see Figure 5.4): At the top level one can find the overall goal to have the best *architecture quality*. At the first level contains quality attributes like *modifiability*, *privacy*, *reuse*, etc., which contribute to the quality of an architecture. The scenarios are used at the second level to give a more detailed description of how the quality attributes have to be established. At the bottom level we can find the architectural variants which have to support the scenarios.

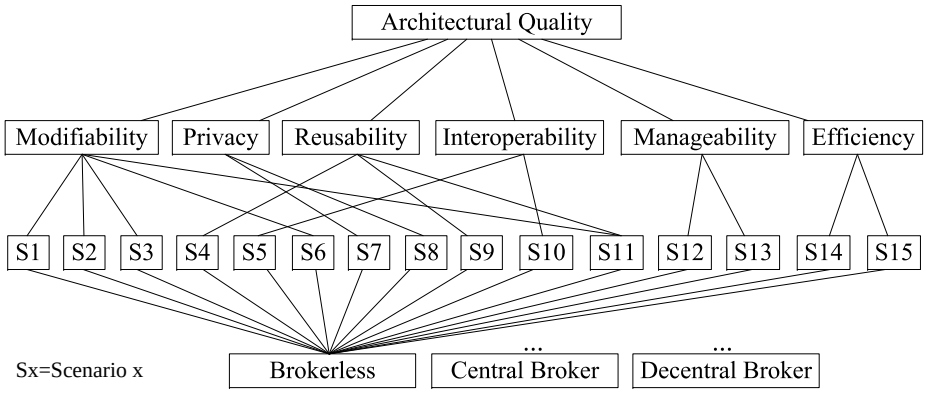


Figure 5.4: AHP decomposition tree for CBP evaluation model

**Pairwise Comparisons** AHP uses pairwise comparison for both determining the priority for the subjective factors and rating the architectural alternatives.

**Weighting the Subjective Factors** To determine the weights for subjective factors, i.e. which scenarios or quality attribute is more important than another, pairwise comparisons are conducted between the first-level factors and the second-level factors. Therefore the factors are arranged in a matrix $a$ and the evaluators have to determine the ratings $a_{ij}$ of the factors by pairwise comparisons. They use a scale to measure relative importance ranging from one to nine (one means that both factors are equally important; nine means that one factor is extremely more important than another). To calculate the ratios of the factors $v_i$, the entries of the matrix $a_{ij}$ have to be normalized to $\overline{a_{ij}}$. Then the normalized matrix entries $\overline{a_{ij}}$ of each row are summed up and divided through the number factors, i.e. the average value of the normalized matrix entries for each row is determined.

$$v_i = \frac{\sum_{j=1}^{n} \overline{a_{ij}}}{n} = \frac{\sum_{j=1}^{n} \frac{a_{ij}}{\sum_{i=1}^{n} a_{ij}}}{n} \tag{5.7}$$

As a result, $v_i$ is the weight for the respective $SFW'_j$ or $SSW'_{k_j}$ for the first and second level factors. It holds that $SFW'_j = SFW_j$ and $SSW'_{k_j} = SSW_{k_j}$ since the weights of the factors $v_i$ are already normalized. The aggregation of the factor weights is achieved by multiplying the second level factor weight with the respective first level factor weight.

**Rating the Scenarios** To rate the scenarios, our decision method applies a relative measurement, which is based on a scale (see above) to express preference of one alternative over another. For example, one can say that to support a scenario under certain contingencies, alternative $a_1$ is strongly favoured instead of alternative $a_2$. For each scenario an evaluation matrix is established, in which the alternatives are compared. To determine the rating of the alternatives (i.e. the priority vector), we apply the 'ideal mode' which should be used in cases where one alternative shall be chosen [258, 259]. The 'ideal mode' solves the rank reversal problem, where the number and kind of alternatives might influence the decision. The matrix is constructed analogous to the matrix for weighting the scenarios. Only the calculation of the priority vector's values differs,

since we apply the 'ideal mode' and not the 'distributive mode'. One obtains the values of the priority vector in ideal mode $v_i^{id}$ by dividing $v_i$ by the maximal value of $v$: $v_i^{id} = \frac{v_i}{max(v_i)}$; $v_i^{id}$ corresponds to the rating of the architecture variant $SAW'_{ik_j}$.

The measurement values of how good ICT coordination architectures support the scenarios is specific to organisational and collaboration context, i.e. the contingencies. It is possible that under certain contingencies one alternative is the best for supporting a scenario, while under different contingencies this alternative may be less appropriate to support the same scenario.

**Consistency and Plausibility of Pairwise Comparisons** The AHP method involves also redundant comparisons to improve validity, recognizing that participants may be uncertain or make poor judgements in some of the comparisons. This redundancy leads to multiple comparisons that may lead to numerical inconsistencies. A consistency ratio ($CR$) is used to estimate accuracy of data. The consistency ratio can be calculated on the basis of the maximum eigenvalue $\lambda_{max}$ of the comparison matrix and a coefficient $RI$[2]: $CR = CI/RI = \frac{\lambda_{max}-n}{n-1}/RI$. Saaty suggested that errors in the measurements are tolerable when $CR \leqslant 10\%$ [259].

For example, if alternative $a_1$ is better than alternative $a_2$ by the factor 3, and alternative $a_2$ is better than alternative $a_3$ also by the factor 3, than alternative $a_1$ has also to be better than alternative $a_3$. For example, the factor 3 for the pairwise comparison between $a_1$ and $a_3$ would not be sufficient, since this would imply that $a_1$ is equal to $a_2$ ($CR = 13\%$).

### Rating the ICT Architecture Alternatives

To compare the ICT coordination architectures one needs to know how good these architectures support architecture quality attributes and scenarios. Therefore it is necessary to understand by which means an architect influences the quality attributes of an architecture. As described in [21], software architects use so-called tactics to achieve quality attributes (see Section 2.3.1).

In the case of *scenario 1* the architect applies tactics that reduce the number of modules and processes (*response of scenario 1*) that are affected by changes to processes (*stimulus of scenario 1*). Through the *maintenance of semantic coherence* the architect ensures that the responsibilities among the services in a CBP work together without excessive reliance on each other. The tactic *anticipate expected changes* reduces the services that need to be modified in case of certain changes. *Generalized services* allow to compute a broader range of functions based on the same input. An architect can apply these three tactics to CBP architectures by using the patterns *abstraction*, *loose coupling*, and *orchestration*.

With this information it is, in general, possible to decide whether one architecture variant supports a scenario better than another one. Having a look at *scenario 1* (cp. Table 5.1), the decentral broker architecture incorporates the patterns *abstraction*, *loose coupling*, and *orchestration* for CBPs, which is the artifact of the scenario description. Thus it realizes the tactics *maintain semantic coherence*, *anticipate expected changes*, and *generalize module*. The brokerless architecture instead, realizes none of these patterns and tactics for the artifact CBPs of scenario 1. Thus we can infer that the decentral

[2]The author of [259] suggests to use the following values for $RI$: $n = 3$, $RI = 0.52$; for $n = 4$, $RI = 0.89$; for $n = 5$, $RI = 1.11$; for $n = 6$, $RI = 1.25$.

broker architecture better supports scenario 1 than the brokerless architecture. The remaining question is, how contingencies influence the ratings and the distance between the ratings of the evaluated architectures?

### Influences on the Ratings

The ratings of the architecture alternatives are influenced by various factors. In the following, we discuss for CBPs how tactics and patterns, that are used to achieve architectural quality, as well as contingencies, i.e. internal and external factors of the application scenario, influence the ratings.

**Tactics and Patterns** To compare the software architectures in the *Rating of Qualitative Factors* step of the evaluation methodology (see Section 5.2.1), it is necessary to understand by which means an architect influences the quality attributes of an architecture. As described in Section 2.3.1, software architects use so-called tactics to achieve quality attributes.

Table 5.2 provides an overview about which tactics and patterns are most sensibly applied in an ICT coordination architecture for CBPs. In the case of scenario 1 the architect applies tactics that reduce the number of modules and processes (response of scenario 1) that are affected by changes to processes (stimulus of scenario 1). Through the *maintenance of semantic coherence* the architect ensures that the responsibilities among the services in a CBP work together without excessive reliance on each other. The tactic *anticipate expected changes* reduces the services that need to be modified in case of certain changes. *Generalized services* allow to compute a broader range of functions based on the same input. An architect can apply these three tactics to CBP architectures by using the patterns *abstraction*, *loose coupling*, and *orchestration*.

With this information it is, in general, possible to decide whether one architecture variant supports a scenario better than another or not. Having a look at scenario 1 (cp. Table 5.1), the decentral broker architecture incorporates the patterns *abstraction*, *loose coupling*, and *orchestration* for CBPs, which is the artifact of the scenario description. Thus it realizes the tactics *maintain semantic coherence*, *anticipate expected changes*, and *generalize module*. The brokerless architecture instead, does realize none of the these patterns and tactics for the artifact (CBPs) of scenario 1. Thus, we can infer that the decentral broker architecture better supports scenario 1 than the brokerless architecture.

**Contingencies** The ratings of architectures depend to a high degree on the organisational, the collaboration, and the external context for which the architectures are evaluated. Like in contingency theory [72] (see 2.3.3), where the performance and efficiency of an organisational structure depends on internal and external contingencies, the rating of the evaluated architectures also depends on such contingencies.

In our decision method we consider contingencies within the collaboration network (internal contingencies) and outside the collaboration network (external contingencies). Internal contingencies characterize the collaboration model and the organisations participating in the collaborations. These are: the *collaboration topology*, that takes into account the distribution of influence and power among the partners; the complexity and specificity of the *products* developed by the collaborating organisations; the *service flow* that is characterized by the amount of data and the number of messages exchanged; aspects related with the *process* itself like length of the process, defined through the num-

| | Patterns | | | | | Tactics | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Wrapper | Broker | Abstraction | Loose coupling | Orchestration | Maintain semantic coherence | Anticipate expected changes | Generalize module | Restrict communication paths | Use an intermediary | Maintain existing interfaces | Hide information |
| Scenario 1 | - | - | x | x | x | x | x | x | - | - | - | - |
| Scenario 2 | - | x | - | x | x | - | x | - | - | - | x | - |
| Scenario 3 | - | x | x | x | x | x | x | - | x | - | - | - |
| Scenario 4 | - | x | - | - | x | x | - | x | - | x | - | x |
| Scenario 5 | - | x | x | - | x | - | x | - | - | x | x | x |
| Scenario 6 | x | x | - | - | - | - | x | - | - | x | x | x |
| Scenario 7 | x | x | x | - | - | - | - | - | - | x | x | x |
| Scenario 8 | - | x | x | - | x | - | - | x | - | - | x | x |
| Scenario 9 | - | x | x | - | x | - | - | x | - | x | x | - |
| Scenario 10 | - | x | x | - | x | - | x | - | x | x | - | - |
| Scenario 11 | - | - | x | x | x | x | x | - | - | x | - | - |

Table 5.2: Patterns and tactics that can be used to support scenario 1 to 11

ber of processing steps, or the estimated number of process instances during execution. External contingencies are external factors that highly influence organisations' decision and strategies, and therefore impact also the choice of an ICT coordination architecture: *standardization* considers the existence of industry-specific, national, or international standards; *maturity* takes the existence of commonly accepted processes, protocols, etc., into account; *business semantics* considers the availability of standards and their maturity with regard to defining semantics of a specific domain; *legislation* comprises the regulations which impose special requirements regarding security, monitoring, and other aspects of the collaboration. [165]

Table 5.3 illustrates how contingencies influence the scenario ratings (for a complete overview see Appendix B.2). Depending on the contingencies the importance of realizing tactics to best support scenarios varies. In the case this is directly proportional (⇈), we can say that the stronger the influence of contingencies (e.g. higher product complexity), the higher is the difference between two architecture ratings, where one architecture supports and the other architecture does not support the scenario by tactics. Inversely proportional (↑↓) represents the fact that a higher influence of contingencies (e.g. high degree of standardization) results in a lower difference between the architectures in the scenario ratings.

| | | Scenario 1 | Scenario 2 | Scenario 3 | Scenario 4 | Scenario 5 |
|---|---|---|---|---|---|---|
| Internal Contingencies | Coll. topology | initiator ⇈ | power of players ⇈ | n/a | n/a | n/a |
| | Product | complexity ⇈ | complexity ⇈ | n/a | complexity/ specifity ↑↓ | n/a |
| | Service flow | n/a | n/a | n/a | n/a | ⇈ |
| | Process | n/a | length ⇈ | # process instances ⇈ | length ⇈ | length ⇈ |
| External Contingencies | Standardization | ↑↓ | ↑↓ | ↑↓ | ↑↓ | ↑↓ |
| | Maturity | ↑↓ | ↑↓ | ↑↓ | ↑↓ | ↑↓ |
| | Bus. semantics | n/a | ↑↓ | ↑↓ | ↑↓ | ↑↓ |
| | Legislation | n/a | n/a | n/a | n/a | n/a |

Table 5.3: Influence of contingencies on scenario ratings

If we assume for example a high degree of standardization to rate scenario 1, the decentral broker architecture is not much better than or even equal to the brokerless architecture. Standardized parts of the CBP and the ESs can be reused and combined in arbitrary ways adapting to the change in business (stimulus of scenario 1). Necessary changes affect about the same number of modules (cp. response and response measure of scenario 1 in Table 5.1) in both coordination architectures.

Of course there exist other contingencies, which are also relevant for the decision about an ICT coordination architecture. For example, the dynamics of the collaboration (internal contingency) and the industry dynamics (external contingency) both address the aspect of change. Since change is already covered by the scenario descriptions, this aspect has to be considered by weighting scenarios and quality attributes. Change is not addressed a second time in rating the scenarios.

### 5.2.4 Measuring Quantitative Factors

In the decision method quantitative factors are evaluated in monetary terms on the basis of the discounted cash flow approach. The discounted present value of the future cash flows $FV_i^D$, which corresponds to the objective factor measure $OFC_i$ for a software architecture pattern $i$, is defined as follows:

$$OFC_i = \sum_{j=1}^{m} FV_{i_j}^D = \sum_{t=0}^{N-1} \frac{FV_{i_{j_t}}}{(1+d)^t} \tag{5.8}$$

where

$FV_{i_j}^D$ = discounted present value of the future cash flow ($FV$) for factor $j$
$FV_{i_{j_t}}$ = nominal value of a cash flow amount in a future period $t$ for factor $j$
$d$ = discount rate
$N$ = number of discounting periods
$m$ = total number of objective factors

For simplicity reasons we assume that all expenses necessary to set up an ICT system occur at present time ($FV_{i_0}; t = 0$). Running costs like for maintenance of the system or changes to the systems are considered annually ($FV_{i_t}; t > 0$). The decision model considers costs for software (*purchasing costs* and *annual licences*), hardware (*purchasing costs* and *annual leasing fees*) and labor (*costs to set up the systems*, *maintenance costs*, and *costs to develop and deploy new and modified processes*).

#### Software

For software costs $FV_{i_1}^D$ the model considers *purchasing costs* $FV_{i_{11}}^D$, which are accounted at $t = 0$, and *annual licence fees* $FV_{i_{12}}^D$.

$$FV_{i_1}^D = FV_{i_{11}}^D + FV_{i_{12}}^D = FV_{i_{11_0}} + \sum_{t=0}^{N-1} \frac{FV_{i_{12_t}}}{(1+d)^t} \tag{5.9}$$

**Purchasing** To calculate *purchasing costs* $FV_{i_{11_0}}$ the average software costs of one broker component $C_{purchaseSWBr}$, the average software costs of one elementary service $C_{purchaseSWES}$, the total number of broker processes $n_{purchaseSWBr_i}$, and total number of elementary services $n_{purchaseSWES_i}$ have to be determined.

$$FV_{i_{11_o}} = n_{purchaseSWBr_i} \cdot C_{purchaseSWBr} + n_{purchaseSWES_i} \cdot C_{purchaseSWES} \tag{5.10}$$

**Licencing** To calculate *annual licence costs* $FV_{i_{12_t}}$ with $0 \leqslant t < N$, the average annual licence costs of one broker component $C_{annualSWBr}$, the average annual licence costs of one elementary service $C_{annualSWES}$, the total number of broker processes $n_{annualSWBr_i}$, and the total number of elementary services $n_{annualSWES_i}$ have to be determined.

$$FV_{i_{12_t}} = n_{annualSWBr_i} \cdot C_{annualSWBr} + n_{annualSWES_i} \cdot C_{annualSWES} \tag{5.11}$$

### Hardware

For hardware costs $FV_{i_2}^D$ the model considers *purchasing costs* $FV_{i_{21}}^D$, which are accounted at $t = 0$, and *leasing fees* $FV_{i_{22}}^D$, which are accounted annually for $0 \leqslant t < N$.

$$FV_{i_2}^D = FV_{i_{21}}^D + FV_{i_{22}}^D = FV_{i_{21_0}} + \sum_{t=0}^{N-1} \frac{FV_{i_{22_t}}}{(1+d)^t} \tag{5.12}$$

**Purchasing** To calculate *purchasing costs* $FV_{i_{11_0}}$ the average hardware costs for computer machines hosting broker components $C_{purchaseHWBr}$, the average hardware costs for computer machines hosting elementary services $C_{purchaseHWES}$, the total number of computer machines hosting broker processes $n_{purchaseHWBr_i}$, and the total number of computer machines hosting elementary services $n_{purchaseHWES_i}$ have to be determined.

$$FV_{i_{21_0}} = n_{purchaseHWBr_i} \cdot C_{purchaseHWBr} + n_{purchaseHWES_i} \cdot C_{purchaseHWES} \tag{5.13}$$

**Leasing** To calculate *annual leasing fees* $FV_{i_{12_t}}$ with $0 \leqslant t < N$ the average annual leasing costs for computer machines hosting broker components $C_{leasingHWBr}$, the average annual costs for computer machines hosting elementary services $C_{leasingHWES}$, the total number of computer machines hosting broker processes $n_{leasingHWBr_i}$, and the total number of computer machines hosting elementary services $n_{leasingHWES_i}$ have to be determined.

$$FV_{i_{22_t}} = n_{leasingHWBR_i} \cdot C_{leasingHWBR} + n_{leasingHWES_i} \cdot C_{leasingHWES} \tag{5.14}$$

### Labor

For the factor labor costs $FV_{i_3}^D$ the model distinguishes between *costs to set up systems* $FV_{i_{31}}^D$ , system *maintenance costs* $FV_{i_{32}}^D$, and *costs to develop and deploy new and modified processes* $FV_{i_{33}}^D$.

$$FV_{i_3}^D = FV_{i_{31}}^D + FV_{i_{32}}^D + FV_{i_{33}}^D = FV_{i_{31_0}} + \sum_{t=0}^{N-1} \frac{FV_{i_{32_t}}}{(1+d)^t} + \sum_{t=0}^{N-1} \frac{FV_{i_{33_t}}}{(1+d)^t} \tag{5.15}$$

**Setup** Costs to *set up the system* $FV_{i_{31_t}}$ are accounted at $t = 0$ and aggregate costs for development, integration, customization, and deployment of the system but also costs for consulting, training, etc. Therefore the average costs to set up the system $C_{setup_i}$ have to be accounted.

$$FV_{i_{31_0}} = C_{setup_i} \tag{5.16}$$

**Maintenance** *Maintenance costs* $FV_{i_{32_t}}$ are accounted annually $0 \leqslant t < N$. This factor also comprises costs for various tasks. Therefore the average costs for maintaining the system $C_{maintenance_i}$ have to be accounted.

$$FV_{i_{32_t}} = C_{maintenance_i} \tag{5.17}$$

**Change** *Costs to develop and deploy new and modified processes* are accounted annually $0 \leqslant t < N$. The average number of broker processes that have to be changed $n_{changeBr_i}$, the average number of elementary services that have to be changed $n_{changeES_i}$, the average costs for changing a broker process $C_{changeBr_i}$, the average costs for changing an elementary service $C_{changeES_i}$, and the average number of changes per year $n_{change}$ have to be determined.

$$FV_{i_{33_t}} = n_{change} \cdot \left( n_{changeBr_i} \cdot C_{changeBr_i} + n_{changeES_i} \cdot C_{changeES_i} \right) \tag{5.18}$$

## 5.3 Applying the Evaluation Method

As described in the introduction of this chapter, companies organise themselves into global networks and outsource those activities that can be performed more quickly and effectively or at lower cost, by others [270]. However, outsourcing and interacting in global networks also increases overhead costs for collaboration, coordination, and intermediation. One approach to describe the influence of organisational structure on these overhead costs is the transaction cost model [322, 323]. In today's economies, transactions for example make up more than 30% of the total costs of an automobile [280]. Transaction costs heavily depend on the capabilities of business systems to keep up with constantly evolving business relationships and cross-organisational value chains. However, in the comparison to transaction costs, IT costs are much less than transaction costs (in the Automotive example this is about 6% of the overall costs [280]).

In this section, we apply the evaluation method to two scenarios: a virtual enterprise scenario (Section 5.3.1) and to a scenario with collaborating Small and Medium Enterprises (SMEs) (Section 5.3.2). In doing so, the goal is to identify the collaboration architecture which best supports the cross-organisational value chain and helps to reduce transaction costs. The trade-off between reducing transaction costs (qualitative factors) and reducing of IT costs (quantitative factors) through the choice of a collaboration architecture is discussed in a sensitivity analysis.

### 5.3.1 Virtual Enterprise Scenario

This scenario deals with virtual enterprises that collaborate in big, long-running CBPs (approx. 90 processing steps). The OEM and the big first-tier suppliers introduced in the automotive scenario in Section 5.1 together form a virtual enterprise, which builds a temporary network of independent companies, suppliers, customers. They are linked by information technology to share costs, skills, and access to one another's markets. The services the partners provide to the CBP are to 50% legacy applications, which will be replaced within the next five years. The services, their interfaces, and data types are not standardized, so that interoperability is an important issue. About 30% of the CBP are standardized and it may be necessary to provide variants of the CBP. The privacy of the enterprises' services is only medium important, since the enterprises make their profit through economy of scale. Hence, they also participate with their elementary services in other CBPs.

In the next paragraphs we will present the results of applying our evaluation method to the virtual enterprise scenario. The complete data relevant for the evaluation can be found in Appendix B.3.

### Determining the Qualitative Measure

**Weighting the Subjective Factors** To determine the weight of the quality attributes and the scenarios pairwise comparison are applied like described in Section 5.2.3.

| | mod. | pri. | reuse | int. | eff. | man. | $v_i$ |
|---|---|---|---|---|---|---|---|
| modifiability | 1 | 7 | 3 | $\frac{1}{3}$ | 3 | 3 | 0.21 |
| privacy | $\frac{1}{7}$ | 1 | $\frac{1}{4}$ | $\frac{1}{9}$ | $\frac{1}{5}$ | $\frac{1}{5}$ | 0.03 |
| reuse | $\frac{1}{3}$ | 4 | 1 | $\frac{1}{5}$ | 1 | 1 | 0.10 |
| interoperability | 3 | 9 | 5 | 1 | 5 | 5 | 0.45 |
| efficiency | $\frac{1}{3}$ | 5 | 1 | $\frac{1}{5}$ | 1 | 1 | 0.10 |
| managability | $\frac{1}{3}$ | 5 | 1 | $\frac{1}{5}$ | 1 | 1 | 0.10 |

Table 5.4: Priority comparison matrix for the first level factors

Table 5.4 depicts the weighting of the first level factors, i.e. the quality attributes, for the virtual enterprise scenario. Modifiability is considered more important than privacy and reuse but less important than interoperability. The column of the priority vector $v_i$ depicts the weighting of the quality attributes.

| modifiability | sc.1 | sc.2 | sc.3 | sc.6 | sc.11 | $v_i$ |
|---|---|---|---|---|---|---|
| scenario 1 | 1 | 3 | 7 | $\frac{1}{5}$ | $\frac{1}{3}$ | 0.14 |
| scenario 2 | $\frac{1}{3}$ | 1 | 5 | $\frac{1}{5}$ | $\frac{1}{5}$ | 0.08 |
| scenario 3 | $\frac{1}{5}$ | $\frac{1}{7}$ | 1 | $\frac{1}{9}$ | $\frac{1}{9}$ | 0.03 |
| scenario 6 | 5 | 5 | 9 | 1 | 3 | 0.47 |
| scenario 11 | 3 | 5 | 9 | $\frac{1}{3}$ | 1 | 0.27 |

Table 5.5: Priority comparison matrix for the second level factor modifiability

Table 5.5 depicts the weighting of the scenarios are used to describe the modifiability attribute in the virtual enterprise scenario. The scenarios are analogously compared as the other quality attributes in Table 5.4. The column of the priority vector $v_i$ depicts the weighting of the scenarios.

**Rating the Scenarios** The scenarios are rated by pairwise comparing the architecture alternatives. The decisions are based on how good the architectures support the scenarios via tactics and patterns). The rating, i.e. the values decision, also depend on the characteristics of the contingency factors of the application scenario for which the evaluation is performed.

| scenario 1 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | $\frac{1}{7}$ | $\frac{1}{7}$ | 0.14 |
| Cen-Br. | 7 | 1 | 1 | 1.00 |
| Dec-Br. | 7 | 1 | 1 | 1.00 |

Table 5.6: Rating scenario 1

Table 5.6 depicts the rating matrix for scenario 1. As described in Section 5.2.3 the central broker alternative supports scenario 1 better than the brokerless alternative. Relevant contingencies for scenario 1 are the grade of standardization and the maturity of the CBP and the services. Since both contingencies are rather low in the virtual enterprise scenario, the architectural quality is important for the support of this scenario,

which leads to the comparison value 7 between the central broker and brokerless architecture. The central and decentral broker architecture a rated equally important, i.e. 1. The column of the priority vector $v_i^{id}$ depicts the weighting of the scenarios.

**Overall Subjective Measure** The overall subjective measure is computed on the basis of the factor weights and the scenario ratings. Table 5.7 depicts the relevant data. In row two one can find the weighting of the quality attributes from Table 5.4. The weighting of the scenarios that describe the quality attributes are specified in row four. The scenario ratings can be found in the columns of the respective scenarios. For example the rating, i.e. priority vector values $v_i^{id}$, for scenario 1 can be found in column 2 row 5-7. The overall subjective measure is calculated with the formula (5.3) and can be found in the last column.

| | Modifiability | | | | | Sec. | | Reuse | | | Int.op. | | Eff. | | Man. | | $SFM_i^{id}$ | $SFM_i$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0.21 | | | | | 0.03 | | 0.10 | | | 0.45 | | 0.10 | | 0.10 | | | |
| | S1 | S2 | S3 | S6 | S11 | S7 | S8 | S4 | S9 | S11 | S5 | S10 | S12 | S13 | S14 | S15 | | |
| | 0.14 | 0.08 | 0.03 | 0.47 | 0.27 | 0.17 | 0.83 | 0.43 | 0.43 | 0.14 | 0.17 | 0.83 | 0.25 | 0.75 | 0.67 | 0.33 | | |
| Wo-Br. | 0.14 | 0.11 | 0.08 | 0.16 | 0.08 | 0.24 | 0.20 | 0.25 | 0.17 | 0.08 | 0.14 | 0.17 | 1.00 | 0.17 | 0.12 | 0.11 | 0.178 | 0.108 |
| Cen-Br. | 1.00 | 0.44 | 0.30 | 0.46 | 0.30 | 0.14 | 0.20 | 0.25 | 0.59 | 0.30 | 1.00 | 0.41 | 0.33 | 1.00 | 1.00 | 1.00 | 0.568 | 0.343 |
| Dec-Br. | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.41 | 0.60 | 0.44 | 0.908 | 0.549 |

Table 5.7: Overall subjective measure

### Determining the Quantitative Measure

**Overall Objective Measure** The objective measure is calculated on the basis of the cash flow of the costs for software, labor, hardware. For the virtual enterprise scenario with four collaborating enterprises we have estimated the following costs. It is important to understand, that the scale (euro, dollar, etc.) is not important for the overall objective measure, since the scale is transformed into an dimensionless index. In Table 5.8 one can see that for the brokerless architecture 5075 thousand cost units were estimated ($OFC_i$). The overall objective measure $OFM_i$ can be found in the last column.

| | Software | Hardware | Labour | $OFC_i$ | $OFM_i$ |
|---|---|---|---|---|---|
| Wo-Br. | 45K | 75K | 4955K | 5075K | 0.127 |
| Cen-Br. | 69K | 95K | 1200K | 1364K | 0.474 |
| Dec-Br. | 118K | 135K | 1367K | 1620K | 0.399 |

Table 5.8: Overall objective measure

### Sensitivity Analysis and Interpretation

The architectural evaluation measure $AEM_i$ for each architecture variant is determined on the basis on the objective factor measure $OFM_i$ and the subjective factor measure $SFM_i$ (see formula (5.1)). The measure depends on the weight $X$ assigned to the objective and subjective factor. This weight lends itself also for sensitivity analysis.

Figure 5.5 depicts the sensitivity analysis chart for the virtual enterprise scenario. The x-axis represents the importance of the objective factors measure and the y-axis the architecture evaluation measure for the respective architecture variant.

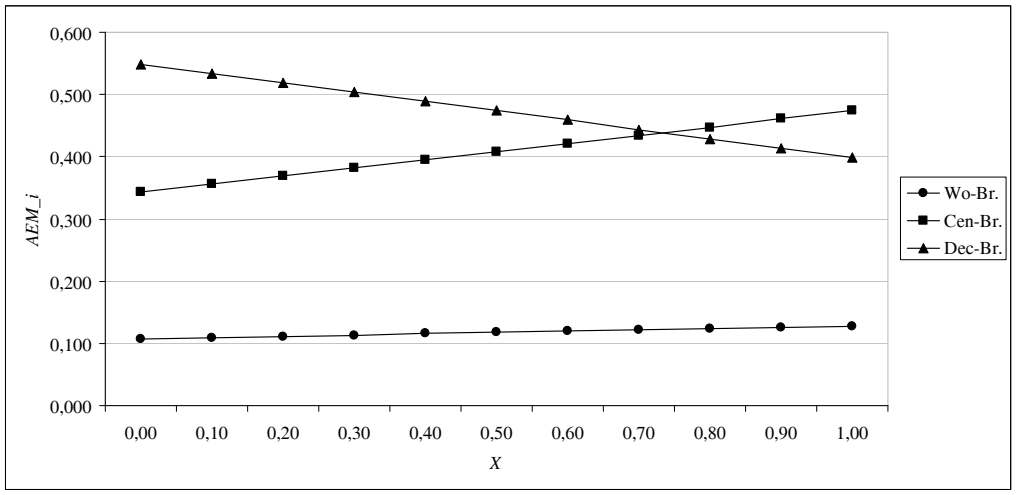


Figure 5.5: Sensitivity analysis chart

On the basis of this evaluation result we can conclude, that either the central broker or the decentral broker architecture variant should be selected. The brokerless variant gets significantly lower rating values for all $X$ than the other ones. The decentral broker architecture scores better for the qualitative measurement (especially for $X = 0$), while the central broker architecture is better in terms of IT costs. A feasible estimation of $X$ is to consider the relationship between the percentage of transaction costs and IT costs of the total costs. In the automotive industry IT costs (6%) are low in comparison to the transaction costs (30%) (cp. [280]). This leads to an estimation of $X \approx 0.2$ for the virtual enterprise scenario applied to the automotive industry. Thus, we would suggest to select the decentral broker architecture in the virtual enterprise scenario. Even if transaction costs and IT costs got equally important ($X = 0.5$), the architecture evaluation measure of the decentral broker variant would be still be a bit better than the central broker variant.

### 5.3.2 SME Scenario

This scenario represents the CBPs between the second-tier (or even third- and fourth-tier suppliers) of the automotive scenario from Section 5.1. The second-tier suppliers are SMEs that manufacture parts, which can be largely standardized and can be reused in many cars or other application domains. The SMEs produce for example screws, fuses, circuit boards, etc. They support rather short processes with approx. 20 processing steps. The specificity of the service is low. Smaller and equal partners (SMEs) frequently join and leave the collaborations and most SMEs also participate in other similar collaborations. Participating partners have similar interfaces, data types, etc., and the services and CBPs are de-facto standardized (e.g. already formulated in ebXML BPSS [203]). Hence, interoperability is not so an important issue to these organisations. Also changes to the existing CBPs are rare (up to three times a year). However, about 50% of the service offer by the SMEs are legacy applications, which will be partially replaced within the next five years.

In the next paragraphs we will present the results of applying our evaluation method to the SME scenario. The complete data relevant for the evaluation can be found in Appendix B.4.

### Determining the Qualitative Measure

The overall subjective measure can be found in Table 5.9.

| | Modifiability 0.16 | | | | | Sec. 0.04 | | Reuse 0.25 | | | Int.op. 0.06 | | Eff. 0.25 | | Man. 0.25 | | $SFM_i^{id}$ | $SFM_i$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | S1 | S2 | S3 | S6 | S11 | S7 | S8 | S4 | S9 | S11 | S5 | S10 | S12 | S13 | S14 | S15 | | |
| | 0.05 | 0.59 | 0.05 | 0.21 | 0.11 | 0.50 | 0.50 | 0.45 | 0.45 | 0.09 | 0.20 | 0.80 | 0.83 | 0.17 | 0.17 | 0.83 | | |
| Wo-Br. | 0.50 | 0.36 | 0.50 | 0.20 | 0.40 | 1.00 | 1.00 | 1.00 | 0.17 | 0.40 | 0.17 | 0.40 | 1.00 | 0.50 | 0.30 | 0.17 | 0.530 | 0.293 |
| Cen-Br. | 1.00 | 0.50 | 1.00 | 1.00 | 0.64 | 1.00 | 1.00 | 0.30 | 0.41 | 0.64 | 0.41 | 0.64 | 0.12 | 1.00 | 1.00 | 1.00 | 0.589 | 0.326 |
| Dec-Br. | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.55 | 1.00 | 1.00 | 1.00 | 1.00 | 0.40 | 1.00 | 0.55 | 0.41 | 0.688 | 0.381 |

Table 5.9: Overall subjective measure

### Determining the Quantitative Measure

The overall objective measure can be found in Table 5.10.

| | Software | Hardware | Labour | $OFC_i$ | $OFM_i$ |
|---|---|---|---|---|---|
| Wo-Br. | 100K | 100K | 100K | 300K | 0.453 |
| Cen-Br. | 124K | 120K | 195K | 439K | 0.310 |
| Dec-Br. | 197K | 180K | 198K | 575K | 0.237 |

Table 5.10: Overall objective measure

### Sensitivity Analysis and Interpretation

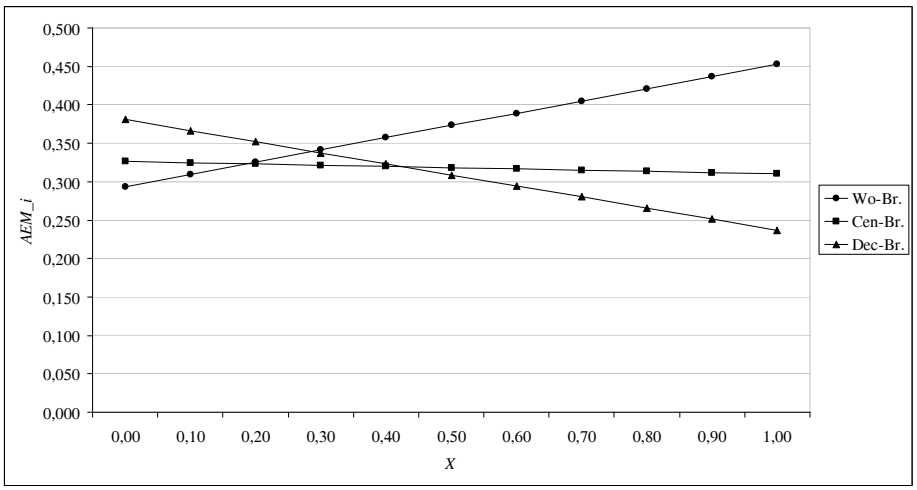


Figure 5.6: Sensitivity analysis chart

Figure 5.6 depicts the sensitivity analysis chart for the SME. One can clearly see how the contingencies standardization and short processes influence the architecture evaluation measure. Although the partners in the collaboration frequently change the brokerless architecture variant scores very well. For most $X$, the brokerless architecture has the highest evaluation measure and even four low $X$ its measure is hardly lower than the

measure for the broker architecture. However, if contingencies change, like new monitoring requirements from the government, the intersection point of the curves would be at a higher $X$ (it would move to the right). This would make the broker architectures more interesting to realize the SMEs scenario.

## 5.4 Discussion and Conclusions

The contribution of the work presented in this chapter is fourfold: First, we developed a model for decision support suitable for IT architects to derive an appropriate architecture paradigm for a given use case or application domain. The decision model combines the Analytic Hierarchy Process (AHP) with scenario-based architecture evaluation techniques. Second, we specified scenario descriptions that allow the evaluation and selection of appropriate ICT system coordination architecture paradigms for CBP enactment. Third, we developed a set of guidelines of how contingencies influence ICT system coordination architecture based on our experiences in model-driven CBP modelling and enactment. Finally, we validated our observations by applying our decision support method, the scenario descriptions for CBP enactment, and the guidelines about contingency influence to application scenarios with differing characteristics.

The evaluation and decision model can be used to validate guidelines like *"Standardization for brokerless approach is very important ..."*, *"The hierarchical structure is typically used in traditional business relationships where interactions between cooperating partners are agreed to in advance."*, or *"Process management and monitoring of the overall process status is generally easier in hierarchical structures ..."* described in [168, p.204f]. We further used the model to develop new guidelines for ICT architecture selection based on the influence on contingencies (cp. Section 5.2.3).

The evaluation enables people to get a better understanding of the influence of contingencies on the overall decisions and to apply these guidelines. Our experience so far indicates that pairwise comparisons reduce the amount of information that is necessary for decisions. Since people can only deal with information involving simultaneously a small number of facts (seven plus or minus two) [188], pairwise comparisons help evaluators to make better judgements compared to methods where more information needs to be considered. Though pairwise comparisons require more complex calculations than other rating approaches, they promise to provide more exact results. The AHP method involves also redundant comparisons to improve validity, recognizing that participants may be uncertain or make poor judgements in some of the comparisons.

Future work is the question how decision methods as the one described in this section can be built into existing enterprise modelling frameworks and model-driven IDEs, to support process modelers and ICT architects in their task of creating and managing executable CBP specifications from business level models. A second area concerns the specification of further scenarios to extend the scope of the evaluation and decision method's application. More fine-grained models and extensions of our decision method need to be developed to support the process down to the platform-specific and code levels.

# Chapter 6

# Ontology-based Model Transformation

MDSD is getting more sophisticated by using more powerful tools and languages for modelling enterprises and developing ICT systems. As a natural course of things a huge diversity of often specialized methodologies, modelling languages, and representation formats has evolved, serving the purposes of the particular application domains. Especially in the context of automated system and code generation there is a strong trend towards the application and usage of DSLs. An exact representation of application domain concepts through DSLs (for more details see Chapter 3.2.3) is a key enabler of efficient and automated generation solutions.

Applying DSLs and specialized metamodels (different metamodels for the same language) in MDSD scenarios also brings certain restrictions and limitations. With the use of different metamodels, models cannot be simply exchanged. One has to specify model transformations that transform models of one metamodel into models of another metamodel. Moreover, with the evolution of metamodels, i.e. new versions like in UML are released, it is necessary to adjust existing generators and model transformations that were built for the old metamodel version.

Syntactic and semantic differences in representation formats, caused through the evolution and use of different metamodels, hinder the efficient exchange and reuse of models and model transformations. This causes interoperability problems, where interoperability can broadly be characterized as the ability of enterprises or systems to cooperate seamlessly with each other. Interoperability is not only an issue of ICT systems collaborating at runtime. It is also a matter of communicating both with internal and external organisation units in order to develop new models for collaboration and supporting ICT systems. Information and knowledge about enterprises, their organisational structure, processes, collaboration with external organisations but also ICT systems is commonly captured in models. To enable seamless collaboration in enterprise and systems modelling, enterprises have to be supported by interoperability solutions for model sharing, model transformation reuse, and model transformation evolution, independent of modelling languages and tools.

Summing up, to cope with the evolution of metamodels and the use of different metamodel versions it has to be possible to generate new model transformations and adjust and reuse existing ones. However, developing model transformations is a kind of metaprogramming and model transformations have to be specified between metamodels

[65]. This is an error-prone and long lasting task, that implies a deep knowledge of all the underlying modelling technology, which in most cases is quite hard to learn.

In this chapter we develop the approach of Ontology-based Model Transformation (OntMT), that provides a solution for (semi-)automated development, reuse, and adjustment of model transformations. OntMT fosters the exchange of models and the reuse and evolution model of transformations. In detail, this chapter provides the following contributions:

- We develop the Ontology-based Model Transformation (OntMT) approach, which provides means to automatically deal with model and model transformation evolution scenarios. We implement OntMT and apply OntMT to two real world case studies.
- We introduce and describe an architecture for a semantic-enabled modelling and development suite. Semantic-enabled tools support developers and modellers in a sophisticated manner by making use of reasoning results. We implement OntMT as such a semantic-enabled tool.
- We develop concepts and techniques to realize and implement the OntMT approach. These are bootstrapping rules to generate QVT Relations model transformations, a higher-order model transformation language for QVT Relations model transformations as well as representation and reasoning techniques to infer relationships between metamodels. We develop a correlation algorithm and a rating that allow to generate and choose applicable model transformations.
- We develop a higher-order model transformation language that allows to modify model transformations and lends itself for automating reuse of model transformations.

This chapter is structured as follows: In Section 6.1 we illustrate problems that occur in collaborative MDSD and provide a concise problem statement. Next, we present the foundations of the OntMT approach and illustrate the mechanisms of OntMT for automated generation and reuse of model transformations (Section 6.2). Section 6.3 introduces the architecture of a semantic-enabled modelling and development suite, and presents how OntMT is realized as a part of this suite. In Section 6.4 we develop concepts and techniques to realize and implement the OntMT approach. Before we discuss the potential impact of applying OntMT in Section 6.6, we present two cases studies in which we applied OntMT to real world scenarios (Section 6.5). Section 6.7 presents related work and provides a comparison of approaches dealing with (meta)model evolution. Section 6.8 provides a summary and conclusions.

## 6.1 Problem Description

To enable collaboration in enterprise and systems modelling, enterprises have to be supported by interoperability solutions for model sharing and model exchange independent of modelling languages and tools. Also the evolution of model transformations has to be considered. To maintain and reuse existing model transformations, these transformations have to be adjusted to new modelling languages or styles. This section illustrates these challenges via a MDSD scenario. It further discusses the problems that possible automation solutions face.

### 6.1.1 A MDSD Scenario

In the field of CBP development and enactment, one can find a huge diversity of DSLs and metamodels like PIM4SOA, BPDM, UML activities or the AgilPro metamodel. In Figure 1.1 of Chapter 1 we illustrated the application of MDSD to CBP development. The vertical dimension distinguishes the different layers of abstraction applied in MDSD, and the horizontal dimension represents the collaborative modelling between two enterprises *A* and *B*. Models of enterprises *A* and *B* have to be shared at different levels of abstraction in order to agree on and develop CBPs.

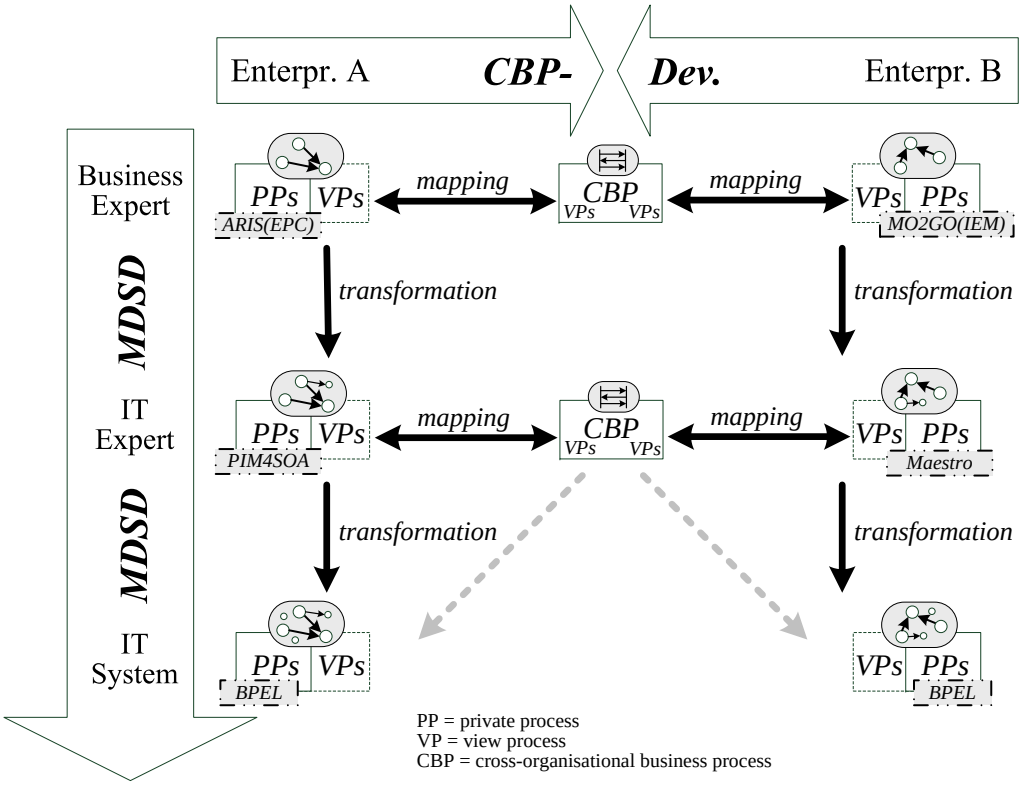


Figure 6.1: Scenario realizing CBP modelling and execution

Figure 6.1 depicts a concrete scenario that implements CBP modelling and execution. We contributed to its realization in the ATHENA IP project (more details can be found in [117]). Enterprises *A* and *B* develop models for their processes (Private Processes (PPs), View Processes (VPs), and Cross-organisational Business Processes (CBPs)) at three levels of abstraction, i.e. business expert, IT expert, and IT system level. Vertical transformations, like presented in [26, 27], encode knowledge about the architecture and the platform in order to transform models from higher to lower abstraction level. For example ARIS models (extended Event-driven Process Chains (eEPCs) [157]) are transformed to models conforming to PIM4SOA [30]. Finally BPEL4WS code is generated from the PIM4SOA models. Enterprises *A* and *B* use different modelling tools and languages at the various abstraction levels. To develop CBPs, both enterprises have to provide public parts of their models as a basis for discussion during collaborative modelling. Hence, mappings between the different representation formats like ARIS and Integrated Enterprise Modelling (IEM)[184, 276] or PIM4SOA and Mae-

stro [260] have to be developed.

To summarize, there are two main issues that prevent a smooth realization of such a MDSD scenario:

- *Exchange of models:* Models are shared across inter-organisational relationships. Hence, mappings have to be developed between the various enterprises' modelling languages and tools. This is necessary to achieve a shared understanding of CBP and to enable collaborative MDSD.
- *Evolution of model transformations*: Over a period of time enterprises will apply new (versions) of modelling languages, metamodels, and modelling styles. Therefore, existing transformations have to be maintained, adjusted, or redeveloped.

### 6.1.2 Problem Statement

Managing and developing model transformations are error-prone and long lasting tasks. Since model transformations are a kind of metaprogramming, they require a deep knowledge of all the underlying modelling technology. It is beneficial to provide support with a solution that automates model transformation development and adjustment tasks. However, the core principles of modelling (i.e. representing information about real world things in models) and problems of such automation solutions remain the same. The core barriers to model exchange and maintenance of model transformations are multiple representation formats and different modelling styles, serving the particular application.

- *Different representation format*: The trend towards the use of Domain Specific Languages (DSLs) leads more and more people to create their own Domain Specific Models (DSMs). This naturally results in a variety of different languages and metamodels. To exchange models that conform to these various metamodels (abstract syntax), model transformations have to be developed. Often there are multiple metamodels for the same modelling language. Also time and again new versions of metamodels, e.g. the metamodels for UML 1.x and UML 2.x, are released. Whenever new versions replace the old ones, new model transformations have to be developed and existing model transformations have to be adjusted. Though visual representations (concrete syntax) should be decoupled from internal representation (abstract syntax), different concrete syntax is often considered in model transformations to provide e.g. views on models.
- *Different semantics*: Since the semantics of modelling languages' concepts is rarely formally specified (in the UML specification this is plain English), different people and organisations can associate different semantics with the same concepts used in the metamodel. This is often done by applying special modelling styles and representation guidelines. Again, model transformations have to be specified enabling sensible exchange of models according to the respective interpretations.

## 6.2 The Ontology-based Model Transformation Approach

To address these issues we have developed the Ontology-based Model Transformation (OntMT) approach. OntMT facilitates methods to generate and adjust model transformations despite of structural and semantic differences of metamodels. Different representation formats and different semantics are overcome by applying semantic technology

of the Ontologyware and Ontology Engineering TS (Ontology TS). In OntMT metamodels are annotated through the elements of a Reference Ontology (RO) and reasoning is applied to the RO and the annotations. OntMT allows to generate and adjust common model transformations automatically in order to apply MDSD in the Modelware and Model-based Technology TS (MDA TS).

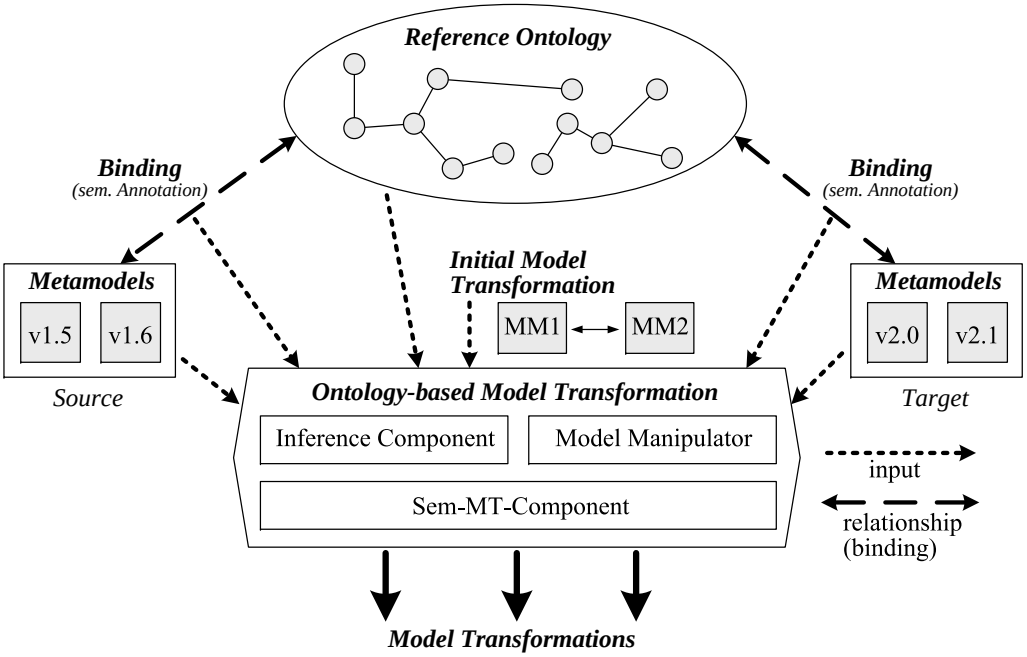


Figure 6.2: Ontology-based model transformation: overall approach

Figure 6.2 depicts the overall approach of OntMT. Different versions of metamodels are bound to a RO of a certain domain. Bindings (semantic annotations) specify the semantic mapping of metamodels to the semantics of their concepts, i.e. to the RO. To generate model transformations for various model transformation languages and to adjust existing model transformations, OntMT makes use of reasoning mechanisms. The metamodels and the RO are given, while the bindings of the metamodels to the RO have to be specified. Finally, an initial model transformation is needed. For the evolution of model transformations the initial model transformation is the model transformation that shall be reused or adjusted (see Section 6.2.2). The initial model transformation (e.g. from metamodel v1.5 to metamodel v2.0) encodes transformation rules and the semantics of the model transformation. If for example the metamodel v2.0 is replaced with a version 2.1, only the delta between these metamodels has to be considered to adjust the existing model transformation. The new model transformation is generated by substituting the concepts of metamodel v2.0 with the concepts of metamodel v2.1 in the initial model transformation. In the case of automated mapping generation, a bootstrapping algorithm generates an initial model transformation (see Section 6.2.1).

### 6.2.1 Automated Generation of Model Transformations

Model transformations between various modelling languages can be automatically derived and generated with the OntMT approach. In this section we describe the procedure to generate mappings, i.e. semantically identical model transformations, between

two modelling languages $A$ and $B$. We illustrate the procedure via a strongly simplified example, where $A$ and $B$ both consist of two concepts: $A = \{Process, Task\}$ and $B = \{EPC, EPCElement\}$.

For both languages there exists an abstract syntax $N_A/N_B$ in various Technological Spaces (TSs): $A$ has (like $B$) an abstract syntax in the MDA TS $N_{A-mda}$ and the Ontology TS $N_{A-ont}$ which are synchronized. Thus, one can work with the syntax and the capability of that TS, that is better suited for solving a problem (see Figure 6.3). The semantics of the concepts is described by the means of the semantic domain $SD$ and its notation in a RO $N_{RO}$ (e.g. OWL) respectively. The semantics of the languages is defined by semantic mappings from the languages to the semantic domain: $M_A : A \rightarrow SD$ and $M_B : B \rightarrow SD$. In this example, the semantic domain is given as $SD = \{Activity, Action\}$, while the semantic mappings are $M_A(Process) = Activity$, $M_A(Task) = Action$, $M_B(EPC) = Activity$, and $M_B(EPCElement) = Action$.

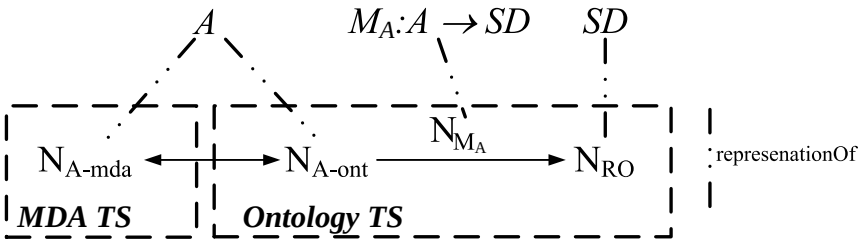


Figure 6.3: Modelling language, semantic mapping, semantic domain and their representations

The ontological grounding[1] is a notation of the semantic mapping from $N_{A-ont}$ to $N_{RO}$. The goal of the transformation to be generated is to define 'identity' relationships between the concepts of $A$ and $B$. The bi-directional model transformation $MT_{mapAB} : A \leftrightarrow B$ between $A$ and $B$ has the following semantics: $M_{MTmapAB}(MT_{mapAB}) = id$, i.e. $\forall(a, b) \in MT_{mapAB} : M_A(a) = M_B(b)$.

The generation procedure works on the model of the model transformation and the models of the modelling languages. It exploits the ontological grounding to the reference ontology. On the basis of reasoning results gained in the Ontology TS ($\{Process \cong EPC$, $Task \cong EPCElement\}$), modification operations are called to obtain the new model transformation working solely on the model of the model transformation. To generate the model transformation $MT_{mapAB}$, the following steps are performed (see Figure 6.4):

1. A bootstrapping algorithm generates the model transformation $MT_{mapAA}$, with $\forall(a, a') \in MT_{mapAA} : a = a'$. This bootstrapping step is necessary to obtain a first model of the model transformation $N_A$ to $N'_A$[2], which only has to be adjusted by modification operations. Assuming the same ontological grounding $M_A$ for $N_A$ and $N'_A$, the bootstrapping model transformation is an $id : M_{MTmapAA}(MT_{mapAA}) = id$. In our example the model transformation relations identified by the bootstrapping are $MT_{mapAA} = \{(Process, Process), (Task, Task)\}$.

[1]The definition of the ontological grounding is a semantic annotation comprising static semantics of the metamodels, i.e. the semantics of the concepts and an ontology respectively.

[2]Such a mapping can be generated on the basis of a metamodel in the MDA TS. The appropriate mapping rules are generated by traversing the metamodel via its composite aggregation (in short composition) relationships.

2. The inference derives relationships between $N'_A$ and $N_B$ in the Ontology TS. This is possible, since both $N'_A$ and $N_B$ are mapped to the same reference ontology $N_{RO}$. It is automatically computed, how the concepts of $N'_A$ can be substituted by semantically identical concepts of $N_B$ ($\sigma : MT \rightarrow MT$, where $MT$ is the set of all model transformations). Those relationships can be transferred to the MDA TS as the modelling languages $A$ and $B$ have synchronous representations in both MDA TS and Ontology TS. The substitutions computed for our example are $[EPC/Process]$ and $[EPCElement/Task]$.

3. Finally, the concepts of $N'_A$ are substituted with the concepts of $N_B$ ($\sigma(MT_{mapAA}) = MT_{mapAB}$) in the model of $MT_{mapAA}$ and we obtain a model of the model transformation $MT_{mapAB}$ with $M_{MTmapAB}(MT_{mapAB}) = id$. The substitution is performed via modification operations on the model of the model transformation $MT_{mapAA}$ in MDA TS. In the example the following model transformation relations are generated: $MT_{mapAB}\{(Process, EPC), (Task, EPCElement)\}$.

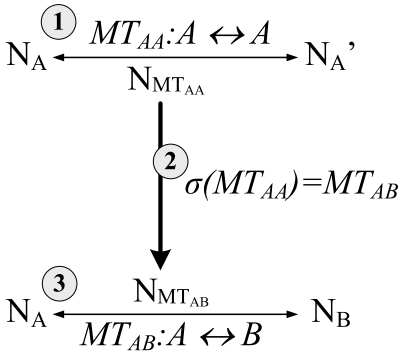


Figure 6.4: Procedure of automated mapping generation

### 6.2.2 Evolution of Model Transformations

OntMT also fosters the evolution and reuse of existing model transformations. Instead of performing the bootstrapping step, the procedure for model transformation evolution takes the model transformation that shall be reused as input (see Figure 6.5). This initial model transformation $MT_{mapAB} : A \leftrightarrow B$ encodes knowledge about how modelling language $A$ is translated into $B$. The steps ② and ③ are the same as for automated mapping generation. In step ②, a substitution $\sigma(MT_{mapAB}) = MT_{mapAC}$ is computed on the basis of inference results. Step ③ applies this substitution and generates a new version of the initial model transformation $MT_{mapAC} : A \leftrightarrow C$. The bootstrapping step helps to extend OntMT to scenarios where existing model transformations are adjusted. Avoiding to derive model transformations directly from ontologies results in a more flexible and well-structured architecture. OntMT can both generate new model transformations and reuse knowledge encoded in existing transformations. Issues concerning the model transformation, like checking if its model conforms to the QVT metamodel or considering the cardinality of associations' ends, are all dealt within the MDA TS. The modification operations are invoked on the basis of the reasoning results and the application of heuristics.

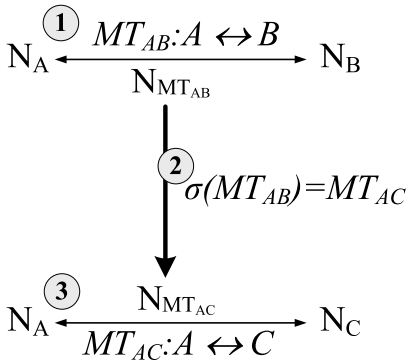


Figure 6.5: Procedure of model transformation evolution

## 6.3 Components of Ontology-based Model Transformation

This section presents the components and concepts of OntMT realized as a tool for a semantic-enabled modelling and development suite, its parts, and its functionality.

### 6.3.1 Components of a Sem-MT-Tool

OntMT, as part of our vision of a semantic-enabled modelling and development suite, is realized as Sem-X-Tool (see Figure 6.6) [29]. The infrastructure provides basic functionality including a bridge ⑥ between models of the MDA TS and application ontologies of the Ontology TS (like it is described in [39]) and an inference component, which can be individually configured and used by Sem-X-Tools registered at the infrastructure. Sem-X-Tools, like the Sem-MT-Tool presented in this paper, are built on top of the infrastructure. They consist of a model manipulator, a Sem-X-Component, and a rule set. The model manipulator reads, creates, modifies and deletes models of the model repository ①. It delivers information about models to the Sem-X-Component ② and provides interfaces for model manipulation ③. The Sem-X-Component implements the core functionality of a Sem-X-Tool. It makes use of the reasoning results gained by inferring ontologies and computes the respective model manipulation ④. Since Sem-X-Tools are based on different relationships between the ontologies' elements, each Sem-X-Tool has its own set of reasoning rules.

Figure 6.7 shows the architecture of the components building the Sem-MT-Tool which is an instantiation of the Sem-X-Tool. The model manipulator provides functionality via three interfaces: one that identifies the concepts of a metamodel that have to be substituted in a model transformation, one that performs a substitution of a metamodel's concepts in the model transformation, and one that provides validation functionality for the generated model transformation. The inference component provides an interface for accessing the reasoning results, i.e. the relationships between the metamodel elements. The Sem-MT-Component is the component of the Sem-MT-Tool, which connects the inference results of the Ontology TS to concrete modification actions on the models of the model transformation in the MDA TS.

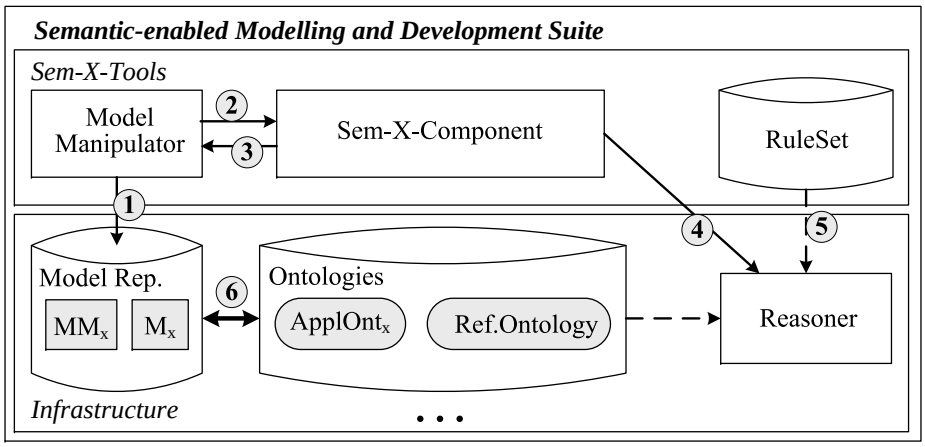


Figure 6.6: OntMT as part of a semantic-enabled modelling and development suite

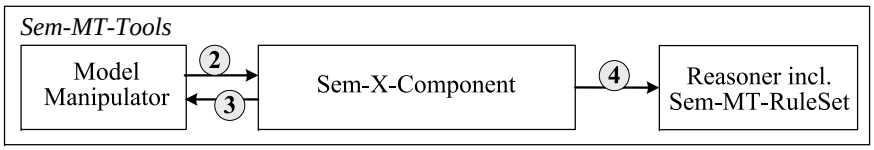


Figure 6.7: Sem-MT-Tool component architecture

### 6.3.2 Architecture of Ontology-based Model Transformation

To validate our approach we have implemented a prototype that realizes OntMT. The following section provides more details about the used techniques, the architecture, and the implementation. Therefore the architectural figures depict the technologies we have used to implement our prototype.

#### Inference Component

Figure 6.8 depicts an architectural view on the inference component of OntMT. The inference component consists of a knowledge base and a reasoner. The base graph contains all facts of the knowledge base before the reasoning, i.e. the RO, application ontologies[3], and the ontological groundings. The reasoner is triggered by rules specific to the Sem-MT-Tool, and computes the inference graph on the basis of the base graph. As the result of the reasoning, the knowledge base contains information about all relationships that are important for OntMT. These are the relationships between the application ontologies.

In [51, 191, 267], equivalence, containment, and overlap are described as the main relationships for mapping ontologies. The inference component identifies (for OntMT) these relationships between the ontology elements.

- *Equivalence* (≡) means that the connected elements represent the same aspect of the real world. An element of an application ontology corresponds to an element in the reference ontology or can be precisely expressed by a composition of elements. We will refer to this relationship by the relationship type <*equal*>.

[3] An application ontology corresponds to a metamodel in the Ontology TS.

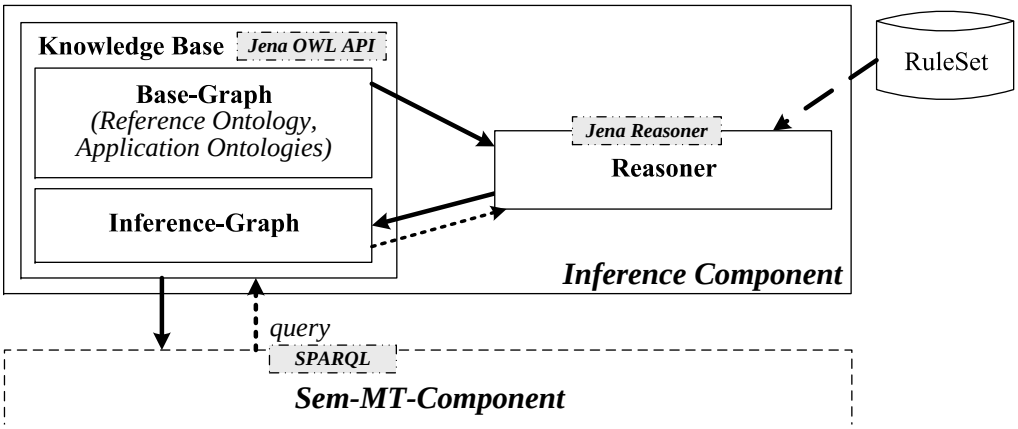


Figure 6.8: Inference component

- *Containment* ($\sqsubseteq, \sqsupseteq$) states that the element in one ontology represents a more specific aspect of the world than the element in the other ontology. The relationship can be defined in one or the other direction, depending on which concept is more specific. When an element is not sufficiently refined we use the relationship <*general*>. When an element is described at a level of refinement that does not match the level of refinement of the other ontology we use the relationship <*special*>.
- *Overlap* ($o$) states that the connected elements represent different aspects of the world, but have an overlap in some respect. This relationship is of the type <*overlap*>.

The relationships are also used for the ontological groundings by specifying mappings between the application ontologies and reference ontologies. This is possible, since the model elements are represented in application ontologies via the UML to OWL mapping described in the ODM standard [230, p.201ff]. More details can be found in Section 6.4.3.

**Implementation** In our current prototype we use the Jena ontology API[4] to create and handle ontologies. The inference is realized through rules deployed to the rule engine included in Jena [138]. Jena also uses this rule engine to provide (partial) RDF(S) and OWL reasoning[5]. The rule 2 in Listing 6.1 for example states, that if $A$ overlaps with $B$ and $B$ is an intersection of $C$ and $D$ then $A$ overlaps with $C$ and $D$. The inference results are obtained with SPARQL Protocol and RDF Query Language (SPARQL), which queries the knowledge base for the relationships between the application ontologies.

Listing 6.1: Sample reasoning rules

```
rule 1: A o B ∧ B ⊑ C → A o C
rule 2: A o B ∧ B ≡ C ⊓ D → A o C ∧ A o D
```

The decision to use the Jena framework and its rule based reasoning support for the prototype implementation was mainly based on two arguments. First, it better met our requirements, which were mainly a combination of TBox reasoning, rule support, and

[4] http://jena.sourceforge.net/tutorial/RDF_API
[5] http://jena.sourceforge.net/inference

good documentation, than other open source projects. Second, the Jena framework provides the possibility to integrate other reasoners like Pellet [245] or future implementations of ontology mapping approaches using local domains like Context OWL (C-OWL) [49].

### Model Manipulator

The model manipulator provides modification operations on model transformations. It implements a language for model transformation modification that is used by the Sem-MT-Component to trigger the modification of the model transformations via modification programs. The semantics of this model transformation modification language treats model transformations as models. The facts that model transformation languages like QVT are represented through metamodels and model transformation programs are models allow higher-order transformations, like transformations taking other transformations as input and producing transformations as output [37].

Due to the gap between the concepts of DSLs and metamodels implementing these DSLs, the semantics of the model transformation modification language needs to provide mechanisms to allow the Sem-MT-Component to adapt a modification program to the best possible solution. Hence the semantics is divided into a *modification semantics* and a *checking semantics* (see Figure 6.9).

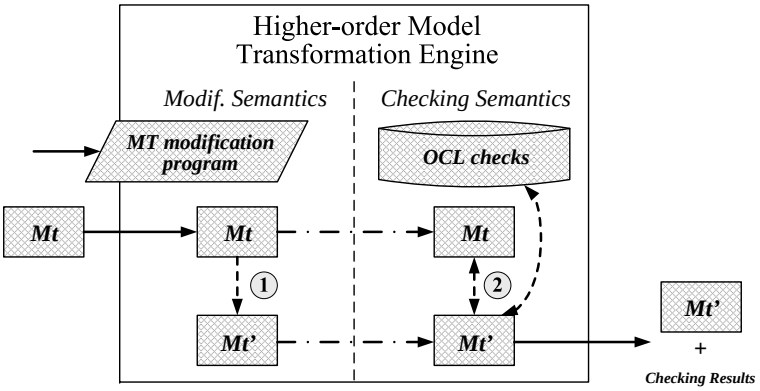


Figure 6.9: Semantics of model transformation modification

**Modification semantics** The modification semantics defines how the modification of model transformations, which is specified in modification programs, is executed. A simplified picture, that helps to work with the model transformation modification language, is to imagine the modification program as a substitution. The elements of the modification program's source metamodel are substituted with the elements of the target metamodel. The detailed implementation realizing the semantics is encapsulated in a separate component of the model manipulator. Currently realized substitution operators provide functionality for one-to-one, one-to-many, and removal substitutions of both classes and properties. In the following we give an outline of the substitution operators' functionality via short examples.

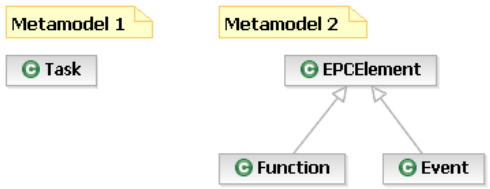


Figure 6.10: Two example metamodels

Listing 6.2: Example model transformation specification (notation similar to QVT)

```
1 relation rule {
2   checkonly domain l_mm var1:Task { };
    enforce domain r_mm var1':Task { };
4 }
```

- *One-to-one substitution*: If the *Task* in the sample model transformation rule shall be substituted by *EPCElement* in the right-hand model, then the one-to-one substitution for classes has to be applied: $[EPCElement/Task]$. The result of applying this one-to-one substitution to the transformation rule of Listing 6.2 is as follows:

Listing 6.3: Model transformation after applying one-to-one substitution

```
1 relation rule' {
2   checkonly domain l_mm var1:Task { };
    enforce domain r_mm var1':EPCElement { };
4 }
```

- *One-to-many substitution*: If the *Task* in the sample model transformation rule (see Listing 6.2) shall be substituted by *Function* and *Event*, then the one-to-many substitution for classes has to be applied: $[\{Function, Event\}/Task]$. The result of the one-to-many substitution is not so obvious like the result of the one-to-one substitution, since the model transformation rule has to be duplicated. Details about one-to-many substitutions can be found in Section 6.4.2.

Listing 6.4: Model transformation after applying one-to-many substitution

```
1 relation rule'_a {
2   checkonly domain l_mm var1:Task { };
    enforce domain r_mm var1':Function { };
4 }

6 relation rule'_b {
    checkonly domain l_mm var1:Task { };
8   enforce domain r_mm var1':Event { };
  }
```

- *Removal substitution*: A removal substitution is sensibly applied when an element of the source metamodel cannot be substituted by any element of the target metamodel. If e.g. a removal substitution $[-/Task]$ is applied to the *Task*, the whole transformation rule of Listing 6.2 would be removed from the model transformation.

**Checking semantics** The checking semantics tests the generated model transformations for so-called problems, which can occur by applying the modification semantics. This is necessary, since the application of the modification semantics substitutes properties and classes in the model transformation separately and may perform substitutions that change the semantics of the model transformation.

One set of problems affects the consistency of model transformation programs with respect to the model transformation language, i.e. the generated model transformations are not valid and cannot be executed. Another kind of problems is caused, when knowledge encoded into the original model transformation is not preserved or lost. In general, problems are detected via OCL [229] constraints. Only for a few problems, where additional information about the execution of the modification is needed, we extend this mechanism with information from the modification execution. The following list gives an overview of problems that can occur:

- *Substitution of property failed*: This problem occurs, when the model transformation modification program did not specify a substitution for a property that is used in the model transformation.
- *Property is not part of class*: The generated model transformation would require a property to be part of a class, what is not the case in the respective metamodel (model types as described in [279] do not match).
- *Further checks*: The checking semantics comprises further OCL constraints to check the validity of generated model transformations. E.g. it is checked if classes referenced by OTEs belong to the metamodel of the respective domain. Another problem occurs when the type of a PTI's value does not conform to the type of the PTI's referred class. For more details see Section 6.4.2.

**Architecture and Implementation** The model manipulator component is divided in a front end and a back end (see Figure 6.11). The front end primarily conducts tasks that depend on the source language, while the back end deals with all issues specific to the target language. The metamodels and the bootstrap model transformation are brought into an intermediate representation format by the scanner and the parser. The substitution algorithm performs the substitutions proposed by the Sem-MT-Component. The validator checks whether any performed substitution leads to problems in the new model transformation.

Our prototypical implementation of the model manipulator is based on Eclipse. It uses the Eclipse Modeling Framework (EMF) [75]. EMF allows the model manipulator to treat the metamodels and the model transformations with a single model manipulation API. This reflective API allows to handle EMF objects generically, regardless of which metamodel they belong to (EMOF, QVT, OCL, etc.). The metamodels are instantiations of the EMF EMOF implementation and the model transformation models are treated as instantiations of the EMF QVT relational implementation. Since the first final adopted version of the QVT standard [233] contains some inconsistencies we had to make some adjustments which are documented in our implementation.

- Parser: The implementation of the parser makes use of the ANother Tool for Language Recognition (ANTLR) parser generator [242, 243] and parses a QVT relational textual syntax into EMF QVT relational models. It has been made available under General Public License (GPL) via the QVT Parser project [273].

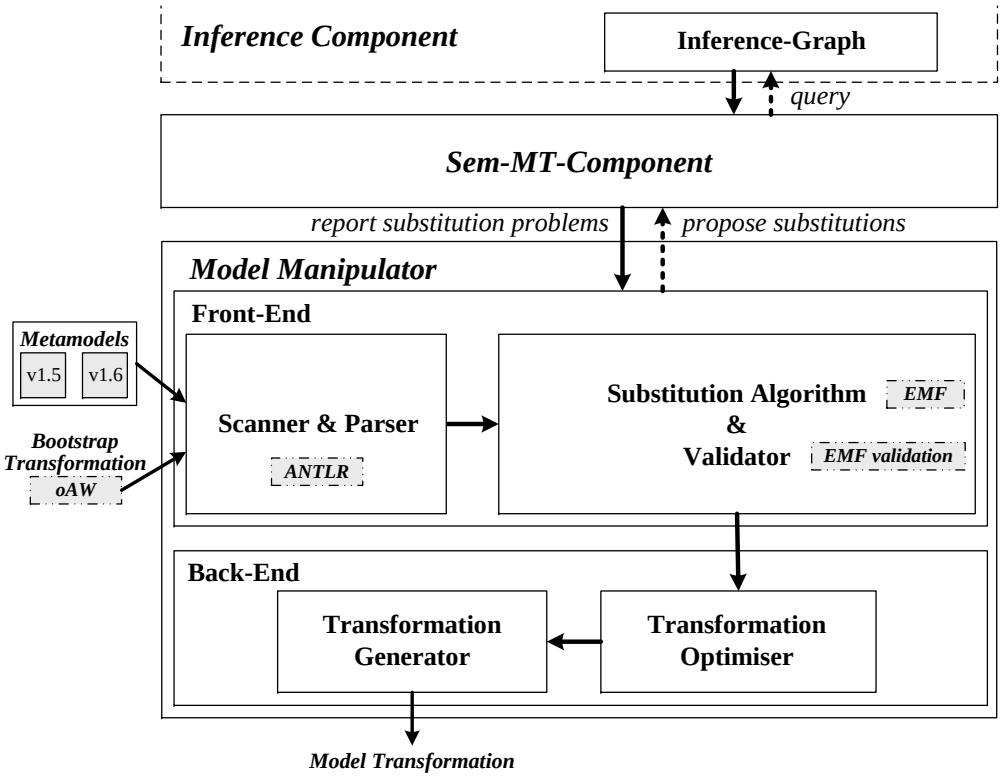


Figure 6.11: Model manipulator

- A prototype of the model manipulator implementation is part of the OntMT project [272]. The substitution algorithm is totally based on the EMF API. The validation component uses the EMF validation framework to check the EMF model of the generated QVT relational transformation with OCL constraints. Since we use EMF, the Eclipse Validation Framework [76] is a consequent choice for implementing and performing the OCL checks. The results of EMF validation lend themselves very well to determine the exact position of problems or inconsistencies in a model transformation. OCL constraints, checking whether a model transformation is syntactically correct, can be automatically generated from the QVT metamodel. It is checked whether the model transformation conforms to the grammar that the QVT metamodel was generated from. With further manually implemented OCL constraints the model manipulator checks whether the generated model transformation is valid and can be executed or whether knowledge has been lost through the substitution.

- The bootstrapping generates a QVT relational model transformation from a metamodel expressed in MOF. It is implemented with templates expressed in the oAW Xpand language [81, p.87ff] and available via the OntMT project. The bootstrapping is well integrated in the model manipulator, since EMF models can be used for oAW code generation. In fact the same metamodels that are used by the QVT Parser and the model manipulator are also used for the bootstrapping.

### Sem-MT-Component

The Sem-MT-Component implements the core part of the OntMT approach. It provides the main functionality of the Sem-MT-Tool. It makes use of the inference results of the Ontology TS and computes modifications programs for the generation and evolution of model transformations in the MDA TS. Algorithm 6.1 illustrates the correlation algorithm implemented by the Sem-MT-Component to generate the new model transformations.

Table 6.1: Correlation algorithm

```
 1: function CORRELATION(initialMT, oldMM, oldMM) : ModelTransformation
 2:     Cs ← FINDCONCEPTS(initialMT, oldMM)                    ▷ concepts to substitute
 3:     CSss ← ∅                                  ▷ concepts and possible substitutions
 4:     SP ←⊥                                                  ▷ substitution proposal
 5:     SPs ← ∅                                              ▷ set of substitution proposals
 6:     tempMT ←⊥                                  ▷ last generated model transformation
 7:     newMT ←⊥                                     ▷ generated model transformation
 8:
 9:     for all c ∈ Cs do
10:         CSss ← CSss ∪ FINDSUBSTITUTIONS(c, oldMM, newMM)
11:     end for
12:
13:     repeat
14:         SP ← CALCULATESUBSTITUTIONPROPOSAL(SP, SPs, Cs, CSss)
15:         tempMT ← PERFORMSUBSTITUTION(initialMT, SP)
16:         SP ← VALIDATESUBSTITUTION(tempMT, SP)
17:         SP ← RATESUBSTITUTION(SP)
18:         SPs ← { SP } ∪ SPs
19:     until SP =⊥ ∨ SP.problems = ∅
20:
21:     SP ← CHOOSESUBSTITUTIONPROPOSAL(SPs)
22:     newMT ← PERFORMSUBSTITUTION(SP)
23:
24:     return newMT
25: end function
```

The Sem-MT-Component takes as input an initial model transformation, the metamodel which has to be substituted in the model transformation, and the new metamodel. In a first step (line 2) it requests the model manipulator to compute a set of all classes and properties *Cs* that have to be substituted in the initial model transformation. Second, it invokes the inference component to obtain possible substitutions for all classes and properties to substitute *CSss* (line 10). Next, the computation of a substitution proposal *SP* begins. A substitution proposal contains the model transformation modification program, the problems that occur by applying the substitutions to a model transformation, and a rating of the performed substitutions. After a substitution proposal is calculated by the Sem-MT-Component (line 14), the model manipulator performs the substitution (line 15) and validates (line 16) the generated model transformation. Then the substitution proposal is rated by the Sem-MT-Component (line 17). The Sem-MT-Component tries to compute alternative substitution proposals until the application of a substitution proposal does not lead to any problems, or no further substitution proposals can be found. Finally, the Sem-MT-Component chooses the substitution proposal with the

best rating from all computed substitution proposals *SPs* (line 21) and the new model transformation is generated on the basis of this substitution proposal (line 22).

The choice of the substitution proposal, which is used to generate the new model transformation, is based on the ratings of the substitution proposals. A rating of a substitution proposal is a measure of the generated model transformation's quality. The rating is based on factors that are measured for each substitution proposal like the problems occurring in the substitution proposal, number of concepts that could be substituted, relationships used for substitution, and the position of the problems and used relationships in the model transformation.

Concrete ratings always depend on the purpose OntMT is used for. For the different application scenarios separate metrics are defined. A metric, which was developed for automated mapping generation, will put more emphasis on an executable model transformation than on the relationships used for substitution. If OntMT is used to support developers in adjusting their model transformations, OntMT will only make suggestions to the developer. Hence, the metric puts more emphasis on exact substitutions of metamodel elements than on the execution of the new model transformation.

**Implementation** The OntMT project currently provides a simple implementation of the correlation algorithm, which is described in Algorithm 6.1. However, an automated synchronisation of the modelling and the reasoning world (see ⑥ in Figure 6.6) is not yet fully integrated. We are developing a prototype that synchronizes EMF and Jena OWL models and allows to answer SPARQL like queries on EMF models with reasoning support. The synchronization mechanism makes use of the UML to OWL mapping described in the ODM standard [230, p.201ff]. However, we plan to replace our prototype with an implementation of the Eclipse EMF Ontology Definition Metamodel (EODM) project [77]. This project aims to provide inference capabilities for OWL models implemented in EMF and model transformations of RDF/OWL to other modelling languages such as UML.

## 6.4 Realization of Ontology-based Model Transformation

In this section we present concepts and techniques to realize the components described in Section 6.3. These are bootstrapping rules to generate initial model transformations (Section 6.4.1), a higher-order model transformation language for QVT Relations model transformations (Section 6.4.2), representation and reasoning techniques to infer relationships between metamodels (Section 6.4.3), and the correlation and rating to realize the Sem-MT-Component (Section 6.4.4).

### 6.4.1 Model Transformation Bootstrapping

The model transformation bootstrapping generates endogenous QVT Relations model transformations that map metamodels on themselves. Therefore the bootstrapping exploits the composite aggregation hierarchy in the metamodel and maps classes, attributes, and associations. In this section we present the mapping rules that we developed for the model transformation bootstrapping.

### Bootstrapping Rules

***[Rule 1]* – Mapping Classes** For each class *C* in the metamodel one top relation is generated in the bootstrapped model transformation. This relation *CtoC* maps the class *C* on itself. Further, a key is generated for the class *C*. Listing 6.5 depicts the code that is generated with rule 1.[6]

Listing 6.5: Bootstrapping: mapping classes

```
1  key MM:C { };
2
   top relation CtoC {
4    checkonly domain mm c:C { } ;
     enforce domain mm_ c_:C { } ;
6  }
```

***[Rule 2.1]* – Mapping Mandatory Attributes** Each mandatory attribute *C.man* is directly added as an attribute to the top relation generated for the class *C*. More exactly, *C.man* is added to all OTEs that map *C* as a PTI. The value of the PTI is a *Variable* or an *OclExpression*. The *Variable* and its type *<typeOf(C.man)>* have to be declared in the relation. Since *C.man* is a mandatory attribute, it is also added to the key generated for *C*. This rule takes into account all attributes of a class, including the attributes a class inherits from its superclasses. Listing 6.6 shows the code that is generated with rule 2.1.

Listing 6.6: Bootstrapping: mapping mandatory attributes

```
1  key MM::C { man };
2
   top relation CtoC {
4  man_var : <typeOf(man)>;
     checkonly domain mm c:C { man = man_var };
6    enforce domain mm_ c_:C { man = man_var };
   }
```

***[Rule 2.2]* – Mapping Optional Attributes** For an optional attribute *C.opt* of a class *C* the bootstrapping generates a new relation, which is not a top relation. This relation is constructed analogously to the relation, to which a mandatory attribute was added. The new relation is referenced by the *where*-clause of the top relation, that was generated from *C*. Listing 6.7 shows the code that is generated with rule 2.2.

Relations for optional attributes are only generated for the classes where these optional attributes are defined. Relations are not generated for classes that inherit these optional attributes. However, classes inheriting an optional attribute reference the relation that was generated for the optional attribute via a *where*-clause.

[6]The QVT Relations code that is generated by generation templates is depicted in italics.

Listing 6.7: Bootstrapping: mapping optional attributes

```
1  top relation CtoC {
2    checkonly domain mm c:C { } ;
     enforce domain mm_ c_:C { } ;
4    where
     {
6      CtoC_opt(c,c_);
     }
8  }

10 relation CtoC_opt {
     opt_var : <typeOf(opt)>;
12   checkonly domain mm c:C { opt = opt_var } ;
     enforce domain mm_ c_:C { opt = opt_var } ;
14 }
```

***[Rule 3.1]* – Mapping Mandatory Composition Associations** From this kind of association at most one is allowed to exist for one class. If there exist more than one mandatory composition associations in one class, the metamodel is inconsistent and the generated model transformation will not be able to produce usable results.

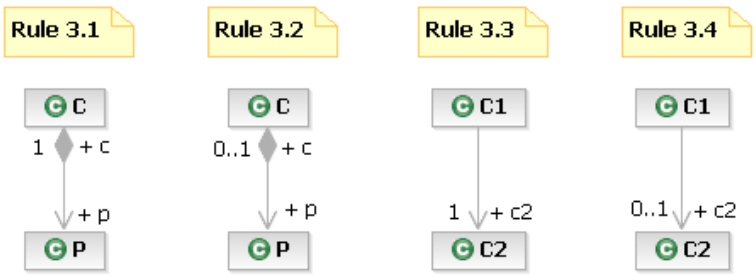


Figure 6.12: Bootstrapping patterns for rules 3.1-3.4

Let us consider an association (cp. Figure 6.12) between a containing class *C* and a contained class *P* as two directed associations *C.p* (from *C* to *P*) and *P.c* (from *P* to *C*). The association *C.p* is handled like a normal association (see later rule 3.3 and rule 3.4). *P.c* is added like a mandatory attribute to the top relation, which was generated for *P*. Additionally the *when*-clause of the *P* relation references all top relations that are used for mapping *C*. If the *when*-clause of the *P* relation has to reference more than one top relation of *C*[7], then the references are combined by the boolean operator *or*. In Listing 6.8 the second top relation that was generated for *C* is *CtoC_example*. This rule takes into account all mandatory composition associations of classes, including the associations the class inherits from its superclasses.

[7]This case can occur, when rule 3.2 was applied to the superclass *C*.

Listing 6.8: Bootstrapping: mapping mandatory composition associations

```
top relation CtoC {
  checkonly domain mm c:C { } ;
  enforce domain mm_ c_:C { } ;
}

top relation CtoC_example {
  ...
}

top relation PtoP {
  checkonly domain mm p:P { c = c_var:C { } } ;
  enforce domain mm_ p_:P { c = c_var_:C { } } ;
  when
  {
    C2C(c_var,c_var_) or C2C_example(c_var,c_var_);
  }
}
```

***[Rule 3.2]* – Mapping Optional Composition Associations** It is possible that for one class multiple optional containment associations exist. Let us consider an association (cp. Figure 6.12) between a containing class *C* and a contained class *P* as two directed associations *C.p* (from *C* to *P*) and *P.c* (from *P* to *C*). The association *C.p* is handled like a normal association (see later rule 3.3 and rule 3.4).

For *P.c* a new top relation is constructed. The *when*-clause references the top relation, which was generated from *P*. Additionally the *when*-clause of the new relation references all top relations that are used for mapping *C*. If the *when*-clause of the new relation has to reference more than one top relation of *C*, then the references are combined by the boolean operator *or*. This rule takes into account all optional composition associations of classes, including the associations the class inherits from its superclasses.

Listing 6.9: Bootstrapping: mapping optional composition associations

```
top relation CtoC {
  checkonly domain mm c:C { } ;
  enforce domain mm_ c_:C { } ;
}
top relation CtoC_example {
  ...
}
top relation PtoP {
  checkonly domain mm p:P { } ;
  enforce domain mm_ p_:P { } ;
}
top relation PtoP_c {
  checkonly domain mm p:P { c = c_var:C { } } ;
  enforce domain mm_ p_:P { c = c_var_:C { } } ;
  when
  {
    PtoP(p,p_);
    CtoC(c_var,c_var_) or C2C_example(c_var,c_var_);
  }
}
```

***[Rule 3.3]* – Mapping Mandatory Associations** For bootstrapping mechanism bidirectional normal associations can be reduced to two directed associations. Hence, we present a pattern of how to generate QVT Relations model transformation code for directed associations. For bootstrapping mechanism bidirectional normal associations can be reduced to (two) directed associations. Thus we will present a pattern of how to generate QVT model transformation code for a directed association.

If we consider a mandatory directed association *C1.c2* from class *C1* to *C2* (cp. Figure 6.12), *C1.c2* is added as a mandatory attribute to the top relation generated for *C1*. The OTE that maps *C2* has to contain all mandatory attributes as PTIs (i.e. the attributes occurring in the key of *C2*). This rule takes into account all optional composition associations of classes, including the associations the class inherits from its superclasses.

Listing 6.10: Bootstrapping: mapping mandatory associations

```
1  key MM::C2 { man };
2
   top relation C1toC1 {
4    man_var : <typeOf(man)>;
     checkonly domain mm c1:C1 {
6      c2 = c2_var:C2 { man = man_var }
       } ;
8    enforce domain mm_ c1_:C1 {
       c2 = c2_var_:C2 { man = man_var }
10     } ;
   }
```

***[Rule 3.4]* – Mapping Optional Associations** Like for mandatory associations we present a pattern of how to generate QVT Relations model transformation code for directed optional associations (cp. Figure 6.12). If we consider an optional directed association *C1.c2* from class *C1* to *C2*, for *C1.c2* a new relation is generated like for an optional attribute. The OTE that maps *C2* has to contain all mandatory attributes as PTIs (i.e. the attributes occurring in the key of *C2*). The new relation is referenced by the *where*-clause of the top relation, which was generated from *C1*.

This rule generates only relations for classes where these optional associations are defined. However, the generated relation is referenced by the top relations of all classes, that contain the respective optional association.

Listing 6.11: Bootstrapping: mapping optional associations

```
1  key MM::C2 { man };
2
   top relation C1toC1 {
4    checkonly domain mm c1:C1 { } ;
     enforce domain mm_ c1_:C1 { } ;
6    where
     {
8      C1toC1_opt(c1,c1_);
     }
10 }

12 relation C1toC1_c2 {
     man_var : <typeOf(man)>;
14   checkonly domain mm c1:C1 {
       c2 = c2_var:C2 { man = man_var }
```

```
16        } ;
      enforce domain mm_ c1_:C1 {
18        c2 = c2_var_:C2 { man = man_var }
          } ;
20    }
```

**Summary**

In our approach of OntMT we use the bootstrapping rules to generate initial model transformations. This bootstrapping is a prerequisite to realize the automated generation of model transformations scenario. We have implemented the bootstrapping rules as described in this section in the OntMT project [272].

### 6.4.2 Higher-order Model Transformation Language

OntMT fosters reuse of knowledge encoded into model transformations and reuse of models through automated mapping generation. Beneath reasoning techniques used in the inference component, OntMT is based on a model manipulator implementing model transformation modification techniques. In order to successfully trigger the modification mechanisms on the basis of inference results (this is done by the Sem-MT-Component), it is necessary to provide a language for model transformation modification at an appropriate level of abstraction. This language has to lend itself for automating reuse. In this section we present a syntax to specify those modifications as dependencies between metamodels and a semantics to execute those model transformation modifications.

**Problem Description**

Model transformations encode patterns about how to transform concepts of one modelling language into concepts of another modelling language. Model transformations are a kind of metaprogramming, since their specification uses the meta-information of modelling language definitions. So, if we want to reuse model transformations, these model transformations have to be adjusted to the various software development contexts, each with its own particular DSLs and metamodels representing the DSLs. As developing model transformations is a demanding and time consuming task and model transformations can be very complex programs, there is a need for techniques which provide abstraction mechanisms for model transformation modification and adjustment problems. However, when considering models and model transformations (which are often also treated as models in MDSD) as assets, reuse and evolution is a very important issue.

**Reuse of Models** Reuse of models addresses challenges that occur in model integration and migration scenarios. The trend towards more and more people using DSLs and creating their own domain-specific models is a main driver for model integration projects. In cross-organisational development projects, different enterprise model formalisms have to be mapped on each other to achieve a shared understanding of the enterprise domain. One may also use different DSLs to model the various aspects of a system. In order to achieve a complete solution such domain-specific models also have to be integrated. This is commonly done by mapping the various metamodels (abstract syntax representation of the DSLs) on each other via model trasformations. Also time

and again new versions of metamodels, e.g. the metamodels for UML 1.x and UML 2.x, are released. This evolution of (meta)models can be supported by migrating models conforming to the old metamodel to models that conform to the new metamodel. Whenever new versions replace the old ones, new model transformations have to be developed.

**Reuse of Model Transformations** In MDSD knowledge about IT solutions and platforms is encoded in model transformations. Organisations naturally want to treat this knowledge as assets and therefore reuse model transformations. In a common MDSD scenario one needs to model at different levels of abstraction to develop a certain kind of application. The models of the higher abstraction layers need to be transformed into models of the lower abstraction layers by transformation.

### Example

Figure 6.13 illustrates an abstract scenario. Two DSLs are used for modelling, and a set of implementation patterns *S* is used to transform models from higher to lower abstraction level. One organisation chooses the metamodels *MMa* and *MMc* representing the two DSLs used for modelling at the two levels of abstraction. Further it discovers a model transformation *Mt* in a library, that implements the set of implementation patterns *S*. Unfortunately, this model transformation *Mt:Ma→Mb* transforms models *Ma* conforming to metamodel *MMa* into models *Mb* conforming to metamodel *MMb*. In order to reuse the transformation patterns encoded in *Mt* and to obtain models *Mc*, that correspond to the metamodel *MMc*, mechanisms that help to reuse the model transformation *Mt* are needed.

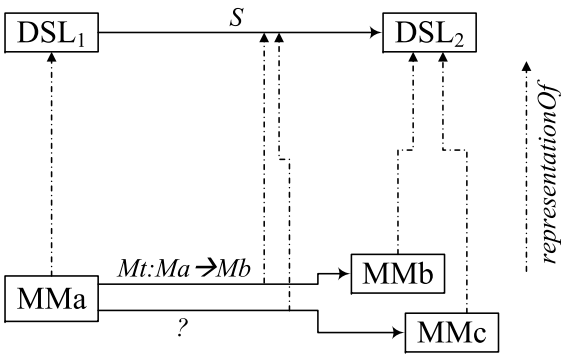


Figure 6.13: Reuse scenario

For more comprehensive descriptions throughout this section we use a concrete example. Figure 6.14 and Figure 6.15 depict the three metamodels *MMa*, *MMb*, and *MMc* of the above described application scenario. The structure of *MMa* is similar to *MMb*. Real world examples in which the higher-order model transformation language is used can be found in Chapter 6.5.

All three metamodels are models that represent libraries. In each *Library* one can find *Novels* and *Journals*. Both are *Books* and have a *name*. A journal has also a *topic*. Each book has *Authors*. In metamodel *MMa* and *MMb* the profession of authors is represented through the *profession* attribute. In metamodel *MMc* different professions are realized through an inheritance hierarchy. For each profession exists a class (*StoryWriter* and *Scientist*), which inherits from the abstract class *Author*.

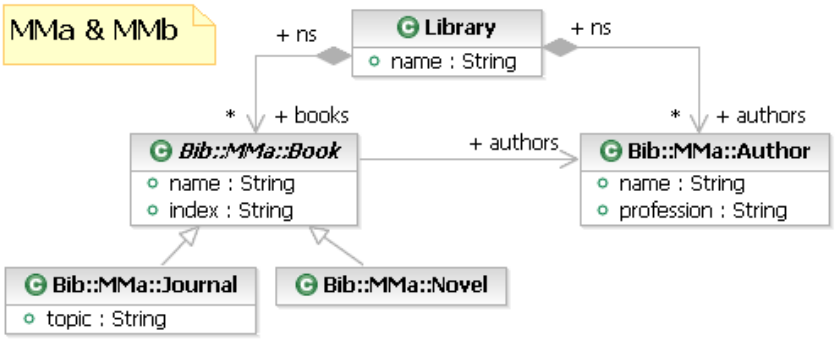


Figure 6.14: Metamodels *MMa* and *MMb*

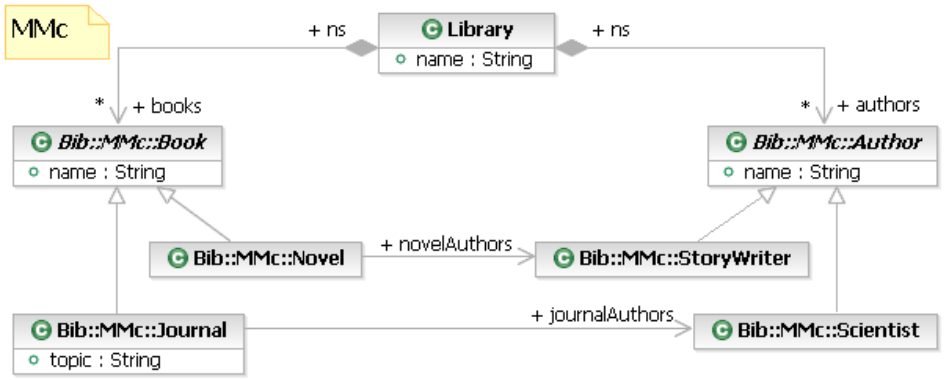


Figure 6.15: Metamodel *MMc*

There also exists a model transformation *Mt:Ma→Mb* from *MMa* to *MMb*. Listing 6.12 depicts this model transformation in QVT relational syntax [233]. QVT is an OMG's standard for model transformations (for more details see Section 3.3.3). This transformation, which is specified in QVT relational syntax, has been executed with the ModelMorf tool [291].

Listing 6.12: QVT specification of *Mt:Ma→Mb*

```
transformation MMa2MMb(ma:MMa; mb:MMb) {

  key MMb::Library{name};
  key MMb::Book{name};
  key MMb::Novel{name};
  key MMb::Journal{name};
  key MMb::Author{name};

  top relation LibraryToLibrary {
    n: String;
    checkonly domain ma c:Library {name=n};
    enforce domain mb c_:Library {name=n};
  }

  top relation NovelToNovel {
    i, n: String;
    checkonly domain ma c:Novel {ns=l:Library{},name=n,index=i} ;
    enforce domain mb c_:Novel {ns=l_:Library{},name=n,index=i} ;
    when {
      LibraryToLibrary(l,l_);
    }
    where {
      AuthorsToAuthors(c,c_);
    }
  }

  top relation JournalToJournal {
    i, n, t: String;
    checkonly domain ma c:Journal {ns=l:Library{},name=n,
      topic=t,index=i} ;
    enforce domain mb c_:Journal {ns=l_:Library{},name=n,
      topic=t,index=i} ;
    when {
      LibraryToLibrary(l,l_);
    }
    where {
      AuthorsToAuthors(c,c_);
    }
  }

  top relation AuthorToAuthor_StoryWriter {
    n: String;
    checkonly domain ma c:Author {ns=l:Library{}, name=n,
      profession='StoryWriter'} ;
    enforce domain mb c_:Author {ns=l_:Library{}, name=n,
      profession='StoryWriter'} ;
    when {
      LibraryToLibrary(l,l_);
    }
  }
```

```
52   top relation AuthorToAuthor_Scientist {
       n: String;
54     checkonly domain ma c:Author {ns=l:Library{}, name=n,
         profession='Scientist'} ;
56     enforce domain mb c_:Author {ns=l_:Library{}, name=n,
         profession='Scientist'} ;
58     when {
         LibraryToLibrary(l,l_);
60     }
   }
62
   relation AuthorsToAuthors {
64   n: String;
     checkonly domain ma c:Book {authors=a:Author{name=n}} ;
66   enforce domain mb c_:Book {authors=a_:Author{name=n}} ;
   }
68 }
```

### The Approach

For the design and specification of the semantics of our model transformation modification language, there arise requirements from the OntMT approach. These are requirements fostering the compatibility with other existing approaches, and other requirements concerning general model integration and evolution scenarios. The following list covers the most important implementation directives, what may make it easier to grasp our intentions behind the design of the approach and the semantics specification.

- The model transformation modification language shall lend itself for the integration with existing model matching, mapping, and linking approaches like model weaving [44] and Semaphore [169]. It should be possible to easily integrate mapping specifications developed with these approaches with our model transformation modification language.

- One challenge of OntMT is to foster direct reuse of model transformations. Direct reuse of a transformation means to provide a new or adjusted transformation instead of chains of transformations and adapters.

- To solve the automation challenge, OntMT uses inference results from the Ontology TS to generate transformations between models conforming to different metamodels. Due to the enormous gap between the conceptual level and the implementation of DSLs' concepts through metamodels [91, 151], it has to be possible for the computer to 'learn' an appropriate solution. The modification language has to permit modification programs that might lead to incorrect model transformations and provide sensible, computer-processible feedback by checking programs' results.

The semantics of the model transformation modification language treats model transformations as models. The fact that model transformation languages like QVT [233] are represented through metamodels and model transformation programs are models, allows higher-order transformations like transformations taking other transformations as input and producing transformations as output [37]. Other higher-order transformations

[38, 295] are for example applied to refactor a given set of transformations (e.g. a family of code generators) to reduce the amount of code duplication in these transformations.

Like one can see in Figure 6.16 a mapping specification between two metamodels is used as a model transformation modification program. Taking up the application scenario from Section 6.4.2, the modification program is a mapping specification between the metamodels *MMb* and *MMc*. A higher-order model transformation engine takes the model transformation *Mt:Ma→Mb*, executes the modification program and produces a new model transformation *Mt':Ma→Mc* as output. This technique enables direct reuse of model transformations and lends itself for the generation of totally new model transformations[8].

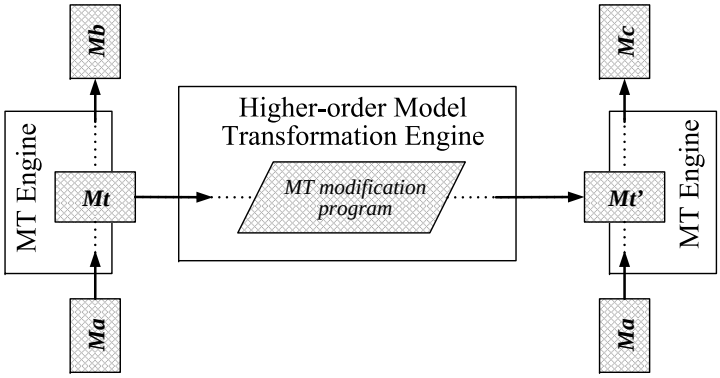


Figure 6.16: External view on higher-order model transformation

The definition of the semantics has mainly to deal with the challenges of automated generation and modification of model transformations. Due to the gap between the concepts of DSLs and metamodels implementing these DSLs, the semantics needs to provide mechanisms to allow the computer to adapt a modification program to the best possible solution. Hence the semantics is divided into a *modification semantics* and a *checking semantics* (see Figure 6.17). The modification semantics defines how the modification of model transformations, which is specified in modification programs, is executed. We use UML action semantics to specify the modification semantics.

In a second step, the checking semantics is applied to test the model transformation *Mt'* generated by the modification semantics. One set of OCL constraints is used to check whether wellformedness rules for model transformations are violated, i.e. whether the model transformation is valid and can be executed. Another set of OCL constraints is used to compare the generated model transformation with the original model transformation. It checks for example whether knowledge encoded into the original model transformation has been lost and whether the relations of the original model transformation have been modified in a correct manner. The checking semantics produces a set of checking results containing information about the kind of the detected problem, which part of the modification program caused the problem, and at which point in the model transformation the problems occurred.

[8]For the generation of mappings between metamodels, a bootstrapping of model transformations is conducted before the modification is applied. For more details see Section 6.4.1.

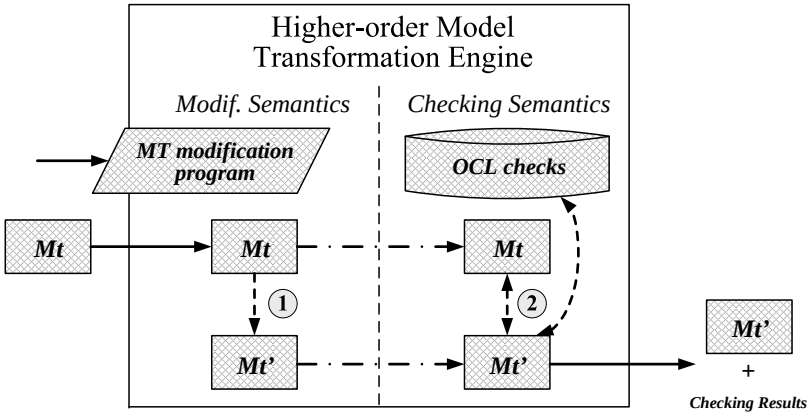


Figure 6.17: Internal view on higher-order model transformation

### Model Transformation Modification Language

The requirements presented at the beginning of Section 6.4.2 do not only influence the choice to use higher-order transformations, but also the design and implementation of the model transformation modification language. This section presents the syntax and the semantics of our model transformation modification language in detail.

**Syntax** Figure 6.18 depicts the metamodel, which represents the abstract syntax of the model transformation modification language. Though relatively simple, this syntax is sufficient to express non-trivial mapping between (meta)models. The abstract syntax is also based on the abstract syntax of other languages for model mapping and linking. A *Mapping* represents a modification program and can have a *name*. A mapping references (+*source_model*, +*target_model*) the metamodels between which the mapping is specified. A mapping consists of *Links* which can be nested (+*nested*). Links either map classes (*emof::Class*) or properties (*emof::Property*). A link has one mandatory source element (+*source*) and an arbitrary number of target elements (+*target*). Property Template Items (PTIs) (*qvttemplate::PropertyTemplateItem*) are used to specify constraints (+*constraint*) on the values of the properties of classes. The constraint is on the property that is an instance of the referred property (+*referredProperty*) and the constraining expression is given by the value expression (+*value*). A PTI has a valid match when the value of the referred property (*emof::Property*) matches the value specified by the value expression (*essentialocl::OclExpression*). For more details see the QVT specification [233, p.31f].

The grammar $G := < N, T, P, \Sigma >$ defines a concrete syntax of the model transformation modification language:[9]

$N$ := {map, cLink, pLink, class, property, name, letter}
$T$ := {**a**, **b**, ..., **z**, **A**, **B**, ..., **Z**, **:**, **_**, **(**, **)**, **,**, **;**, **{** , **}** , →}
$\Sigma$ := map
$P$ :=
<map> ::= <name>**(**<name>**;**<name>**) {** [ <cLink> | <pLink> ]*__}__
<cLink> ::= <name>**(**<class> [ **{**<constraint>**}** ] → [ <class> ] **)**

[9]The definition of the nonterminal <*propertyTemplate*> can be found in the QVT specification [233, p.39].

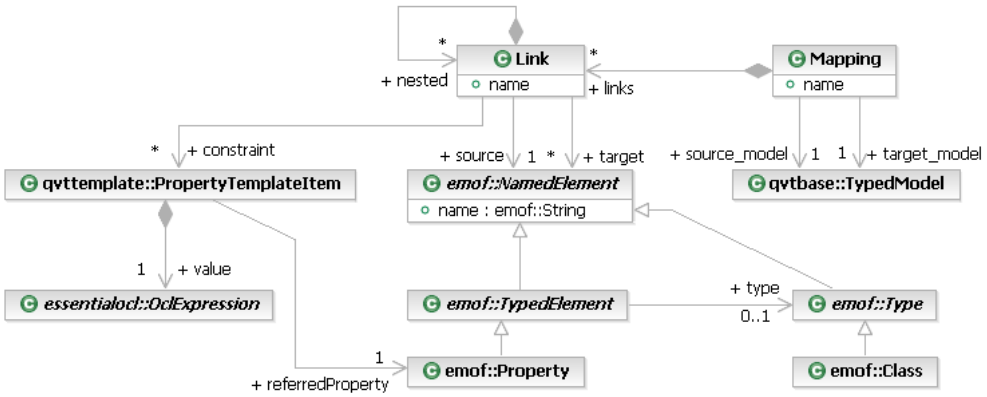


Figure 6.18: Abstract syntax of the higher-level model transformation language

**{**<pLink>* <cLink>* **};**
<pLink> ::= <name>**(**<property> → [ <property> ] **);**
<class> ::= <name> [ **,** <class> ]
<property> ::= <name> [ **,** <property> ]
<constraint> ::= <*propertyTemplate*> [ **,** <constraint> ]
<name> ::= <letter> $^{+}$
<letter> ::= **a** | **b** | ... | **z** | **A** | **B** | ... | **Z** | **:** | **_**

**Example** Applying this concrete syntax to the example introduced above, we can now specify the dependencies between the metamodels *MMb* and *MMc* (cp. Listing 6.14 and Listing 6.15). The fact, that the concept library is represented in both metamodels as a class *Library*, is expressed by the *CL_lib*. The properties *name*, *books*, and *authors* are mapped via property links.

*CL_book* maps the class *Book*, while *CL_nov* and *CL_jour* map *Novel* and *Journal*. The properties *ns* and *name* of class *Book* are mapped in so-called global property links. These are property links that can be used for all class mappings where the mapped class owns or inherits the mapped property. In the example these are the property links *PL_name* and *PL_ns*.

The mapping of the *authors* property depends on the class that is mapped. For books the mapping of *authors* on *novelAuthors* and *journalAuthors* represents a choice; the source property is mapped on one of the properties in the target list. For novels and journals the *authors* property is mapped on a distinct target property, *novelAuthors* and *journalAuthors* respectively.

For the mapping of the author concept we specify three mapping rules. First, *author* is mapped on the classes *StoryWriter* and *Scientist*. However, we also want to specify mappings for the concepts story writer and scientist. Both concepts are represented through the same class (*Author*) in *MMb*, where only the values of the property instances differ. Hence, we specify additional mappings for the class *Author* and constrain these mappings with PTIs. For example, *class_link6* specifies the mapping of the class *Author* with the *profession 'StoryWriter'* from *MMb* on the class *StoryWriter* from *MMc*.

Finally, the property *profession* has no direct representation in the target metamodel

*MMc*. This is expressed in *PL_auth_prof*, which is a removal mapping.

The model transformation that is generated by applying the modification program (Listing 6.13) to the model transformation of the library example (Listing 6.12) can be found in Appendix C.1.

Listing 6.13: Example modification program

```
1  modification_mapping (MMb ; MMc) {
2    CL_lib( Libary → Libary ) {
3      PL_lib_name ( Libary:name → Libary:name );
4      PL_lib_book ( Libary:books → Libary:books );
5      PL_lib_auth ( Libary:authors → Libary:authors );
6    } ;
     CL_book ( Book → Book ) {
8      PL_book_auth ( Book:authors →
9        Novel:novelAuthors,Journal:journalAuthors );
10   } ;
     PL_book_name ( Book:name → Book:name );
12   PL_book_ns ( Book:ns → Book:ns );
13   CL_nov ( Novel → Novel ) {
14     PL_nov_auth ( Book:authors → Novel:novelAuthors );
15   } ;
16   CL_jour ( Journal → Journal ) {
17     PL_jour_auth ( Book:authors → Journal:journalAuthors );
18     PL_jour_top ( Journal:topic → Journal:topic );
19   } ;
20   CL_auth_1 ( Author → StoryWriter,Scientist ) { };
21     PL_auth_name ( Author:name → Author:name  );
22     PL_auth_prof ( Author:profession → );
23     CL_auth_2 ( Author {profession='StoryWriter'} →
24       StoryWriter ) { };
       CL_auth_3 ( Author {profession='Scientist'} →
26       Scientist ) { };
     };
28 }
```

**Semantics** A simplified picture that helps to work with the model transformation modification language is to imagine the modification program as a substitution. The elements of the modification program's source metamodel are substituted by the elements of the target metamodel. The detailed implementation realizing the semantics is encapsulated in a separate component. In the following we present the semantics of the model transformation modification language and illustrate it via the library example.

**Interpretation of the Modification Program** The semantics distinguishes between three kinds of substitutions:

*One-to-one substitution.* The semantics of an one-to-one substitution is applied when a *Link* has exact one target. This is the case when a link does reference (+*target*) exactly one *NamedElement*. In the example this is the case for the *CL_lib*, where *MMb:Library* is substituted with *MMc:Library*.

*One-to-many substitution.* The semantics of an one-to-many substitution is applied when a *Link* has multiple targets. This is the case when a link does reference multiple *NamedElements*. In the example this is the case for the *CL_book*, where *MMb:Book* is substituted with *MMc:Novel* and *MMc:Journal*.

*Removal substitution.* The semantics of a removal substitution is applied when a *Link* has no targets. This is the case when a link does not reference any *NamedElement* as target. In the example this is the case for the *PL_auth_prof*, where *MMb:Author:profession* cannot be substituted with an element from metamodel *MMc*.

Beneath these substitutions the model transformation modification language offers two further techniques, *nesting* and *constraining*, to specify modification programs. It is possible and often sensible to use both techniques in combination.

*Constraining.* Constraining is used when different concepts are represented in a metamodel through the same class and can only be distinguished by the values of the class instances' properties. Constraints can be specified on the values of the properties. A constraint for class *C* of the modification program is satisfied, when a template for the class *C* of the QVT model transformation also specificies this constraint through a PTI. A constrained class link *CL* can be applied as a substitution for a template *T* in the model transformation, when *T* satisfies all constraints of *CL*. In the example, the template that maps *Author* in the *AuthorToAuthor_Scientist* relation in Listing 6.12 satisfies *CL_auth_1* and *CL_auth_3* of the modification program in Listing 6.13. The semantics always chooses the class link, that satisfies the most constraints, i.e. in the example *CL_auth_3*.[10]

*Nesting.* Nesting determines the scope of propery links. A property link is only visible in class links that are siblings or children of the siblings of the property link. For example the mappings of the *Book:authors* property are only visible within the respective class link (*PL_book_auth* within *CL_book*, etc.). When there exist multiple property links for a certain property that are visible within a class link, then the most local one is chosen. However, global property links like *PL_book_name* in Listing 6.13 are an elegant way to specify modification programs with only few property links.

As described above, the semantics of the model transformation modification language consists of a modification semantics and a checking semantics. The modification semantics is used to execute the modification and produces adjusted model transformations as output. The checking semantics tests the modified model transformation and produces computer processible checking results. The interrelation between modification and checking semantics allows the computers (and humans) to develop and adapt appropriate model transformation modification programs in an incremental procedure.

**Modification Semantics** The modification semantics describes how model transformation modification programs are executed and new model transformations are generated. To describe the modification execution mechanisms we use the UML action semantics.

Action semantics [198] is a framework for the formal specification of programming languages that directly reflects ordinary computational concepts and is easy to read and understand. With the UML action semantics the OMG has integrated a specification of actions in UML. UML action semantics lends itself for manipulation of UML elements, that includes to transform model transformations which themselves are models.

[10] When there exist multiple class links with max#(satisfied constraints), the first class link is chosen.

The UML action semantics is not a concrete action language and does not enforce any notation. Action languages are free to provide more sophisticated constructs, as long as these constructs can be translated into the basic concepts defined by the Action Semantics. The details of concrete syntaxes are left to so-called surface languages, which have to comply with the Action Semantics. All such languages have the Action Semantics as their abstract syntax [200]. For our specifications we use the concrete syntax of Action Specification Language (ASL) [321] that is also used for specifying processing behaviour in the context of Executable UML.

**Modification Semantics – Data** The specification of the modification semantics uses three data structures: the QVT relational metamodel (see Section 3.3.3), the abstract syntax of model transformation modification language (see Figure 6.18), and an internal data structure for processing the substitutions on the model transformation (see Figure 6.19).

The internal data structure contains *SubstitutionItems* that represent one-to-one, one-to-many, and removal substitutions. Substitution items are computing instructions that represent concrete substitutions (like substitution for *TypedModel*, *Key*, *OTE*, *PTI*, *Variable* and *VariableExp*) on QVT relational model transformations. Multiple substitution items are necessary to realize substitutions represented by the links of modification programs. Substitution items are stored in an ordered list (+*siList*) that is contained in a substitution proposal. The source element (+*srcEl*) is the element of a metamodel, that shall be substituted through one or more other metamodel elements (i.e. +*trgEl* and +*trgEls*). The containing element (+*contEl*) is the element of the QVT model for which the substitution is applied (e.g. an OTE) and which references the metamodel (source and target) elements.

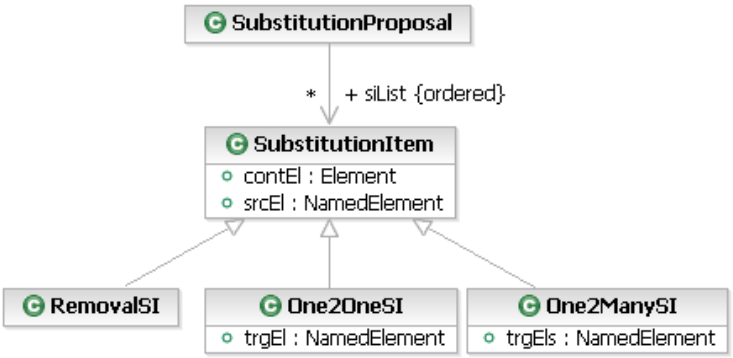


Figure 6.19: Modification semantics internal data

**Modification Semantics – Actions** The modification semantics has two main processing steps. First, a list of substitution items (+*siList*) is generated on the basis of the input model transformation and links in the modification program. Second, the substitution items stored in this list are processed.

*Computing substitution items.* To generate the list of substitution items, the input model transformation is traversed via its composite aggregation and its implicit template hierarchy starting with the transformation element. Substitution items are generated

for *TypedModel*, *Key*, *OTE*, *PTI*, *Variable* and *VariableExp*. Listing 6.14 describes the processing steps that are necessary to generate substitution proposals for an OTE and one-to-one substitutions in the modification program. First, the respective link in the modification program is obtained, whose source class matches the class that is referenced by the OTE instance ①. If the link refers exactly to one target, one-to-one substitutions are generated for the OTE. Otherwise removal or one-to-many substitution items are generated ②. In the listing one-to-one substitution items are generated for the class referenced by the OTE (+*referredClass*) ③, the type referenced by the OTE (+*type*) ④, the variable the OTE binds to (+*bindsTo*) ⑥, and all variable expressions that reference this variable (+*referredVariable*) ⑤.

For the OTE that refers to *MMb:Novel* in the relation *NovelToNovel* in Listing 6.12, the modification semantics generates five substitution items: one for the referred class of the OTE, one for the type of the OTE, one for the type of the bound variable, and two for the type of the respective variable expressions *c_*.

Listing 6.14: Compute substitution items for OTE

```
1  define instance function ObjectTemplateExp::computeSIs
2  input classLink:Link, subProp:SubstitutionProposal
   output ---
4
   switch countof{classLink→target} # (2)
6    case 0
     # compute substitution items for removal
8
     case 1
10   # (3) substitute class of OTE
     sItemClass = create One2OneSI
12     with srcEl = this→referredClass
13       & trgEl = classLink→target
14       & contEl = this
     append sItemClass to subProp.siList;
16
     # (4) substitute type of OTE
18   sItemType = create One2OneSI
       with srcEl = this→type
20       & trgEl = classLink→target
21       & contEl = this
22   append sItemType to subProp.siList;

24   boundVar = this-> bindsTo
     varExps = find VariableExp
26     where referredVariable = boundVar
     if boundVar != UNDEFINED then
28     for varExp in varExps do
     # (5) substitute type of variable expressions
30       sItemVE = create One2OneSI
           with srcEl = varExp→type
32           & trgEl = classLink→target
33           & contEl = varExp;
34       append sItemVE to subProp.siList;
       endfor
36   # (6) substitute type of variable
       sItemVar = create One2OneSI
38       with srcEl = boundVar→type
39         & trgEl = classLink→target
40         & contEl = boundVar;
```

```
    append sItemVar to subProp.siList;
42
default # ≧2
44    # compute substitution items for one-to-many
  endif
46 endswitch

48 enddefine
```

*Processing the substitution items.* The substitution items, which are stored in the list (+*siList*) of the substitution proposal, are processed one after another, starting with the first substitution item and proceeding with its successor. In the following we describe how the three different substitution item types *One2OneSI*, *One2ManySI*, and *RemovalSI* are processed.

Listing 6.15 depicts the computing instructions that are performed for the *One2OneSI* substitution item. First, the association between the containing element *contEl* and the source element *srcEl* is deleted. Then an association of the same kind is created between the containing element and the target element *trgEl*.

Listing 6.15: Execution of one-to-one substitution

```
1 define instance function One2OneSI::execute
2 input ---
  output ---
4 unlink self→contEl <Association> self→srcEl
5 link self→contEl <Association> self→trgEl
6 enddefine
```

In the code extract of Listing 6.16 one can find, that multiple *One2OneSI* substitution items have been applied to the *MMa2MMb* model transformation. First the metamodel *MMb* has been replaced with *MMc* for the respective *TypedModel*. The class *MMb:Library* has been substituted with *MMc:Library* in a *Key* and an OTE of the *AuthorToAuthor_Scientist* rule. The class *MMb:Author* has been replaced with *MMc:Scientist* in another OTE of this rule (cp. Figure 6.20).

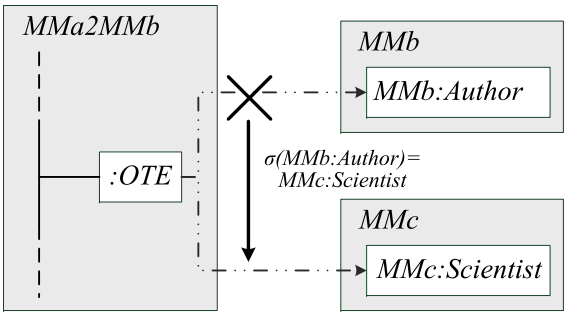


Figure 6.20: One-to-one substitution example

Further, it was necessary to replace some properties: *MMb:Library:name* with *MMc:Library:name*, *MMb:Author:name* with *MMc:Author:name*, *MMb:Author:ns* with *MMc:Author:ns*, etc. What one cannot directly see in the code is the substitution of types. Types had to be substituted for *OTEs*, *Variables* and *VariableExps* (e.g. *c_* or *l_*).

Listing 6.16: Example one-to-one substitution

```
1  transformation MMa2MMb(ma:MMa; mc:MMc) {
2    key MMc::Library{name};
     ...
4    top relation AuthorToAuthor_Scientist {
       n: String;
6      checkonly domain ma c:Author {...} ;
       enforce domain mc c_:Scientist {
8        ns=l_:Library{},
         name=n,
10       ...
       } ;
12     ...
     }
14 }
```

The execution of *One2ManySI* substitution items replicates relations.[11] Like one can see in Listing 6.17, the relation *AuthorsToAuthors* has been duplicated to *AuthorsToAuthors_1* and *AuthorsToAuthors_2*. This means, that the execution of a *One2ManySI* substitution item copies whole relations of a model transformation, the elements that are contained by those relations (like *RelationDomain*, *OTE*, *Variable*, etc.), and adds them to the model transformation. Further it replaces the *One2ManySI* substitution item with *One2OneSI* substitution items in the *+siList* of the substitution proposal for the relation and each copy of the relation.

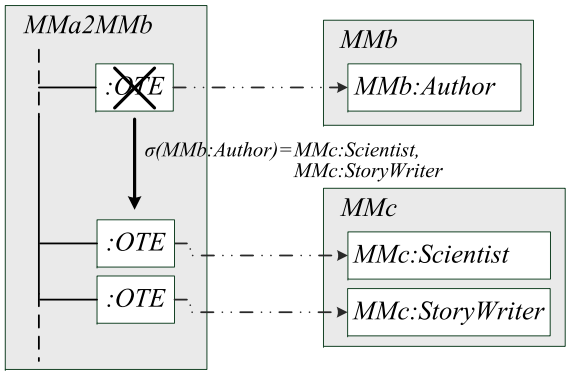


Figure 6.21: One-to-many substitution example

In the library example the *One2ManySI* substitution item has two target elements *MMc:Scientist* and *MMc:StoryWriter*. Thus, the *One2ManySI* substitution item is replaced by two *One2OneSI* substitution items, one that has the target element *MMc:Scientist* and one that has the target element *MMc:StoryWriter* (cp. Figure 6.21). Finally, all substitution items of a copied relation, which have not been processed before the current *One2ManySI* substitution item, are also replicated. In Listing 6.17 this is for example the *One2OneSI* substitution item for *MMc:Author:name*. The copies of a substitution item are directly inserted behind the original one in the substitution item list. The execution of the substitution items in the *+siList* proceeds with the first substitution item that has not yet been executed.

[11] Rules are only replicated for *One2ManySI* substitution items that references classes in the metamodel. *One2ManySI* substitution items that reference properties behave like *One2OneSI* substitution items by choosing one target element (property) that fits the substitution's context.

Listing 6.17: Example one-to-many substitution

```
relation AuthorsToAuthors_1 {
  n: String;
  checkonly domain ma c:Book {...} ;
  enforce domain mc c_:Book {
    journalAuthors=a_:Scientist{name=n}
  } ;
}
relation AuthorsToAuthors_2 {
  n: String;
  checkonly domain ma c:Book {...} ;
  enforce domain mc c_:Book {
    novelAuthors=a_:StoryWriter{name=n}
  } ;
}
```

The third substitution item *RemovalSI* is used to remove *Keys*, *OTEs*, *PTIs*, or even *Relations*. Having a look at the library example, several removal substitution items have to be generated to remove the property *MMb:Author:profession* from the relation *AuthorToAuthor_Scientist*. In Listing 6.18 one substitution item removed the *OclExpression 'Scientist'* and one the PTI that references the property *MMb:Author:profession* (cp. also Figure 6.22).

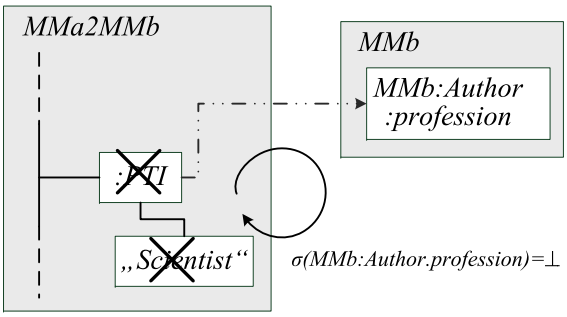


Figure 6.22: Removal substitution example

In the case that there occur nested OTEs or PTIs, the nested OTEs and PTIs also have to be removed. For example in the relation *AuthorToAuthor_Scientist* the OTE *Library{}* is nested within the PTI *ns*, since a variable with the type of a *Library* is assigned to a property (*ns=l_:Library{}*). If the PTI *ns* had to be removed, also the OTE *Library{}* had to be removed. Finally, if an OTE that binds the root variable of a relation domain has to be removed (like *Author* in the relation *AuthorToAuthor_Scientist*), the whole relation has to be removed from the model transformation.

Listing 6.18: Example removal substitution

```
top relation AuthorToAuthor_Scientist {
  n: String;
  checkonly domain ma c:... ;
  enforce domain mc c_:Scientist {
    ns=l_:Library{}, name=n
    # removed Property MMb:Author:profession
    # removed OclExpression 'Scientist'
  } ;
}
```

**Checking Semantics** The checking semantics represents the second part of the model transformation modification language's semantics. It tests the generated model transformations for so-called problems, which can occur by applying the modification semantics.

- One set of problems affects the consistency of model transformation programs with respect to the model transformation language, i.e. the generated model transformations are not valid and cannot be executed.

- Another kind of problems is caused, when knowledge encoded into the original model transformation is not preserved or lost. This is the case when modifications and substitutions are applied to relations where they (normally) do not make sense.

In general, problems are detected via OCL [229] constraints. Only for a few problems, where additional information about the execution of the modification is needed, we extend this mechanism with information from the substitution proposal; this is the case for the second kind of problems where the generated model transformation has to be compared with the original model transformation.

In the following we describe the problems that are detected by the checking semantics and illustrate selected problems on the basis of the application scenario's example and the OCL constraints that are used for checking.

**Checking Semantics – Checking validity** This set of checks tests, whether the generated model transformation conforms to the QVT relational metamodel and is consistent, i.e. the generated model transformation can be executed.

- *Problem 1: property is not part of class.* This problem occurs, when a generated model transformation would require a property to be part of a class, what is not the case in the respective metamodel. In the library example this is the case in the relations *AuthorsToAuthors_1* and *AuthorsToAuthors_2* of the generated model transformation (see Listing 6.17). The pattern *mb c_:Book {journalAuthors=...}* requires the property *MMc:Journal:journalAuthors* to be part of the class *MMc:Book* in the metamodel that is associated with the typed model *mc*. This is not the case in the library example (cp. Figure 6.23).

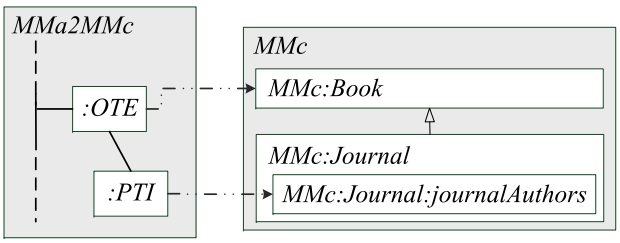


Figure 6.23: Example for *property is not part of class* problem

The OCL invariant *property_part_of_class* in Listing 6.19 is used to check this fact. The constraint is satisfied, if the respective OTE's referred class contains the property or if the OTE does not reference a class, i.e. the OTE could not be substituted (see *substitution of class failed*).

Listing 6.19: OCL constraint: property part of class

```
package qvttemplate
  context PropertyTemplateItem
    inv property_part_of_class:
      self.objContainer.referredClass.
                    hasProperty(self.referredProperty)
      or
      self.objContainer.referredClass→isEmpty()
endpackage

package emof
  context Class
    def: hasProperty(property: Property): Boolean =
      self.ownedAttribute→exists(p: Property | p = property)
      or
      self.superClass→exists(c: Class | c.hasProperty(property))
endpackage
```

- *Problem 2: substitution of property failed.* This problem occurs, when the model transformation modification program did not specify a substitution for a property that is referred by a PTI. Since in such cases the modification semantics sets the *referredProperty* of the PTI to the value *UNDEFINED*, the OCL constraint checks whether the referred property of a PTI is not empty (see Listing 6.20). In the library example this problem is detected for the property *MMb:Book:index* in the relations *NovelToNovel* and *JournalToJournal*.

Listing 6.20: OCL constraint: substitution of property failed

```
package qvttemplate
  context PropertyTemplateItem
    inv referredProperty_must_be_set:
      self.referredProperty→notEmpty()
endpackage
```

- *Problem 3: substitution of class failed.* This problem occurs, when the model transformation modification program did not specify a substitution for a class that is referred by an OTE. Since in such cases the modification semantics sets the *referredClass* of the OTE to the value *UNDEFINED*, the OCL constraint checks whether the referred class of an OTE is not empty (see Listing 6.21).

Listing 6.21: OCL constraint: substitution of class failed

```
package qvttemplate
  context ObjectTemplateExp
    inv referredClass_must_be_set:
      referredClass→notEmpty()
endpackage
```

- *Problem 4: value of PTI must have a type compatible with the type of the referred property.* The OCL constraint in Listing 6.22 checks, whether the type of a PTI's value does conform to the type of the PTI's referred property.

Listing 6.22: OCL constraint: value of PTI must have a type compatible with the type of the referred property

```
package qvttemplate
  context PropertyTemplateItem
    inv type_of_value_must_match_type_of_referredProperty:
      value.type→isEmpty()
      or
      referredProperty.isOfType(value.type)
endpackage

package emof
  context TypedElement
    def: isOfType(type: Type): Boolean =
      self.type.isAssignableFrom(type)

  context Type
    def: isAssignableFrom(type: Type): Boolean =
      type→isEmpty()
      or
      self = type
      or (
        oclIsKindOf(Class)
        and
        oclAsType(Class).superClass→exists(
          superClass: Class | superClass.isAssignableFrom(type)
          )
      )
endpackage
```

- *Problem 5: OTE must refer to a class that belongs to the metamodel of the domain.* Listing 6.23 checks whether classes referenced by OTEs belong to the metamodel of the respective domain.

Listing 6.23: OCL constraint: OTE must refer to a class that belongs to the metamodel of the domain

```
package qvtrelation
  context RelationDomain
    inv class_must_be_contained_in_metamodel:
      pattern.templateExpression→forAll(
        expression: qvttemplate::TemplateExpression |
        expression.oclIsKindOf(qvttemplate::ObjectTemplateExp)
        and (
          let referredClass: emof::Class = expression.oclAsType(
            qvttemplate::ObjectTemplateExp).referredClass
          in
            referredClass→isEmpty()
            or
            typedModel.usedPackage→exists(
                metamodel: emof::Package |
                metamodel.ownedType→exists(type: emof::Type |
                    type = referredClass
                    )
                )
        )
      )
endpackage
```

- *Further problems.* The checking semantics comprises further OCL constraints to check the validity of generated model transformations like whether the substituted *Keys* belong to the metamodel or whether the metamodel could be substituted.

**Checking Semantics – Comparing model transformations** This kind of checks is used to compare the generated model transformation with the original one. The checks determine whether knowledge encoded into the original model transformation has been lost and whether the relations of the original model transformation have been modified in a correct manner. The occurrence of those problems does not necessarily depend on the (in)validity of model transformations, but on information from the original model transformation and the transformation execution. Thus we extended the OCL constraints that check the validity of the generated model transformation with checks working on the original model transformation and the substitution proposal.

- *Problem 1: pattern removal.* A loss of information occurs when a class or a property could not be substituted and therefore the respective patterns had to be removed from the model transformation. In our example this problem is caused by the *property_link5b(Author:profession → )* in the modification program. This link triggers the removal of the PTI that matches the property *MMb:Author:profession* in the relations *AuthorToAuthor_StoryWriter* and *AuthorToAuthor_Scientist*.

- *Problem 2: granularity of model transformation does not fit modification program.* This problem occurs, when a one-to-many substitution is applied to a top level relation. If we had in the library example (see Listing 6.12) only one top relation (*AuthorToAuthor*) for mapping *Author* instead of two relations (*AuthorToAuthor_StoryWriter* and *AuthorToAuthor_Scientist*), the modification semantics would try to apply the class link *(Author → StoryWriter,Scientist)*, i.e. a one-to-many substitution, to the *top relation AuthorToAuthor*. This would result in a new model transformation, which would produce for each instance of *Author* not one element but two elements (one that is instance of *StoryWriter* and one that is instance of *Scientist*) in the target model. The granularity of the *AuthorToAuthor* relation in the initial model transformation specification is not fine enough. However, an exclusive-OR semantics where either an instance of *StoryWriter* or an instance of *Scientist* is produced is more obvious. Therefore the initial model transformations is required to have relations, where the patterns matching the source model *Ma* make the essential distinction.

### Conclusions

In our approach of OntMT we address the automation challenge to achieve more efficient reuse and evolution of models and model transformations. To put the respective mechanisms at a higher level of abstraction we developed a language to specify modifications of model transformations. We use this language in the implementation of the OntMT approach as an abstraction layer to decouple modelling issues from ontology reasoning issues. The (abstract) syntax we use for this language is based on the syntax of languages used for model mapping and linking and thus is similar to the syntax of such languages. This enables easier integration of these approaches and may also foster synergy effects. However, the semantics totally differs from existing approaches and provides a new and novel way to deal with the above described challenges.

### 6.4.3 Ontology Representation and Reasoning

In a semantic-enabled modelling and development suite like introduced in Section 6.3.1 models and metamodels are automatically represented in both the MDA TS and the Ontology TS. Approaches like described in [39, 69, 234] can be used realize mappings between (meta)models and ontologies. The EODMvproject [77] already provides a first implementation of the mapping between UML and OWL that is described in [230].

Having described extensively the operations that are performed in the MDA TS for OntMT in the previous sections, we now deal with the Ontology TS in this section. We describe how the metamodels can be represented and annotated with OWL. Since inference is an integral part of the OntMT approach, we introduce a set of relationships and reasoning rules that are used to infer relationships between metamodel elements for OntMT. These techniques are used to implement the inference component of the Sem-MT-Tool. For a better illustration we use the metamodels, the RO and the semantic mappings of the process case study in Section 6.5.1 as a running example. Further details and a description of the implementation can be found in [95].

#### Ontology Representation

For representing our metamodels, the RO, and semantic mappings in OWL, we decided to assume the existence of a global domain. We decided to not apply approaches that support local domains like [49, 55, 56, 109, 161] in OntMT, due to the poor reasoning support that is currently available for these approaches. Thus, semantic relations are interpreted in the same way as axioms in the ontologies [51]. All data that is relevant for the reasoning is stored in one ontology. In the following we introduce, how metamodels, a RO, and semantic mappings are stored in an OWL ontology. Since the way concepts are represented in an OWL ontology has also implications on the reasoning and vice versa, we will also discuss the reasoning possibilities of different representation mechanisms.

**Representing the Reference Ontology** Representing the RO in the (global) common ontology is relatively straight forward, since the RO has often already been built in the OWL format. Listing 6.24 depicts an excerpt of the RO from Figure 6.27 represented in OWL.

Listing 6.24: OWL representation of RO

```
<owl:Class rdf:about="urn:ua:pvs/CompositeProcess"/>
<owl:Class rdf:about="urn:ua:pvs/ProcessComponent">
  <owl:...  />
</owl:Class>
...
<owl:ObjectProperty rdf:about="urn:ua:pvs/composedOf_1">
    <owl:inverseOf rdf:resource="urn:ua:pvs/composedOf_2"/>
</owl:ObjectProperty>
...
<owl:DatatypeProperty rdf:about="urn:ua:pvs/ProcessComponent_name">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="urn:ua:pvs/ProcessComponent"/>
</owl:DatatypeProperty>
...
```

**Representing Metamodels** Metamodels are represented in MOF and UML respectively. To convert them to OWL we can make use of the UML to OWL mapping described in the ODM [230, p.201ff] standard of the OMG. Table 6.2 gives an overview on which OWL element is the equivalent to which UML element. Both OWL and UML are based on classes. A class in OWL is associated with a set of individuals. A class in UML is a more general construct, but one of its uses is as a set of instances. Thus, UML classes are mapped on *owl:Class*. The relationships among classes represented in OWL by *owl:ObjectProperty* and *owl:DatatypeProperty* come from two different sources in MOF models.

- One source is the association *ownedAttribute* between class and property. Instances of classes' *ownedAttribute* properties translate to *owl:ObjectProperties*, if the types of the UML properties are classes. MOF *ownedAttribute* associations are distinct, even if *ownedAttributes* have the same name associated with different classes. The OWL property names therefore must be unique. One way to ensure this is to use a combination of the class name and the owned property name. One further problem regarding attributes is the distinct definition of the subclass relationship in MOF and OWL (OWL *rdfs:subClassOf*, MOF generalization). Whereas it is possible to directly access an inherited attribute in MOF via stating *"Class.attribute"*, this is not possible in OWL, especially in a subclass hierarchy.

- The second source of OWL properties in a MOF model are associations. A binary MOF association translates directly to an *owl:ObjectProperty*.

The mapping, which is depicted in Table 6.2, can be used to transform metamodels to so-called application ontologies in OWL format. Representing the metamodels in OWL is the key prerequisite to add them to the (global) common ontology and to apply inference.

| UML | OWL |
|---|---|
| Class | owl:Class |
| Property | owl:DatatypeProperty |
| Association | owl:ObjectProperty |

Table 6.2: UML to OWL mapping

Listing 6.25 depicts an excerpt of the running example's process metamodel (see Figure 6.25) represented in OWL.

Listing 6.25: OWL representation of metamodels

```
<owl:Class rdf:about="urn:ua:pvs/P_Step"/>
<owl:Class rdf:about="urn:ua:pvs/P_Process">
   <owl:ObjectProperty rdf:resource="urn:ua:pvs/P_steps"/>
</owl:Class>
<owl:Class rdf:about="urn:ua:pvs/P_Flow"/>
...
<owl:ObjectProperty rdf:resource="urn:ua:pvs/P_source"/>
...
<owl:DatatypeProperty rdf:about="urn:ua:pvs/P_Step_name"/>
...
```

**Representing Semantic Mappings** Semantic mappings (ontological groundings) specify relationships between metamodels and ontologies. These relationships have to be added to the global ontologies in order to connect the metamodels and the RO. This is a prerequisite to allow reasoning about dependencies between metamodels via the RO.

**Relationships between Classes** Relationships between OWL classes can be represented as object properties between these classes. OWL already provides object properties like *owl:equivalentClass* or *owl:disjointWith* to specify such relationships between classes. To represent the semantic mappings we use the relationships *equivalence*, *containment right* and *containment left*. In the reasoning Section 6.4.3 we introduce more relationships (*overlap* and *possible overlap*) that can be derived between classes via reasoning. In the following, we present two ways to represent relationships, one by using the expressiveness of OWL DL and one by using the expressiveness of OWL Full. Via both ways the same reasoning results can be inferred. However, the realization of the reasoning and the execution time differs.

*Representation with OWL DL.* OWL DL requires that object properties only exist between instances of classes. Hence, in the OWL DL representation only one class is specified in the TBox which serves as the basic type for every element of either the metamodel or the RO. Therefore an instance of this class is created for every class of the running example. Listing 6.26 depicts the necessary OWL code. Both *CompositeProcess* and *P_Process* are instances of *TBOX_BaseClass*.

Listing 6.26: Representing relationships between classes in OWL DL

```
<owl:Class rdf:about="urn:ua:pvs/TBOX_BaseClass"/>
<urn:ua:pvs/:TBOX_BaseClass rdf:about="urn:ua:pvs/CompositeProcess">
  <urn:ua:pvs/:eq>
    <urn:ua:pvs/:TBOX_BaseClass rdf:about="urn:ua:pvs/P_Process"/>
  </urn:ua:pvs/:eq>
</urn:ua:pvs/:TBOX_BaseClass>
```

*Representation with OWL Full.* Listing 6.27 depicts the ontological grounding of *Process* of the process metamodel to *CompositeProcess* of the RO via an equivalence relation *urn:ua:pvs/:eq*. The equivalence relation is directly inserted between both classes. Although being TBox elements, both classes are treated as instances which leads to the ontology being in OWL Full.

Listing 6.27: Representing relationships between classes in OWL Full

```
<owl:Class rdf:about="urn:ua:pvs/CompositeProcess"/>
<owl:Class rdf:about="urn:ua:pvs/P_Process">
  <urn:ua:pvs/:eq rdf:resource="urn:ua:pvs/CompositeProcess"/>
</owl:Class>
```

**Relationships between Datatype- and Object Properties** Depending on which OWL level is needed (OWL DL or OWL Full), the expression of relationships between datatype and object properties poses far more difficult challenges than representing relationships between classes.

- The first problem is that OWL DL does only allow to specify object properties between classes. This means that it is not possible to represent relationships between datatype or object properties with object properties in OWL DL. This is only possible in OWL Full.

- The second problem relates to inheritance in MOF/UML. In the example in Figure 6.24 the class *B* inherits from class *A*. *A* has an attribute *name*. In MOF, UML, or an object-oriented language, *B.name* represents the attribute *name* of the class *B*.

  In OWL DL this fact is more complex to define. Attributes are represented as datatype properties. To specify that a datatype property belongs to a certain class, this class type is added to the set of domains of the datatype property. In the example, *name* would have *{A,C}* as its domains. *A*, *B*, and *C* can now have a *name* property. However, it is not possible to represent only those *name* properties, which belong to the class *B* via *B.name*. In OWL DL a new class, which is an intersection of *B* and the attribute *name* (see Figure 6.24), has to be built. In OWL Full one can alternatively create a unique datatype property for every element in the inheritance hierarchy. In the example the two new datatype properties would be *A_name* and *B_name*.

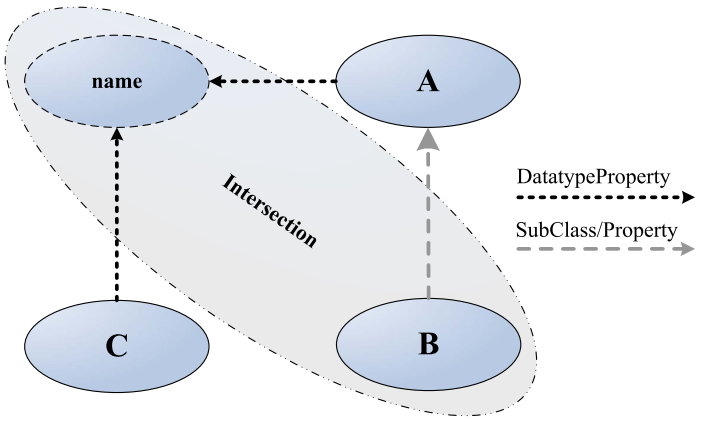


Figure 6.24: Accessing *B.name* in OWL via an Intersection

In our concrete realization, we decided to choose a simpler representation of the relationships and used OWL Full reasoning, instead of realizing a more complex relationship representation so that OWL DL reasoning can be applied. In the following we introduce the representation with OWL Full. A description of OWL DL representation can be found in [95].

*Representation with OWL Full.* OWL Full allows to specify object properties not only between classes but also between datatype and object properties, i.e. datatype and object properties are treated as instances of classes. Listing 6.28 depicts an example of how datatype properties are related via an object property. The *P_name* datatype property has an object property *urn:ua:pvs/:eq*, which relates to the datatype property *name*.

For the described inheritance problem an unique-name approach is the most consequent solution in OWL Full. This way, a unique attribute is created for every

attribute in a class hierarchy. In the example of Figure 6.24, a uniquely identifiable datatype property is created for every single class (*A* and *B*) in the hierarchy. *A* has the datatype property *A_name* (where *A_* is a prefix of the namespace of the attribute *name*) and *B* contains *B_name*. To define that both attributes represent one and the same MOF/UML attribute, both are connected via an equivalence relationship.

Listing 6.28: Relationships between datatype properties with OWL Full

```
<owl:DatatypeProperty rdf:about="urn:ua:pvs/P_name">
  <urn:ua:pvs/:eq>
    <owl:DatatypeProperty rdf:about="urn:ua:pvs/name"/>
  </urn:ua:pvs/:eq>
</owl:DatatypeProperty>
```

### Reasoning

**Types of Relationships** In the field of ontology matching and ontology mapping one can in general identify three relationships between ontology elements. According to [51, 191, 267] these are *equivalence*, *containment* and *overlap*.

- *Equivalence* ($\equiv$) means that the connected elements represent the same aspect of the real world. An element of an application ontology corresponds to an element in the reference ontology or can be precisely expressed by a composition of elements. Later, we will refer to this relationship by the relation type also as <*equal*>.

- *Containment* ($\sqsubseteq$,$\sqsupseteq$) states that the element in one ontology represents a more specific aspect of the world than the element in the other ontology. The relation can be defined in one or the other direction, depending on which concept is more specific. When an element is not sufficiently refined (i.e. it does not match the accuracy level of the ontology) we use the relation <*general*>. When an element is described at a level of refinement that does not match the level of refinement of the other ontology, we use the relation <*specific*>.

- *Overlap* ($o$) states that the connected elements represent different aspects of the world, but have an overlap in some respect. This relation is of the type <*overlap*>.

Based on these general relationship types we define five relationships, which we will use for the inference in this thesis: *equals*, *containment right*, *containment left* and *overlap*. The fifth relationship specifies a *possible overlap*. This relationship is weaker than the overlap relationship, since the relationship only holds with a given percentage $p$ between two elements. Table 6.3 provides an overview of these relationship types. For each relationship one can find a short description, a symbol that represents this relationship, and the description of the semantics.

The decision to base our relationships on relationships that are also used in ontology mapping approaches offers the possibility to exchange the reasoning rules defined in this section with other calculi. For example, parts of the reasoning could be accomplished by OWL DL or OWL Full reasoning capabilities or one could use a reasoning calculus that supports local domains like C-OWL [49], ontology integration framework [55], DL for Information Integration [56], *E*-connections [161], or distributed first-order logic [109].

| Name | Description (C relation D) | Symbol |
|---|---|---|
| *Equivalence* | C and D represent the same concept | $\equiv$ |
| | ***Semantics***: $C \equiv D \Leftrightarrow C^{\mathcal{I}} = D^{\mathcal{I}}$ | |
| *Containment right* | C is more <*specific*> than D | $\sqsubseteq$ |
| | ***Semantics***: $C \sqsubseteq D \Leftrightarrow C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ | |
| *Containment left* | C is more <*general*> than D | $\sqsupseteq$ |
| | ***Semantics***: $C \sqsupseteq D \Leftrightarrow C^{\mathcal{I}} \supseteq D^{\mathcal{I}}$ | |
| *Overlap* | The concepts of C and D overlap | $o$ |
| | ***Semantics***: $C \; o \; D \Leftrightarrow C^{\mathcal{I}} \; O \; D^{\mathcal{I}}$ | |
| *Possible Overlap* | There is a certain probability $P$ that the concepts of C and D overlap | $\Theta$ |
| | ***Semantics***: $C \; \Theta \; D \Leftrightarrow C^{\mathcal{I}} \; \theta \; D^{\mathcal{I}}$ | |

Table 6.3: Available relationship types

**Reasoning Rules** To derive the above described relationships between more (meta-model) elements, we have developed a set of reasoning rules. In this section we specify these rules that can be used to derive concrete implementations for specific rules engines like we have done for the Jena general purpose reasoner. More detailed specifications of these rules can be found in the Appendix C.2.

Listing 6.29 depicts rules that use simple OWL class definitions and relationships to compute new relationships.

Listing 6.29: Reasoning rules using simple class definitions

Equality symmetry:
$$C \equiv D \rightarrow D \equiv C$$
Overlap symmetry:
$$C \; o \; D \rightarrow D \; o \; C$$
Possible overlap symmetry:
$$C \; \Theta \; D \rightarrow D \; \Theta \; C$$
Containment right inverse:
$$C \sqsubseteq D \rightarrow D \sqsupseteq C$$
Containment left inverse:
$$C \sqsupseteq D \rightarrow D \sqsubseteq C$$
Equality transitivity:
$$C \equiv D \sqcap D \equiv E \rightarrow C \equiv E$$
Containment right transitivity:
$$C \sqsubseteq D \sqcap D \sqsubseteq E \rightarrow C \sqsubseteq E$$
Containment left transitivity:
$$C \sqsupseteq D \sqcap D \sqsupseteq E \rightarrow C \sqsupseteq E$$
Overlap (Pseudo-) transitivity:
$$C \; o \; D \sqcap D \; o \; E \rightarrow C \; \Theta \; E$$
Possible overlap transitivity:
$$C \; \Theta \; D \sqcap D \; \Theta \; E \rightarrow C \; \Theta \; E$$
Containment right 1:
$$C \sqsubseteq D \sqcap D \equiv E \rightarrow C \sqsubseteq E$$
Containment right 2:
$$C \equiv D \sqcap D \sqsubseteq E \rightarrow C \sqsubseteq E$$
Containment left 1:
$$C \sqsupseteq D \sqcap D \equiv E \rightarrow C \sqsupseteq E$$
Containment left 2:
$$C \equiv D \sqcap D \sqsupseteq E \rightarrow C \sqsupseteq E$$
Overlap 1:
$$C \; o \; D \sqcap D \equiv E \rightarrow C \; o \; E$$
Overlap 2:
$$C \equiv D \sqcap D \; o \; E \rightarrow C \; o \; E$$
Possible overlap 1:
$$C \; \Theta \; D \sqcap D \equiv E \rightarrow C \; \Theta \; E$$
Possible overlap 2:

$C \equiv D \sqcap D \,\Theta\, E \rightarrow C \,\Theta\, E$
Containment in general:
$C \sqsubseteq D \sqcap D \sqsupseteq E \rightarrow C \,\Theta\, E$
Containment of special:
$C \sqsupseteq D \sqcap D \sqsubseteq E \rightarrow C \; o \; E$
Containment right and overlap:
$C \sqsubseteq D \sqcap D \; o \; E \rightarrow C \,\Theta\, E$
Overlap and containment right:
$C \; o \; D \sqcap D \sqsubseteq E \rightarrow C \; o \; E$
Containment left and overlap:
$C \sqsupseteq D \sqcap D \; o \; E \rightarrow C \; o \; E$
Overlap and containment left:
$C \; o \; D \sqcap D \sqsupseteq E \rightarrow C \,\Theta\, E$

Classes in ontologies can be assembled of other classes. We support reasoning rules that match classes that are defined in OWL through union, intersection, and complement of classes. Listing 6.30 depicts rules that match complex classes, which are defined by the union of other classes, to compute new relationships (see also Table C.8 and Table C.9 in Appendix C.2).

Listing 6.30: Rules matching $A \equiv \bigcup_{i=1}^{n} A_i \wedge 1 \leqslant m \leqslant n$

$X$ equals $A$:
$X \equiv A \sqcap A \equiv \bigcup_{i=1}^{n} A_i \rightarrow X \sqsupseteq A_1 \sqcap \ldots \sqcap X \sqsupseteq A_n$
$X$ contained in $A$:
$X \sqsubseteq A \sqcap A \equiv \bigcup_{i=1}^{n} A_i \rightarrow X \,\Theta\, A_1 \sqcap \ldots \sqcap X \,\Theta\, A_n$
$X$ contains $A$:
$X \sqsupseteq A \sqcap A \equiv \bigcup_{i=1}^{n} A_i \rightarrow X \sqsupseteq A_1 \sqcap \ldots \sqcap X \sqsupseteq A_n$
$X$ overlaps $A$:
$X \; O \; A \sqcap A \equiv \bigcup_{i=1}^{n} A_i \rightarrow X \,\Theta\, A_1 \sqcap \ldots \sqcap X \,\Theta\, A_n$
$X$ possible overlap $A$:
$X \,\Theta\, A \sqcap A \equiv \bigcup_{i=1}^{n} A_i \rightarrow X \,\Theta\, A_1 \sqcap \ldots \sqcap X \,\Theta\, A_n$
$A_m$ equals $X$:
$A_m \equiv X \sqcap A \equiv \bigcup_{i=1}^{n} A_i \rightarrow A \sqsupseteq X$
$A_m$ is contained in $X$:
$A_m \sqsubseteq X \sqcap A \equiv \bigcup_{i=1}^{n} A_i \rightarrow A \; o \; X$
$A_m$ contains $X$:
$A_m \sqsupseteq X \sqcap A \equiv \bigcup_{i=1}^{n} A_i \rightarrow A \sqsupseteq X$
$A_m$ overlaps $X$:
$A_m \; o \; X \sqcap A \equiv \bigcup_{i=1}^{n} A_i \rightarrow A \; o \; X$
$A_m$ possible overlap $X$:
$A_m \,\Theta\, X \sqcap A \equiv \bigcup_{i=1}^{n} A_i \rightarrow A \,\Theta\, X$

Listing 6.31 depicts rules that match complex classes, which are defined by the intersection of other classes, to compute new relationships (see also Table C.10 and Table C.11 in Appendix C.2).

Listing 6.31: Rules matching $A \equiv \bigcap_{i=1}^{n} A_i \wedge 1 \leqslant m \leqslant n$

$X$ equals $A$:
$X \equiv A \sqcap A \equiv \bigcap_{i=1}^{n} A_i \rightarrow X \sqsubseteq A_1 \sqcap \ldots \sqcap X \sqsubseteq A_n$
$X$ contained in $A$:
$X \sqsubseteq A \sqcap A \equiv \bigcap_{i=1}^{n} A_i \rightarrow X \sqsubseteq A_1 \sqcap \ldots \sqcap X \sqsubseteq A_n$
$X$ contains $A$:
$X \sqsupseteq A \sqcap A \equiv \bigcap_{i=1}^{n} A_i \rightarrow X \; o \; A_1 \sqcap \ldots \sqcap X \; o \; A_n$
$X$ overlaps $A$:
$X \; o \; A \sqcap A \equiv \bigcap_{i=1}^{n} A_i \rightarrow X \; o \; A_1 \sqcap \ldots \sqcap X \; o \; A_n$
$X$ possible overlap $A$:
$X \,\Theta\, A \sqcap A \equiv \bigcap_{i=1}^{n} A_i \rightarrow X \,\Theta\, A_1 \sqcap \ldots \sqcap X \,\Theta\, A_n$
$A_m$ equals $X$:
$A_m \equiv X \sqcap A \equiv \bigcap_{i=1}^{n} A_i \rightarrow A \sqsubseteq X$
$A_m$ contains $X$:
$A_m \sqsupseteq X \sqcap A \equiv \bigcap_{i=1}^{n} A_i \rightarrow A \,\Theta\, X$
$A_m$ overlaps $X$:
$A_m \; o \; X \sqcap A \equiv \bigcap_{i=1}^{n} A_i \rightarrow A \,\Theta\, X$
$A_m$ possible overlap $X$:

$A_m \ \Theta \ X \sqcap A \equiv \bigcap_{i=1}^{n} A_i \rightarrow A \ \Theta \ X$

Listing 6.32 depicts rules for relationships between a class X and a complex class A, which is built by the complement of B (see also Table C.12 and Table C.13 in Appendix C.2).

Listing 6.32: Rules matching $A \equiv \neg B$

$X$ equals $A$:
$X \equiv A \sqcap A \equiv \neg B \rightarrow X \equiv \neg B$
$X$ contained in $A$:
$X \sqsubseteq A \sqcap A \equiv \neg B \rightarrow X \sqsubseteq \neg B$
$X$ contains $A$:
$X \sqsupseteq A \sqcap A \equiv \neg B \rightarrow X \ \Theta \ B$
$X$ overlaps $A$:
$X \ o \ A \sqcap A \equiv \neg B \rightarrow X \ \Theta \ B$
$X$ possible overlap $A$:
$X \ \Theta \ A \sqcap A \equiv \neg B \rightarrow X \ \Theta \ B$
$B$ equals $X$:
$B \equiv X \sqcap A \equiv \neg B \rightarrow A \equiv \neg X$
$B$ is contained in $X$:
$B \sqsubseteq X \sqcap A \equiv \neg B \rightarrow A \ \Theta \ X$
$B$ contains $X$:
$B \sqsupseteq X \sqcap A \equiv \neg B \rightarrow \neg X \sqsupseteq A$
$B$ overlaps $X$:
$B \ o \ X \sqcap A \equiv \neg B \rightarrow A \ \Theta \ X$
$B$ possible overlap $X$:
$B \ \Theta \ X \sqcap A \equiv \neg B \rightarrow A \ \Theta \ X$

### Conclusions

In this section we presented a way to represent metamodels and ontological groundings of metamodels. We developed a set of reasoning rules that allow us to infer relationships between metamodels on the basis of the introduced representation concepts. We used the ontology representation and the reasoning rules to implement the process modelling case study (see Section 6.5.1) and showed the applicability of these techniques in the context of OntMT. As discussed in Section 6.6, the representation formalisms and the reasoning can be further improved by providing solutions that lie in OWL DL or by using reasoning calculi that support a local domain approach.

## 6.4.4 Sem-MT-Component

The Sem-MT-Component implements the core part of the OntMT approach. It provides the main functionality of the Sem-MT-Tool by making use of the inference results of the Ontology TS and computing modification programs for the generation and evolution of model transformations in the MDA TS. In this section we describe our realization of the correlation algorithm used to compute model transformation programs. We present a rating that can be used to select the 'best' generated model transformation and summarize with future extensions of the correlation algorithm.

### Correlation

Algorithm 6.1 in Section 6.3.2 illustrates the correlation algorithm implemented by the Sem-MT-Component to generate new model transformations. The Sem-MT-Component

takes as input an initial model transformation, the metamodel which has to be substituted in the model transformation, and the new metamodel.[12] In a first step, it requests the model manipulator to compute a set of all classes and properties *Cs* that have to be substituted in the initial model transformation. Second, it invokes the inference component to obtain a set of possible substitutions for all classes and properties to substitute *CSss*. Next, the computation of a substitution proposal *SP* begins. A substitution proposal contains the model transformation modification program, the problems that occur by applying the substitutions to a model transformation, and a rating of the performed substitutions. After a substitution proposal is calculated by the Sem-MT-Component, the model manipulator performs the substitution and validates the generated model transformation. Then the substitution proposal is rated by the Sem-MT-Component. The Sem-MT-Component tries to compute alternative substitution proposals until the application of a substitution proposal does not lead to any problems, or no further substitution proposals can be found. Finally, the Sem-MT-Component chooses the substitution proposal with the best rating from all computed substitution proposals *SPs* and the new model transformation is generated on the basis of this substitution proposal.

Algorithm 6.4 is used to obtain possible substitutions for the concepts of the metamodel that shall be substituted in the model transformation (*oldMM*). The reasoner is invoked in line 4 and returns a set of reasoning results (*RR*)[13]. From the reasoning result only relationships are chosen, where the source is the concept *c* and the target concept is part of the new metamodel (*newMM*). We have specified the query on the reasoning results in relational algebra [58, 59, 155].

Table 6.4: Obtain reasoning results

1: **function** FINDSUBSTITUTIONS(*c*, *oldMM*, *newMM*) : *ReasoningResults*
2: $RR \leftarrow \varnothing$ ▷ set of reasoning results
3:
4: $RR \leftarrow$ PERFORMREASONING(*oldMM*, *newMM*) ▷ reasoning returns 3-tuples
5: **return** PERFORMQUERY( $\pi_{x_1,x_2,x_3}((RR \ltimes_{[x_1]} (\{(c)\})) \ltimes_{[x_3]} (newMM))$ )
6: **end function**

The calculation of a new substitution proposal distinguishes between two main cases (see Algorithm 6.5) calculating an initial substitution proposal and calculating an alternative substitution proposal. The calculation of an initial substitution proposal takes only <*equal*> relationships between metamodel elements into account. If there doesn't exist such a relationship for a concept *c*, a removal substitution is added to the substitution proposal (see *(c,NULL)* in line 9). To calculate an alternative substitution proposal, also other relationships than <*equal*> between the metamodel elements are considered. For each concept *c*, that caused a problem in the execution of the previous substitution proposal, the algorithm tries to find a better substitution. The choice of the new proposed substitutions depends on heuristics that try to find a substitution that best fits for the initial model transformations.

[12] Call-by-reference is used for all parameters of the algorithms described in this section.
[13] The reasoner returns 3-tuples, which are of the form *(Subject,Predicate,Object)*.

Table 6.5: Calculate substitution proposal

```
1:  function calculateSubstitutionProposal(lastSP, SPs, Cs, CSss) : SubstitutionProposal
2:      newSP.subst ← ∅
3:      RR ← ∅
4:
5:      if SPs = ∅ then                                   ▷ calculate initial substitution proposal
6:          for all c ∈ Cs do
7:              RR ← performQuery( π_{x1,x3} (σ_{x2='equal'} (CSss ⋉_[x1] ({(c)}))) )
8:              if RR = ∅ then
9:                  RR ← {(c,NULL)}
10:             end if
11:             newSP.subst ← newSP.subst ∪ first(RR)
12:         end for
13:     else                                              ▷ calculate alternative substitution proposal
14:         for all c ∈ Cs do
15:             if causedProblem(c, lastSP) then
16:                 RR ← performQuery( π_{x1,x3} (CSss ⋉_[x1] ({(c)})) )
17:                 RR ← RR \ usedSubstitutions(c, CSss)
18:                 newSP.subst ← newSP.subst ∪ chooseSubstitutions(RR)   ▷ usage of heuristics
19:             else
20:                 newSP.subst ← newSP.subst ∪ getSubstitution(c, lastSP)
21:             end if
22:         end for
23:     end if
24:     return newSP
25: end function
```

### Rating substitutions proposals

The choice of the substitution proposal, which is used to generate the new model transformation, is based on the ratings of the substitution proposals. A rating of a substitution proposal is a measure of the generated model transformation's quality. The rating is based on factors that are measured for each substitution proposal:

- *Problems occurring in the substitution proposal*: This measure counts the problems that were detected by the validator in the generated model transformation. The measure distinguishes between the different kinds of problems and assigns different weights to the various problem types according to the severity.
- *Number of concepts that could be substituted*: This measure counts how many classes and properties could be substituted. From this measure it can be derived how many concepts could not be substituted.
- *Relationships used for substitution*: This measure counts and rates the relationships that are used in the modification program of the substitution proposal. In general a substitution derived from an <*equal*> relationship gets a better rating than substitutions derived from other relationships.
- *Position of the problems and used relationships in the model transformation*: This is an optional factor that can influence the three other ratings described above. The assumption for this factor is, that some relations of the model transformation are more important to the overall result than others.

Algorithm 6.6 depicts a sample rating of substitution proposals. For each substitution proposal two rating values are computed. One rating counts the problems that were detected in the generated model transformation after executing the substitution proposal. The second rating is a value for the substitutions used in the substitution proposal. Each substitution used in the substitution proposal is multiplied with a factor that is highest for <*equal*> relationships.

Table 6.6: Rating of substitution proposals

```
1: function RATESUBSTITUTION(SP) : SP
2:     SP.rateProblems ← #( GETPROBLEMS(SP) )
3:     SP.rateSubst ← SP.rateSubst + 10 * PERFORMQUERY( COUNT(σ_{x2='equal'}(SP.rateSubst)) )
4:     SP.rateSubst ← SP.rateSubst + 7 * PERFORMQUERY( COUNT(σ_{x2='specific'}(SP.rateSubst)) )
5:     SP.rateSubst ← SP.rateSubst + 5 * PERFORMQUERY( COUNT(σ_{x2='general'}(SP.rateSubst)) )
6:     ...
7:     return SP
8: end function
```

Finally, the substitution proposal with the best rating is chosen to generate the new model transformation (see Algorithm 6.7). In our basic correlation algorithm the substitution proposal with the fewest detected problems is chosen. If this choice has more than one result, we also consider the ratings of the substitutions used in the substitution proposals.

Table 6.7: Choosing a substitution proposal

```
1: function CHOOSESUBSTITUTIONPROPOSAL(CSss, SPs) : SubstitutionProposal
2:     remSPs ← ∅                                   ▷ remaining substitution proposals in choice
3:
4:     remSPs ← PERFORMQUERY( π_SP(GROUPBY(SPs,SP,MIN,SP.rateProblems)) )
5:     if #(remSPs) == 1 then
6:         return FIRST(remSPs)
7:     end if
8:
9:     remSPs ← PERFORMQUERY( π_SP(GROUPBY(remSPs,SP,MAX,SPs.rateSubst)) )
10:    if #(remSPs) == 1 then
11:        return FIRST(remSPs)
12:    end if
13:    ...
14:    return FIRST(remSPs)
15: end function
```

Concrete ratings depend always on the purpose OntMT is used for. For the different application scenarios separate metrics are defined. A metric, which was developed for automated mapping generation, will put more emphasis on an executable model transformation than on the relationships used for substitution. If OntMT is used to support developers in adjusting their model transformations, OntMT will only make suggestions to the developer. Hence, the metric puts more emphasis on exact substitutions of metamodel elements than on the execution of the new model transformation.

**Summary**

As we illustrate in the process modelling case study (see Section 6.5.1), the described basic correlation algorithm can be used to automatically generate model transformations. However, there are a few issues that can be improved in future. At the moment the correlation algorithm tries to find alternative substitutions for all concepts where problems occurred in one iteration. With this strategy one might loose substitution solutions that generate better model transformations. Hence, we propose to apply only one new substitution to every alternative substitution proposal. Also the implementation of backtracking would help to ensure that the correlation algorithm does not miss adequate substitution solutions. One can find a critical discussion about the runtime performance of the correlation algorithm in Section 6.6.2.

## 6.5 Case Studies

To show the applicability of the OntMT approach, we conducted two case studies with our prototypical implementation. Each case study realizes one of the application scenarios OntMT was developed for (see Section 6.2). The first case study in Section 6.5.1 deals with the domain of process modelling and realizes the automated generation of model transformations scenario described in Section 6.2.1. The second case study in Section 6.5.2 realizes the model transformation evolution scenario described in Section 6.2.2. It applies OntMT to model transformations that evolve in the context of the OMG's standardization process for an UML Profile and Metamodel for Services (UPMS).

### 6.5.1 Mapping Generation for Process Modelling

This section presents a case study performed with OntMT for the automated generation of model transformations scenario described in Section 6.2.1. The case study deals with the domain of process modelling. In this domain various process modelling formats and metamodels exist [30, 225, 226, 238, 261]. This case study makes use of a generic metamodel *Process* for modelling processes and a metamodel *EPC* that can be used to represent EPCs. As an ontology for process modelling we use the Web Ontology Language for Services (OWL-S) [304, 305], which is under standardization at the W3C.

The goal of the case study is to automatically generate model transformations between different metamodels by using ontologies and reasoning technology. We will use OntMT to generate a QVT Relations model transformation between the metamodels *Process* and *EPC*.

First, we will present the input that is necessary to start the execution of OntMT. These are the metamodels, the RO, and the bindings of the metamodels to the RO. Second, we describe how the bootstrapping for the *Process* metamodel is performed. Third, the reasoner of the Sem-MT-Tool infers relationships between the two metamodels *Process* and *EPC*. Since generating a new model transformation with OntMT is an incremental process, we describe the computation of substitution proposals, the execution of the MT modifications, and the validation of the generation results in separate paragraphs for each substitution proposal (*SP1-SP3*). Finally, the substitution proposals are rated and the substitution proposal with best rating is chosen to generate the new model transformation.

### Input to OntMT

Input for the generation of a *Process* to *EPC* model transformation are the metamodels (*Process* and *EPC*), the RO, and the bindings of the metamodels to the RO.

**Process Metamodel** Figure 6.25 depicts the *Process* metamodel. A *Process* consists of *Steps* and *Flows*. A *Step* can have in flows (*+inFlow*) and out flows (*+outFlow*). *Step* is an abstract class with the realizations *Task*, *Decision*, and *Merge* inheriting from it.

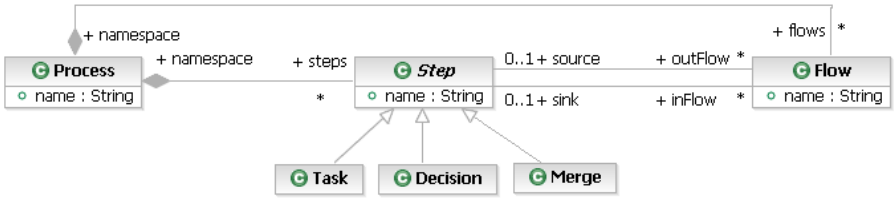


Figure 6.25: Metamodel *Process*

**EPC Metamodel** Figure 6.26 depicts the *EPC* metamodel. An *Epc* consists of *Functions*, *ControlElements*, and *Connectors*. The abstract class *ControlElement* is the top node in the inheritance hierarchy representing EPC nodes. *ControlElement* is also an abstract class with the two subclasses *Join* and *Split*. *Connector* can have source and target *Epcelements*. Therefore *Connector* has relationships to *Function*, *Join*, and *Split*.

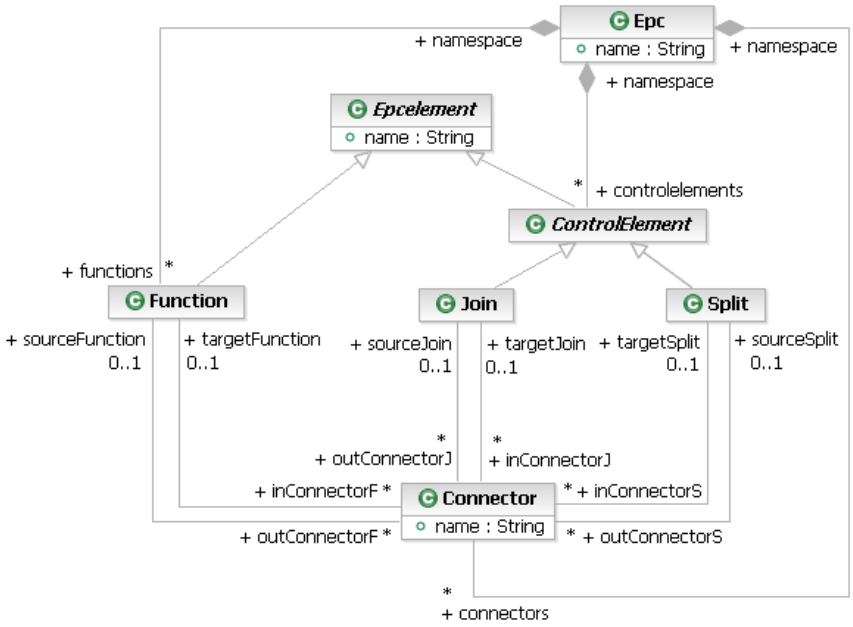


Figure 6.26: Metamodel *EPC*

**Reference Ontology** Figure 6.27 depicts the RO. The RO is an excerpt of the *ServiceModel* of OWL-S. The concepts *Process*, *AtomicProcess*, and *CompositeProcess* inherit from *ServiceModel*. *AtomicProcess*, *ControlContruct*, *Split*, and *Join* inherit from *ProcessComponent*. A *CompositeProcess* is composed of *ProcessComponent* and the *FollowedBy* class, which are separate concepts in OWL-S. *ProcessComponent* and *FollowedBy* are connected by the object properties *connected* and *followed* by.

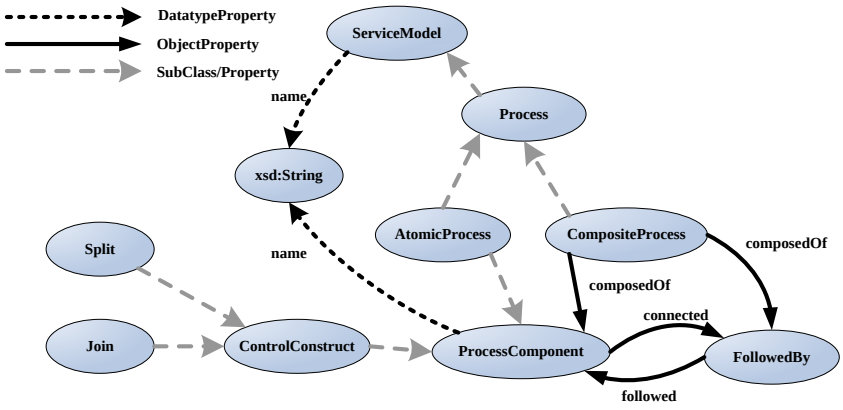


Figure 6.27: Reference ontology for process modelling

**Binding of Metamodels to the Reference Ontology** For the binding (ontological grounding) we use a notation similar to Semantic Mediation and Application Interoperability Language (SMAIL) [191]. *'=:'* stands for a lossless annotation, where the annotation fully captures the intended meaning. *'>:'* denotes an overspecification, where the level of refinement of the annotated element is greater than the level of refinement of the concepts in the RO. *'=:'* and *'>:'* correspond to the relationship types $\equiv$ and $\sqsubseteq$ used for reasoning (see Section 6.4.3). Table 6.8 represents the binding of the *Process* metamodel to the RO. Table 6.9 represents the binding of the *EPC* metamodel to the RO.

### Bootstrapping of the Initial Model Transformation

In a first step, the bootstrap model transformation is generated by traversing the metamodel *Process* via its containment relationships and mapping the associations occurring in the metamodel. The bootstrapping process works as follows:

- For each class in the *Process* metamodel one *top relation* mapping rule is generated in the bootstrap model transformation (*Process*, *Flow*). If there occurs inheritance, mapping rules for the concrete leaf classes (*Task*, *Decision*, *Merge*) are generated instead of mapping the abstract superclass (*Step*). This enhances the granularity of the model transformation specification. The mandatory properties are specified as part of the top relation matching the classes (in the *ProcessToProcess* relation the *name* property; see Listing 6.33). Optional properties would be shifted to separate relations, which are used to further constrain the *top relations* via *where*-clauses.

| | | |
|---|---|---|
| Process | =: | CompositeProcess |
| Process.name | =: | CompositeProcess.name |
| Process.steps | =: | CompositeProcess.composedOf |
| Process.flows | =: | CompositeProcess.composedOf |
| Step | =: | ProcessComponent |
| Step.name | =: | ProcessComponent.name |
| Step.outFlow | =: | ProcessComponent.connected |
| Step.inFlow | =: | ProcessComponent.inverseOf(followed) |
| Step.namespace | =: | ProcessComponent.inverseOf(composedOf) |
| Task | =: | AtomicProcess |
| Task.name | =: | AtomicProcess.name |
| Task.outflow | =: | AtomicProcess.connected |
| Task.inFlow | =: | AtomicProcess.inverseOf(followed) |
| Task.namespace | =: | AtomicProcess.inverseOf(composedOf) |
| Decision | =: | Split |
| Decision.name | =: | Split.name |
| Decision.outFlow | =: | Split.connected |
| Decision.inFlow | =: | Split.inverseOf(followed |
| Decision.namespace | =: | Split.inverseOf(composedOf) |
| Merge | =: | Join |
| Merge.name | =: | Join.name |
| Merge.outFlow | =: | Join.connected |
| Merge.inFlow | =: | Join.inverseOf(followed) |
| Merge.namespace | =: | Join.inverseOf(composedOf) |
| Flow | =: | FollowedBy |
| Flow.name | =: | FollowedBy.name |
| Flow.namespace | =: | FollowedBy.namespace |
| Flow.sink | =: | FollowedBy.followed |
| Flow.source | =: | FollowedBy.inverseOf(connected) |

Table 6.8: Binding of the *Process* metamodel to the reference ontology

| | | |
|---|---|---|
| Epc | =: | CompositeProcess |
| Epc.name | =: | CompositeProcess.name |
| Epc.connectors | =: | CompositeProcess.composedOf |
| Epc.functions | >: | CompositeProcess.composedOf |
| Epc.controlelements | >: | CompositeProcess.composedOf |
| Epcelement | =: | ProcessComponent |
| Epcelement.name | =: | ProcessComponent.name |
| Function | =: | AtomicProcess |
| Function.name | =: | AtomicProcess.name |
| Function.outConnectorF | >: | AtomicProcess.connected |
| Function.inConnectorF | >: | AtomicProcess.inverseOf(followed) |
| Function.namespace | =: | AtomicProcess.inverseOf(composedOf) |
| Join | =: | Join |
| Join.name | =: | Join.name |
| Join.outConnectorJ | >: | Join.connected |
| Join.inConnectorJ | >: | Join.inverseOf(followed) |
| Join.namespace | =: | Join.inverseOf(composedOf) |
| Split | =: | Split |
| Split.name | =: | Split.name |
| Split.outConnectorS | >: | Split.connected |
| Split.inConnectorS | >: | Split.inverseOf(followed) |
| Split.namespace | =: | Split.inverseOf(composedOf) |
| Connector | =: | FollowedBy |
| Connector.name | >: | FollowedBy.name |
| Connector.namespace | >: | FollowedBy.namespace |
| Connector.sinkFunction | >: | FollowedBy.followed |
| Connector.soureFunction | >: | FollowedBy.inverseOf(connected) |
| Connector.sinkJoin | >: | FollowedBy.followed |
| Connector.sourceJoin | >: | FollowedBy.inverseOf(connected) |
| Connector.sinkSplit | >: | FollowedBy.followed |
| Connector.souceSplit | >: | FollowedBy.inverseOf(connected) |

Table 6.9: Binding of the *EPC* metamodel to the reference ontology

Listing 6.33: Bootstrapping top relations code for classes

```
top relation ProcessToProcess {
  n: String;
  checkonly domain prc_1 p_1:Process {
    name=n
    };
  enforce domain prc_2 p_2:Process {
    name=n
    };
}
```

- Containment associations are realized in the bootstrap model transformation as properties of the contained elements (*namespace*) and constrain the *top relations* via *when*-clauses (e.g. in the *FlowToFlow* relation in Listing 6.34).

Listing 6.34: Bootstrapping *when*-clauses for containment associations

```
top relation FlowToFlow {
  n: String;
  checkonly domain prc_1 f_1:Flow {
    namespace=p_1:Process {},
    name=n
    };
  enforce domain prc_2 f_2:Flow {
    namespace=p_2:Process {},
    name=n
    };
  when
  {
    ProcessToProcess(p_1,p_2);
  }
  where
  {
    Flow2Flow_sink(f_1,f_2);
    Flow2Flow_source(f_1,f_2);
  }
}
```

- Other associations are realized via separate relations in the bootstrap model transformation (*StepToStep_out*, *StepToStep_in*, *FlowToFlow_sink*, and *FlowTo than <equal>Flow_source*). The *StepToStep_in* relation is shown as an example in Listing 6.35.

Listing 6.35: Bootstrapping associations as separate relations

```
relation StepToStep_in {
  n: String;
  checkonly domain prc_1 s_1:Step {
    inFlow=in_1:Flow {
      name=n
      }
    };
  enforce domain prc_2 s_2:Step {
    inFlow=in_2:Flow {
      name=n
      }
    };
}
```

The complete QVT Relations code of the bootstrapped model transformation can be found in Appendix C.3.1.

### Reasoning

Before a substitution proposal can be computed by OntMT, the reasoner has to infer relationships between the two metamodels *Process* and *EPC*. For the case study, the reasoner infers[14] 128 relationships between the two metamodels. The relations are represented as triples, which have the form <*subject*> <*predicate*> <*object*>. However, not all reasoning results are used by OntMT, since some of the inferred relationships are stronger than other inferred relationships. For example, OntMT will only use the relationship *A* <*equal*> *B* though there might be other relationships like *A* <*special*> *B*. In the case study 79 OntMT uses inferred relationships.

Listing 6.36 depicts the relationships, that were infered for the *Process*. A complete overview of the reasoning results relevant for OntMT in this case study can be found in Appendix C.3.2.

Listing 6.36: Reasoning results for *Process*

```
<urn:ua:pvs/Process> <equal> <urn:ua:pvs/Epc>
<urn:ua:pvs/Process> <possibleoverlap> <urn:ua:pvs/Join>
<urn:ua:pvs/Process> <possibleoverlap> <urn:ua:pvs/ControlElement>
<urn:ua:pvs/Process> <possibleoverlap> <urn:ua:pvs/Split>
<urn:ua:pvs/Process> <possibleoverlap> <urn:ua:pvs/Epcelement>
<urn:ua:pvs/Process> <possibleoverlap> <urn:ua:pvs/Function>

<urn:ua:pvs/Process:flows> <equal> <urn:ua:pvs/Epc:connectors>

<urn:ua:pvs/Process:steps> <general> <urn:ua:pvs/Epc:controlelements>
<urn:ua:pvs/Process:steps> <general> <urn:ua:pvs/Epc:functions>

<urn:ua:pvs/Process:name> <equal> <urn:ua:pvs/Epc:name>
<urn:ua:pvs/Process:name> <possibleoverlap> <urn:ua:pvs/Split:name>
<urn:ua:pvs/Process:name> <possibleoverlap> <urn:ua:pvs/Epc:name>
<urn:ua:pvs/Process:name> <possibleoverlap> <urn:ua:pvs/Epcelement:name>
<urn:ua:pvs/Process:name> <possibleoverlap> <urn:ua:pvs/Join:name>
<urn:ua:pvs/Process:name> <possibleoverlap> <urn:ua:pvs/ControlElement:name>
<urn:ua:pvs/Process:name> <possibleoverlap> <urn:ua:pvs/Function:name>
```

With this input, metamodels, reference ontology, ontological groundings, bootstrap transformation, and possible substitutions, the computation of the substitution of $N'_{Process}$ with $N_{EPC}$ can start:

### Calculating Substitution Proposals

**Substitution Proposal 1** In the first substitution proposal, the Sem-MT-Component considers only facts with the relationship <*equal*>, in order to find the best possible substitution. Since for the properties <*Process:Flow:outFlow*> and <*Process:Flow:inFlow*> no substitution in the context of <*EPC:Epcelement*> is possible, these properties are added as removal substitutions to the model transformation modification program. This is done in the hope that it does not affect the model transformation. The substitutions of the first substitution proposal can be found in Listing 6.37.

[14] The reasoning was conducted with the reasoning rule presented in Section 6.4.3 and implemented with the JENA general purpose reasoner.

Listing 6.37: Substitution proposal 1 (*SP1*)

```
SubstitutionProposal1 (Process ; EPC) {
  CL1 ( Process → EPC ) {
    PL1 ( Process.name → EPC.name );
  } ;
  CL2 ( Step → Epcelement ) {
    PL2a ( Step.name → Epcelement.name );
    PL2b ( Step.inFlow → );
    PL2c ( Step.outFlow → );
  } ;
  CL3 ( Task → Function ) {
    PL3a ( Task.name → Function.name );
    PL3b ( Task.namespace → Function.namespace );
  } ;
  CL4 ( Decision → Join ) {
    PL4a ( Decision.name → Join.name );
    PL4b ( Decision.namespace → Join.namespace );
  } ;
  CL5 ( Merge → Split ) {
    PL5a ( Merge.name → Split.name );
    PL5b ( Merge.namespace → Split.namespace );
  } ;
  CL6 ( Flow → Connector ) {
    PL6a ( Flow.name → Connector.name );
    PL6b ( Flow.namespace → Connector.namespace );
    PL6b ( Flow.sink →  );
    PL6b ( Flow.source →  );
  } ;
}
```

The model manipulator generates a new model transformation on the basis of the initial substitution proposal (excerpts are shown in Listing 6.38).

Listing 6.38: QVT Relations code generated with *SP1*

```
transformation ProcessToEPC(prc_1:Process; prc_2:EPC) {

  key EPC::Epc {name};
  key EPC::Epcelement {name};
  key EPC::Function {name, namespace};
  key EPC::Split {name, namespace};
  key EPC::Join {name, namespace};
  key EPC::Connector {name, namespace};

  top relation ProcessToProcess {
    n: String;
    checkonly domain prc_1 p_1:Process {
      name=n
      };
    enforce domain prc_2 p_2:Epc {
      name=n
      };
  }

  top relation TaskToTask {
    n: String;
    checkonly domain prc_1 t_1:Task {
      namespace=p_1:Process {},
      name=n
```

```
      };
    enforce domain prc_2 t_2:Function {
      namespace=p_2:Epc {},
      name=n
      };
    when
    {
      ProcessToProcess(p_1,p_2);
    }
    where {
      StepToStep_out(t_1,t_2);
      StepToStep_in(t_1,t_2);
    }
  }

  ...

    relation StepToStep_out {
    n: String;
    checkonly domain prc_1 s_1:Step {
      outFlow=out_1:Flow {
        name=n
        }
      };
    enforce domain prc_2 s_2:Epcelement {
/* removed PropertyTemplateItem 'Step:outFlow' */
    };
  }

  ...
}
```

The model transformation is validated by the model manipulator, which detects four *Substitution of property failed* problems for *Step:inFlow*, *Step:outFlow*, *Flow:sink*, and *Flow:source*.

**Substitution Proposal 2** Thereon the Sem-MT-Component searches for an alternate substitution, also considering relationships with predicates other than <*equal*>. For a substitution decision it applies a hierarchy, in which the predicate <*equal*> is better than <*special*> and <*special*> is better than <*general*>. The facts provided by the reasoner for *Step:inFlow*, *Step:outFlow*, *Flow:sink*, and *Flow:source* are depicted in Listing 6.39.

Listing 6.39: Additional reasoning results used in *SP2*

```
<urn:ua:pvs/Step:outFlow> <general> <urn:ua:pvs/Function:outConnectorF>
<urn:ua:pvs/Step:outFlow> <general> <urn:ua:pvs/Join:outConnectorJ>
<urn:ua:pvs/Step:outFlow> <general> <urn:ua:pvs/Split:outConnectorS>

<urn:ua:pvs/Step:inFlow> <general> <urn:ua:pvs/Function:inConnectorF>
<urn:ua:pvs/Step:inFlow> <general> <urn:ua:pvs/Join:inConnectorJ>
<urn:ua:pvs/Step:inFlow> <general> <urn:ua:pvs/Split:inConnectorS>

<urn:ua:pvs/Flow:sink> <general> <urn:ua:pvs/Connector:targetJoin>
<urn:ua:pvs/Flow:sink> <general> <urn:ua:pvs/Connector:targetFunction>
<urn:ua:pvs/Flow:sink> <general> <urn:ua:pvs/Connector:targetSplit>

<urn:ua:pvs/Flow:source> <general> <urn:ua:pvs/Connector:sourceJoin>
<urn:ua:pvs/Flow:source> <general> <urn:ua:pvs/Connector:sourceFunction>
<urn:ua:pvs/Flow:source> <general> <urn:ua:pvs/Connector:sourceSplit>
```

Based on its history of previously proposed substitutions[15], the Sem-MT-Component computes a second substitution proposal *SP2*, which e.g. proposes to substitute the *Step:outFlow* property with the three different *outConnector* properties (cp. Listing 6.40).

Listing 6.40: Substitution proposal 2 (*SP2*)

```
1  SubstitutionProposal2 (Process ; EPC) {
2    CL1 ( Process → EPC ) {
3      PL1 ( Process.name → EPC.name );
4    } ;
     CL2 ( Step → Epcelement ) {
6      PL2a ( Step.name → Epcelement.name );
7      PL2b ( Step.inFlow → Function:inConnectorF
8                         Join:inConnectorJ
                          Split:inConnectorS );
10     PL2c ( Step.outFlow → Function:outConnectorF
11                        Join:outConnectorJ
12                        Split:outConnectorS );
     } ;
14   CL3 ( Task → Function ) {
15     PL3a ( Task.name → Function.name );
16     PL3b ( Task.namespace → Function.namespace );
17   } ;
18   CL4 ( Decision → Join ) {
19     PL4a ( Decision.name → Join.name );
20     PL4b ( Decision.namespace → Join.namespace );
21   } ;
22   CL5 ( Merge → Split ) {
23     PL5a ( Merge.name → Split.name );
24     PL5b ( Merge.namespace → Split.namespace );
25   } ;
26   CL6 ( Flow → Connector ) {
27     PL6a ( Flow.name → Connector.name );
28     PL6b ( Flow.namespace → Connector.namespace );
29     PL6b ( Flow.sink →  Connector:targetFunction
30                        Connector:targetSplit
                          Connector:targetJoin );
32     PL6b ( Flow.source →  Connector:sourceFunction
33                        Connector:sourceSplit
34                        Connector:sourceJoin );
     } ;
36 }
```

Listing 6.41 shows excerpts of the QVT Relations code, which is generated by the model manipulator on the basis of *SP2*. For the one-to-many substitution of the properties the model manipulator tries to choose one property that fits the context of the relation. However, it is not possible to find an appropriate substitution (i.e. property) for the relations *StepToStep_in* and *StepToStep_out*, since for example *Function:inConnectorF*, *Join:outConnectorJ*, and *Split:inConnectorS* do not belong to the class *Epcelement*.

Hence, the validator of the model manipulator detects two *Property is not part of class* problems. The substitution of *Flow:sink* and *Flow:source* with a property that does not exactly match them is not detected as problem. This compromise was accepted when building *SP2* and will be considered later in the rating of the substitution proposal.

[15]The Sem-MT-Component has a history of its previous substitution proposals, so that it will not make the same proposal a second time and the search for substitutions terminates.

Listing 6.41: QVT Relations code generated with *SP2*

```
relation StepToStep_out {
  n: String;
  checkonly domain prc_1 s_1:Step {
    outFlow=out_1:Flow {
      name=n
      }
    };
  enforce domain prc_2 s_2:Epcelement {
/* no appropriate property found to substitute 'Step:outFlow'*/
    <null>=out_2:Connector {
      name=n
      }
    };
}

relation StepToStep_in {
  n: String;
  checkonly domain prc_1 s_1:Step {
    inFlow=in_1:Flow {
      name=n
      }
    };
  enforce domain prc_2 s_2:Epcelement {
/* no appropriate property found to substitute 'Step:inFlow'*/
    <null>=in_2:Connector {
      name=n
      }
    };
}

relation Flow2Flow_sink {
  n: String;
  checkonly domain prc_1 f_1:Flow {
    sink=sink_1:Step {
      name=n
      }
    };
  enforce domain prc_2 f_2:Flow {
    targetFunction=sink_2:Step {
      name=n
      }
    };
}

relation Flow2Flow_source {
  n: String;
  checkonly domain prc_1 f_1:Flow {
    source=source_1:Step {
      name=n
      }
    };
  enforce domain prc_2 f_2:Flow {
    sourceFunction=source_2:Step {
      name=n
      }
    };
}
```

**Substitution Proposal 3** The Sem-MT-Component again calculates an alternative substitution proposal *SP3*, where a *Step* is substituted by *Function*, *Join*, and *Split* (see Listing 6.42).

Listing 6.42: Substitution proposal 3 (*SP3*)

```
SubstitutionProposal3 (Process ; EPC) {
  CL1 ( Process → EPC ) {
    PL1 ( Process.name → EPC.name );
  } ;
  CL2 ( Step → Function, Split, Join ) {
    PL2a ( Step.name → Epcelement.name );
    PL2b ( Step.inFlow → Function:inConnectorF
                         Join:inConnectorJ
                         Split:inConnectorS );
    PL2c ( Step.outFlow → Function:outConnectorF
                          Join:outConnectorJ
                          Split:outConnectorS );
  } ;
  CL3 ( Task → Function ) {
    PL3a ( Task.name → Function.name );
    PL3b ( Task.namespace → Function.namespace );
  } ;
  CL4 ( Decision → Join ) {
    PL4a ( Decision.name → Join.name );
    PL4b ( Decision.namespace → Join.namespace );
  } ;
  CL5 ( Merge → Split ) {
    PL5a ( Merge.name → Split.name );
    PL5b ( Merge.namespace → Split.namespace );
  } ;
  CL6 ( Flow → Connector ) {
    PL6a ( Flow.name → Connector.name );
    PL6b ( Flow.namespace → Connector.namespace );
    PL6b ( Flow.sink →  Connector:targetFunction
                        Connector:targetSplit
                        Connector:targetJoin );
    PL6b ( Flow.source →  Connector:sourceFunction
                          Connector:sourceSplit
                          Connector:sourceJoin );
  } ;
}
```

The model manipulator generates a new model transformation on the basis of *SP3* (excerpts are shown in Listing 6.43; the complete QVT Relations code can be found in Appendix C.6). In this the relation *StepToStep_out* is split into three relations, one matching *Function*, one matching *Split*, and one matching *Join*.

The validator of the model manipulator comes to the result, that *SP3* leads to a new model transformation in which none of the problems mentioned above occur.

Listing 6.43: QVT Relations code generated with *SP3*

```
  relation StepToStep_out_1 {
    n: String;
    checkonly domain prc_1 s_1:Step {
      outFlow=out_1:Flow {
        name=n
        }
      };
```

```
8     enforce domain prc_2 s_2:Function {
        outConnectorF=out_2:Connector {
10        name=n
          }
12      };
    }
14
    relation StepToStep_out_2 {
16    n: String;
      checkonly domain prc_1 s_1:Step {
18      outFlow=out_1:Flow {
          name=n
20        }
        };
22    enforce domain prc_2 s_2:Split {
        outConnectorS=out_2:Connector {
24        name=n
          }
26      };
    }
28
    relation StepToStep_out_3 {
30    n: String;
      checkonly domain prc_1 s_1:Step {
32      outFlow=out_1:Flow {
          name=n
34        }
        };
36    enforce domain prc_2 s_2:Join {
        outConnectorJ=out_2:Connector {
38        name=n
          }
40      };
    }
```

### Choosing the Best Substitution Proposal

Since, no further problems occurred in applying *SP3* to the bootstrap model transformation, the Sem-MT-Component stops computing new substitution proposals and compares the substitution proposals on the basis of their ratings. Each row represents the rating for one substitution proposal. The first column depicts the name of the substitution proposal. In the second column one can find the problems that occurred in executing the substitution proposal on the input model transformation. The third and the fourth column depict the number and the type of the substituted concepts in the input model transformation as well as the number and the type of inferred relationships used to compute the substitution proposal.

The Sem-MT-Component decides to use *SP3* to generate the new model transformation. This is based on the consideration that a model transformation generated from *SP2* couldn't be executed due to typing problems and that *SP3* can substitute more concepts of the original model transformation than *SP1*. The model transformation between metamodel $N_{Process}$ to $N_{EPC}$ generated by OntMT can be found in Appendix C.6.

| ***Substitution Proposal*** | ***Problems occurred*** | ***Substituted Concepts*** | ***Used relationships*** |
|---|---|---|---|
| *SP1* | 4x *Substitution of property failed* | 6x class<br>10x property | 16x <*equal*> |
| *SP2* | 2x *Property is not part of class* | 6x class<br>14x property | 16x <*equal*><br>12x <*general*> |
| *SP3* | — | 6x class<br>14x property | 15x <*equal*><br>15x <*general*> |

Table 6.10: Rating of the substitution proposals

**Summary**

This case study shows that the OntMT approach can be applied to automatically generate model transformations between different metamodels. It illustrates how reasoning results from the Ontology TS can be used to trigger the automated model transformation in an incremental process. This incremental process is used to bridge the gap between the conceptual representation in the Ontology TS and the concrete implementations of the (meta)models in the MDA TS [91]. It is described how the validation results of the higher-order model transformations semantics and the computation of mapping model, i.e. the modification program, complement one another.

Though the generation of the new model transformation was performed totally automatically in this case study, it may be necessary, due to both technical reasons or human factors, to consider human interaction in the generation process. A more extensive and critical discussion about this and other issue concerning OntMT can be found in Section 6.6. The model transformations presented in this case study have been implemented with QVT Relations and executed and tested with ModelMorf [291].

### 6.5.2 Model Transformation Evolution for Service Modelling

This section presents a case study performed with OntMT for the model transformation evolution scenario described in Section 6.2.2. The case study deals with the domain of service modelling, where various projects have been conducted in the recent years. These projects had a multitude of metamodels and UML profiles as an outcome. At the moment the OMG preforms a standardization process for an UML Profile and Metamodel for Services (UPMS) [232].

The goal of the case study is to evolve model transformations that have been specified before the UPMS standardization process. The goal is to reuse the knowledge encoded into a transformation between the SPL4AOX [23] and the PIM4SOA [30] metamodel. This transformation was developed to transform SPL4AOX models in the PIM4SOA format, which was developed as a platform independent service model interchange format. SPL4AOX models shall now be transformed to models of the initial submission of the UPMS standard [236].

We limited the scope of this case study to the core concepts for service modelling. However, this did lead to trivial model transformations and substitutions. As we will describe later, the initial model transformation encodes non-trivial knowledge about the transformation of SPL4AOX models into PIM4SOA models. This justifies the application of OntMT and the model transformation modification technique to reuse this transformation. The model transformation modification has also to make use of a variety of different substitutions in the service modelling case study.

### Input to OntMT

Input for the generation of a SPL4AOX to UPMS model transformation are the initial model transformation (*SPL4AOXtoPIM4SOA*), the metamodels (*SPL4AOX, PIM4SOA, and UPMS*), and the model transformation modification program (PIM4SOA to UPMS mapping).

**SPL4AOX Metamodel** Figure 6.28 depicts the SPL4AOX metamodel. A *SPL4AOX-model* consists of *ServiceProviders* and *ServiceCollaborations*. Services are collaborations between exactly two *Roles*: the service provider (*Role* with *roleType='PROVIDER'*) and the service requester (*Role* with *roleType='REQUESTOR'*). They are realized as *ServiceCollaborations*. *Roles* define how partners participate in and contribute to collaborations. Partners can offer services in the role of a service provider or request services as a service requestor. A collaboration references the participating roles (+*role*). Services can be defined through the combination of other services. Elementary, not composed collaborations (in most cases these are binary collaborations) are the fundamental parts of composed collaborations. Composed collaborations can be specified through *CollaborationUse* (+*subservices*). The mapping from roles (+*role*) of elementary collaborations to the roles (+*boundRole*) of composed collaborations is defined by a *RoleBinding*. A *ServiceProvider* is an abstract description of a service component, which defines its structure and visible behavior. A *ServiceProvider* is an entity that offers services and defines conditions for the usage of these services. A service provider contains *Roles* (+*role*), with whom it can participate in collaborations. For instance, it can offer and request services in collaborations through these roles.

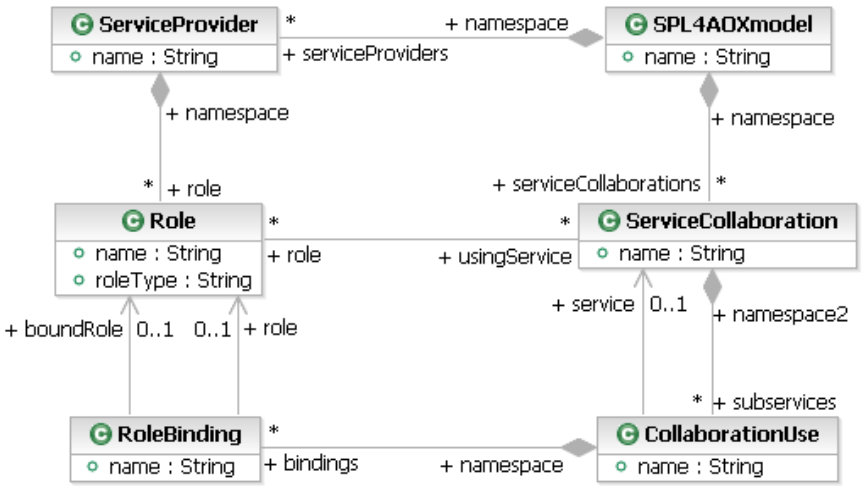


Figure 6.28: Metamodel *SPL4AOX*

**PIM4SOA Metamodel** One can find the concepts of the PIM4SOA metamodel relevant for this case study in Figure 6.29. In PIM4SOA, *Collaborations* are used to specify interactions between service consumers and providers. Collaborations can be defined

on the basis of subcollaborations via *CollaborationUse*, i.e they can be nested to arbitrary depth. A *CollaborationUse* specifies a link (+*collaboration*) to the used *Collaborations* definition and a *RoleBinding* (+*bindings*). The *RoleBinding* relates the specific roles within the current collaboration (+*boundRole*) to the roles of the used collaboration (+*role*). A *Role* can be requester or a provider of a service (indicated by the property *roleType*). A *ServiceProvider* specifies an entity that provides or consumes services. Therefore it takes on roles through which it +*participates* in collaborations and realizes roles in collaborations via *CollaborationUse*. The *RoleBinding* is used to specify which roles of the collaboration are realized by the roles of the service provider.

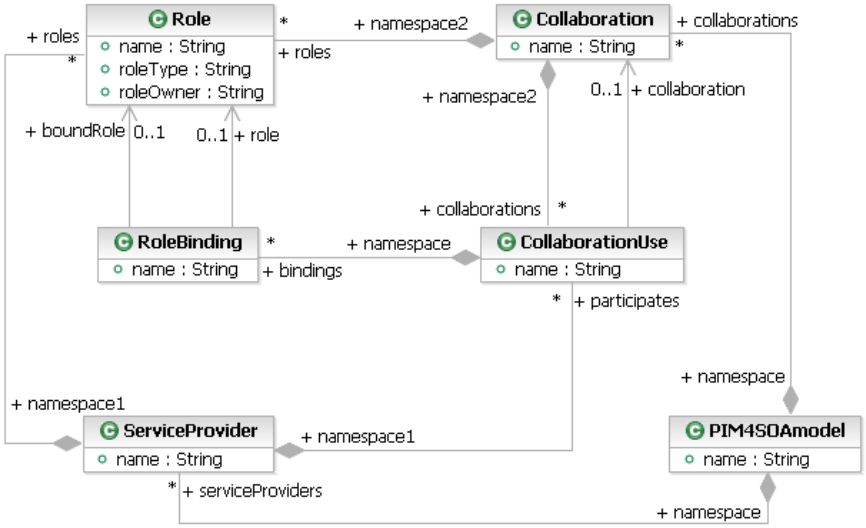


Figure 6.29: Metamodel *PIM4SOA*

**UPMS Metamodel** In the UPMS metamodel a *Component* (see Figure 6.30) specifies provided services (+*services*) and consumes requisitions (+*requisitions*). A *Service* is the mechanism by which a provider makes available capabilities that meet the needs of consumer requisitions. A *Requisition* is the mechanism by which a consumer accesses capabilities of provider services. UPMS captures the service and participant requirements in a *Contract* that specifies what is being accomplished, the roles (*Role*) that participate in achieving the desired result, the responsibilities of those roles, etc. A *Fulfillment* indicates the ability of a service participant (*Component*) to fulfill a *Contract*. The *Fulfillment* has bindings that indicate what role each part plays in the contract. *Fulfillment* can also be used to nest *Contracts* to arbitrary depth.

**SPL4AOXtoPIM4SOA Model Transformation** The input model transformation transforms SPL4AOX models into PIM4SOA models. In the following we present the core mapping rules and describe the non-trivial transformation knowledge that is encoded in the transformation. The complete QVT transformation code can be found in Appendix C.4.1.

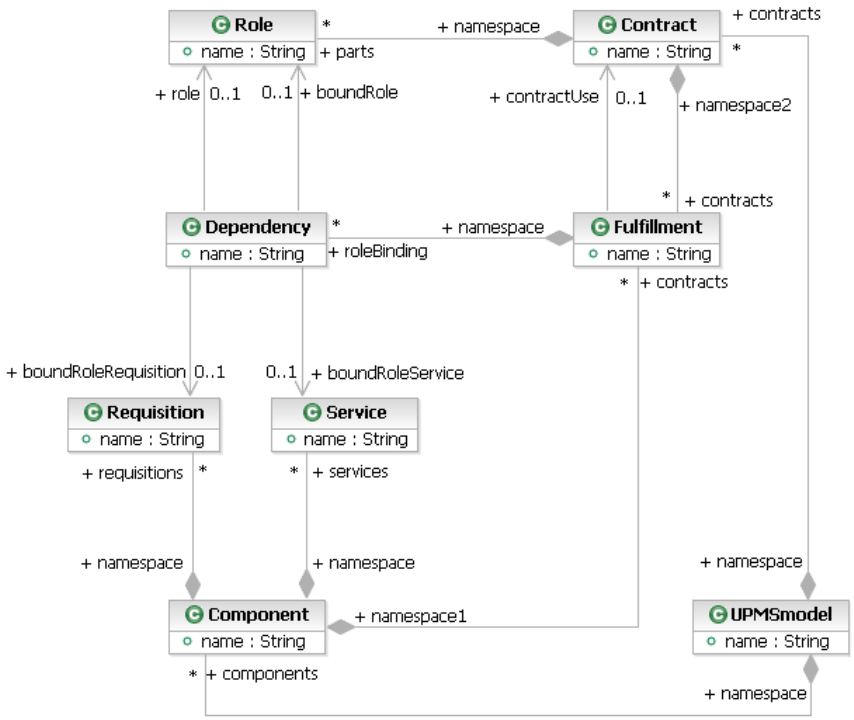


Figure 6.30: Metamodel *UPMS*

**Simple Model Element Mappings** The SPL4AOX elements *SPL4AOXmodel*, *ServiceCollaboration*, and *ServiceProvider* are directly mapped to their correspondents in the PIM4SOA metamodel (*PIM4SOAmodel*, *Collaboration*, and *ServiceProvider*). Listing 6.44 depicts the *SPL4AOXToPIM4SOAmodel* relation that maps *SPL4AOXmodel* to *PIM4SOAmodel*. The variable *n* is used to map the name of the models. The key definition for *PIM4SOAmodel* ensures, that not multiple instances of *PIM4SOAmodel* with the same *name* are created in the enforced domain.

Listing 6.44: *SPL4AOXtoPIM4SOA*: simple model element mapping

```
transformation SPL4AOXtoPIM4SOA(aox:SPL4AOX; soa:PIM4SOA) {

  key PIM4SOA::PIM4SOAmodel {name};
  ...

  top relation SPL4AOXToPIM4SOAmodel {
    n: String;
    checkonly domain aox a_1:SPL4AOXmodel {
      name=n
    };
    enforce domain soa s_1:PIM4SOAmodel {
      name=n
    };
  }

  ...
}
```

**Mapping of Collaboration Nesting** As described in the introduction of the metamodels, *ServiceCollaborations* and *Collaborations* can be nested to arbitrary depth. First of all, roles have to be created for the *Collaborations* in PIM4SOA, since in SPL4AOX *ServiceCollaborations* do not contain roles but only references them. As we can see in Listing 6.45 for each role that is referenced by a *ServiceCollaboration*, the relation *CollaborationRoleToCollaborationRole* creates a role that is contained by the respective *Collaboration*. In the transformation we have to distinguish between roles that are contained by *ServiceProviders* and roles that are contained by *Collaborations* in the PIM4SOA domain. Hence, for the created roles the property *roleOwner* is set to *'C'*.

The roles for mapping the collaboration nesting are straightforward, mapping *CollaborationUse*, *RoleBinding*, and the *Roles*. The QVT transformation code can be found in Appendix C.4.1.

Listing 6.45: *SPL4AOXtoPIM4SOA*: collaboration nesting

```
  key PIM4SOA::Role {name, roleOwner};

  relation CollaborationRoleToCollaborationRole {
    n: String;
    checkonly domain aox a_1:ServiceCollaboration {
      role=ar_1:Role {
        name=n
      }
    };
    enforce domain soa s_1:Collaboration {
      roles=sr_1:Role {
        name=n,
```

```
        roleOwner='C'
      }
    };
  }
```

**Mapping the Participation of Service Providers in Collaborations** To express the participation of *ServiceProviders* in *Collaborations* more transformation knowledge has to be encoded in the QVT transformation code. In SPL4AOX *ServiceCollaborations* simply reference the *Roles* that are provided by the *ServiceProviders*. In PIM4SOA both *ServiceProviders* and *Collaborations* own *Roles*. The participation of *ServiceProvider* in a *Collaboration* is expressed via *CollaborationUse* and *RoleBinding*.

Listing 6.46 depicts the QVT code that implements the necessary transformation logic. The relation *ServiceProviderToServiceProvider* creates for each *ServiceProvider* that participates in a *(Service)Collaboration* a *CollaborationUse* and a *RoleBinding*. The creation of the *RoleBindings* is done by the *roleBindingSP* relations, which are 'invoked' by the *ServiceProviderToServiceProvider* relation via a *where*-clause. The *RoleBinding* references the respective roles of the *ServiceProvider* (via *+boundRole*) and the *Collaboration* (via *+role*).

Listing 6.46: *SPL4AOXtoPIM4SOA*: participation of service providers in collaborations

```
1  top relation ServiceProviderToServiceProvider {
2    n,nr: String;
     checkonly domain aox a_1:ServiceProvider {
4      namespace=am_1:SPL4AOXmodel {},
       name=n,
6      role=ar:Role{
         name=nr,
8        usingService=ac:ServiceCollaboration {}
       }
10   };
     enforce domain soa s_1:ServiceProvider {
12     namespace=sm_1:PIM4SOAmodel {},
       name=n,
14     participates=scu:CollaborationUse {
         name=n+'_SP',
16       collaboration=sc:Collaboration {},
         bindings=srb:RoleBinding {
18         name=n+'_RB'
         }
20     }
     };
22   when
     {
24     SPL4AOXToPIM4SOAmodel(am_1,sm_1);
       CollaborationToCollaboration(ac,sc);
26   }
     where
28   {
       roleBindingSP_Provider(ar,srb);
30     roleBindingSP_Requestor(ar,srb);
     }
32 }

34 relation roleBindingSP_Provider {
     nr: String;
```

```
36      checkonly domain aox a_1:Role {
          name=nr,
38        roleType='PROVIDER'
        };
40      enforce domain soa s_1:RoleBinding {
          role=sRole:Role{
42          name=nr,
            roleOwner='C'
44        },
          boundRole=sBoundRole:Role {
46          name=nr,
            roleOwner='SP',
48          roleType='PROVIDER'
          }
50      };
    }
```

**Mapping of Service Provider Roles** Listing 6.47 depicts the mapping of *Roles*, which are contained by *ServiceProviders* in PIM4SOA models. One rule maps *Roles* with the *roleType='PROVIDER'* and one rule maps *Roles* with the *roleType='REQUESTOR'*. Though this could expressed in one mapping rule, we will see later in the execution of OntMT and the model transformation modification, that this greater granularity is necessary to obtain good results.

Listing 6.47: *SPL4AOXtoPIM4SOA*: service provider roles

```
1   top relation RoleToRole_Provider {
2     n: String;
      checkonly domain aox a_1:Role {
4       namespace=asp_1:ServiceProvider {},
        name=n,
6       roleType='PROVIDER'
      };
8     enforce domain soa s_1:Role {
        namespace1=ssp_1:ServiceProvider {},
10      name=n,
        roleType='PROVIDER',
12      roleOwner='SP'
      };
14    when
      {
16      ServiceProviderToServiceProvider(asp_1,ssp_1);
      }
18  }

20  top relation RoleToRole_Requestor {
      n: String;
22    checkonly domain aox a_1:Role {
        namespace=asp_1:ServiceProvider {},
24      name=n,
        roleType='REQUESTOR'
26    };
      enforce domain soa s_1:Role {
28      namespace1=ssp_1:ServiceProvider {},
        name=n,
30      roleType='REQUESTOR',
        roleOwner='SP'
```

```
32      };
        when
34      {
          ServiceProviderToServiceProvider(asp_1,ssp_1);
36      }
    }
```

**PIM4SOA to UPMS Mapping** Listing 6.48 depicts the PIM4SOA to UPMS mapping, which is used as model transformation modification program in this case study.

Listing 6.48: *PIM4SOAtoUPMSmapping* MT modification program

```
1  PIM4SOAtoUPMSmapping (PIM4SOA ; UPMS) {
2    CL1 ( PIM4SOAmodel → UPMSmodel ) {
3      PL1a ( PIM4SOAmodel:name → UPMSmodel:name );
4    } ;
     CL2 ( ServiceProvider → Component ) {
6      PL2a ( ServiceProvider:name → Component:name );
7      PL2b ( ServiceProvider:namespace → Component:namespace );
8      PL2c ( ServiceProvider:participates → Component:contracts );
9    } ;
10   CL3 ( Collaboration → Contract ) {
11     PL3a ( Collaboration:name → Contract:name );
12     PL3b ( Collaboration:namespace → Contract:namespace );
13     PL3c ( Collaboration:roles → Contract:parts );
14   } ;
     CL4 ( CollaborationUse → Fulfillment ) {
16     PL4a ( CollaborationUse:name → Fulfillment:name );
17     PL4b ( CollaborationUse:namespace2 → Fulfillment:.namespace2 );
18     PL4c ( CollaborationUse:collaboration
                              → Fulfillment:contractUse );
20     PL4d ( CollaborationUse:bindings → Fulfillment:roleBinding );
21   } ;
22   CL5 ( RoleBinding → Dependency ) {
23     PL5a ( RoleBinding:name → Dependency:name );
24     PL5b ( RoleBinding:namespace → Dependency:namespace );
25     PL5c ( RoleBinding:role → Dependency:role );
26     PL5d ( RoleBinding:boundRole
                              → Dependency:boundRole,
28                              Dependency:boundRoleService,
                                Dependency:boundRoleRequisition );
30   } ;
     CL6a ( Role → Role, Service, Requisition ) {
32     PL6a_1 ( Role:name → Role.name,
33                               Service:name,
34                               Requisition:name );
       PL6b_2 ( Role:roleOwner →  );
36   } ;
     CL6b ( Role {roleOwner='C'} → Role ) {
38     PL6b_1 ( Role:name → Role.name );
39     PL6b_2 ( Role:roleOwner →  );
40     PL6b_3 ( Role:roleType →  );
41   } ;
42   CL6c ( Role {roleOwner='SP'} → Service, Requisition ) {
43     PL6c_1 ( Role:name →
44                               Service:name,
                                  Requisition:name );
46     PL6c_2 ( Role:roleOwner →  );
```

```
47      PL6c_3 ( Role:roleType →  );
48      PL6c_4 ( Role:namespace1 → Service:namespace,
49                                 Requisition:namespace );
50    } ;
      CL6d ( Role {roleOwner='SP', roleType='PROVIDER'}
52                               → Service ) {
53      PL6d_1 ( Role:name → Service:name );
54      PL6d_2 ( Role:roleOwner →  );
55      PL6d_3 ( Role:roleType →  );
56      PL6c_4 ( Role:namespace1 → Service:namespace );
57    } ;
58    CL6e ( Role {roleOwner='SP', roleType='REQUESTOR'}
                                 → Requisition ) {
60      PL6e_1 ( Role:name → Requisition:name );
61      PL6e_2 ( Role:roleOwner →  );
62      PL6e_3 ( Role:roleType →  );
63      PL6c_4 ( Role:namespace1 → Requisition:namespace );
64    }
    }
```

### Execution of OntMT

In the following we describe the execution of the model transformation modification for the service modelling case study and provide some examples via code fragments. The complete *SPL4AOXtoUPMS* model transformation, which is the output of the model transformation modification, can be found in Appendix C.4.2.

**One-to-one Substitution** One example of an one-to-one substitution performed in the execution of the case study is the *SPL4AOXToPIM4SOAmodel* relation. In the *soa* domain, *PIM4SOAmodel* has been replaced with *UPMSmodel*. Listing 6.49 depicts the adjusted relation.

Listing 6.49: *SPL4AOXtoUPMS*: one-to-one substitution

```
1   top relation SPL4AOXToUPMSmodel {
2     n: String;
      checkonly domain aox a_1:SPL4AOXmodel {
4       name=n
      };
6     enforce domain soa s_1:UPMSmodel {
        name=n
8     };
    }
```

**One-to-many Substitution** In the case of one-to-many substitutions one has to distinguish between substitutions for classes and substitutions for properties.

**Classes** One-to-many substitutions for classes are executed as splits. In the case study the key *PIM4SOA::Role{name,roleOwner}* is split into three separate keys depicted in Listing 6.50.

Listing 6.50: *SPL4AOXtoUPMS*: one-to-many substitution - classes

```
key UPMS::Role {name};
key UPMS::Service {name};
key UPMS::Requisition {name};
```

**Properties** One-to-many substitutions for classes are executed as alternatives. This means, that the property is chosen for the substitution that best fits the context of the relation. For example, one can find a one-to-many substitution for the *boundRole* property in the relation *roleBindingSP_Provider*. The one-to-many substitution offers the properties *boundRoleRequisition* and *boundRoleService* as possible substitutions. Since, the property should have the type *Service*, the *boundRoleService* property is chosen for the substitution (see Listing 6.51).

Listing 6.51: *SPL4AOXtoUPMS*: one-to-many Substitution - properties

```
relation roleBindingSP_Service {
  nr: String;
  checkonly domain aox a_1:Role {
    name=nr,
    roleType='PROVIDER'
  };
  enforce domain soa s_1:Dependency {
    role=sRole:Role{
      name=nr
    },
    boundRoleService=sBoundRole:Service {
      name=nr
    }
  };
}
```

**Removal Substitution** Examples of removal substitutions can be found in the relation *RoleToRole_Provider*. The properties *roleOwner* and *roleType* are removed from the *soa* domain in the *RoleToRole_Provider* relation. The result can be seen in Listing 6.52.

**Constraining** Finally, we can also find substitutions where the appropriate substitution is selected from a set of mappings on the basis of constraints. In case of the *soa* domain of the *RoleToRole_Provider* relation, the *Role* is constrained with *roleType ='PROVIDER'* and *roleOwner ='SP'*. Hence, as one can see in Listing 6.52, *Service* is chosen to substitute *Role*.

Listing 6.52: *SPL4AOXtoUPMS*: removal and constraining

```
top relation RoleToService {
  n: String;
  checkonly domain aox a_1:Role {
    namespace=asp_1:ServiceProvider {},
    name=n,
    roleType='PROVIDER'
  };

  enforce domain soa s_1:Service {
    namespace=ssp_1:Component {},
```

```
      name=n
12    };
      when
14    {
        ServiceProviderToComponent(asp_1, ssp_1);
16    }
  }
```

**Summary**

The model transformations presented in this case study have been implemented with QVT Relations and executed and tested with ModelMorf [291] and models of the *Purchase Order* example described in [236]. The model transformation modification has been accomplished with the implementation of the higher-order model transformation language (see Section 6.4.2) provided by the OntMT project [272].

The case study shows that OntMT and the model transformation modification approach can very well be applied to real world scenarios like the evolution of metamodels and model transformations in a standardization process. However, one cannot expect that for all model transformation modification executions no problems will be detected. As we will discuss in Section 6.6 this depends on the respective model transformations and the delta between the metamodels.

## 6.6 Assessment of Ontology-based Model Transformation

This section discusses the OntMT approach with respect to its practical application, its limits, and possible weaknesses.

### 6.6.1 Application Areas

OntMT fosters the exchange of models and the evolution of model transformations. Model exchange scenarios are build on the generation of new model transformations, while model transformation evolution scenarios aim at reusing model transformations. As introduced in Section 3.3, one can distinguish between horizontal and vertical model transformations. Horizontal model transformations are mappings between models at a certain abstraction level, where no information is lost and no additional information is added. Vertical model transformations are refinements that add additional information to the generated model about e.g. architecture or platform. Thus, the target model of a refinement is more detailed than the source model.

Figure 6.31 categorizes the support that OntMT can provide to the described application scenarios and the different types of model transformations.

- To exchange models between different DSLs, metamodels and modelling styles, OntMT is able to automatically generate mappings. However, the level of automation depends on how different the DSLs and their modelling approaches are. It may be necessary to provide additional mapping information through an initial model transformation, which cannot be inferred from the ontologies.

- OntMT supports the evolution and reuse of existing mappings. The new model transformation can be either generated from scratch or obtained through adjusting

| | *Mapping* | *Refinement* |
|---|---|---|
| *Model Exchange and Evolution* | autom.gen. (+man) | n/a |
| *Model Transformation Evolution* | autom.gen. autom.mod. | autom.mod. |

Figure 6.31: Application of OntMT to model exchange and model transformation evolution

the existing mapping. The more individual features, which are different to the core structure of the metamodels, are encoded in existing mappings, the more preferable it is to adjust existing mappings. The generation of new mappings is better, if the new metamodel provides extensions to the old one or a new modelling style specifies a fundamentally different composition of modelling elements.

- For the evolution and reuse of refinements, OntMT provides the possibility of automated modification and adjustment of existing model transformations. Refinement model transformations cannot be generated without human interaction, since they contain individual knowledge about software architecture or the platform, e.g. patterns like broker, model-view-controller, etc.

### 6.6.2 Evaluation

The OntMT approach adjusts initial model transformations in order to generate or maintain model transformations. Since mapping knowledge is captured in bindings of the metamodels to the reference ontology, one could favour an approach that derives model transformation rules directly from these bindings. This may very well work for model exchange scenarios. However, in model transformation evolution scenarios the model transformation itself would have to be encoded in the bindings. In our opinion, it is better to encode this transformation knowledge in an initial model transformation, i.e. the model transformation to reuse.

The level of automation that OntMT can provide highly depends on how different metamodels, DSLs, and modelling approaches are. If for example two DSL totally differ in their modelling approaches, their metamodel bindings will be two mostly unconnected sets of the reference ontology. OntMT does not add real transformation knowledge that changes the semantics of model transformation. It depends on the results that are inferable via the ontologies that are used to adjust the syntax of model transformations.

We also made scalability considerations and tests for OntMT in terms of memory requirements, runtime, and size of model transformations that can be processed. This was done for the three components of OntMT (see Section 6.3.1) separately. Memory requirements and runtime of the model manipulator rise linear to the number of rules a model transformation contains. We tested this with model transformations that contain up to 200 rules. In OntMT reasoning has only to be performed once at runtime. Its memory requirements and runtime depends on the size and the complexity of the ontologies. Implementations of other semantic web projects show that even for ontologies that are large and require quite complex reasoning results are achievable in a reasonable

time. Since the application scenarios of OntMT do not have hard real-time constraints, we do not see problems in practice concerning memory requirements, runtime, and size of model transformations for the model manipulator and the inference component.

However, the Sem-MT-Component can be seen as the 'bottleneck' of the OntMT approach. This component has to combine the reasoning results to a sensible input for the model manipulator. For this combination the size of the solution space grows exponentially with the relationships that are inferred for each concept. The size of the solution space is $c^n$, where $c$ is the number of concepts in a metamodel and $n$ is the number of relationships inferred for each concept. We try to solve this problem by restricting the solution space. As exemplified in the case study we apply heuristics that first guess an 'ideal' solution and then try to solve problems locally in the solution space, i.e. where the problems in the generated model transformation were detected.

### 6.6.3 Discussion

The OntMT approach assumes the existence of an appropriate RO. However, developing or agreeing on a RO is a non-trivial task. For example there may exist different versions of (reference) ontologies, what would transfer the problem of heterogeneous models from the MDA TS to the Ontology TS. In those cases techniques for matching and merging ontologies, like linguistic, schema-based, or probabilistic approaches, combined with human intervention have to be applied to obtain a suitable RO. Ontology alignment, matching, and mapping approaches can be also very useful to discover and define bindings from the metamodels to the RO. [172] describes an approach and a conceptual framework for mapping distributed ontologies. It can provide the basis for an interactive and incremental mapping process that is needed for developing the bindings in OntMT. In such a process the SKOS mapping vocabulary [311] could be used to specify mappings between concepts from different ontologies. For this vocabulary a search algorithm has been developed [106] that can discover potential candidates for substitutions in OntMT.

To provide ontological groundings and to find ROs may require to invest a lot of effort. Depending on the concrete application scenario, this effort may not be justifiable with the generation and evolution of model transformations. Developing or adjusting model transformations by hand may be cheaper. Hence, the goal is to reuse ROs and ontological groundings with other applications that are part of a semantic-enabled modelling and development suite (see Section 6.3.1).

A totally automated solution may also have to cope with acceptance problems of software engineers. Software engineers will probably not be willing to give up overall control of model transformation to an automated tool, which makes its choice based on metrics and heuristics. Hence, the majority of application scenarios will be of such a form, that the Sem-MT-Tool makes suggestions with a change and problem history to the software engineer. The engineer has the possibility to accept, correct, or reject the suggestions.

## 6.7 Related Work

The generation of model transformations and the evolution of models, metamodels, and transformations are classical meta data management problems. Model management is an approach to meta data management that offers a higher level programming interface [34].

It tries to solve problems of data programmability via mappings and model management operators [35, 89]. Basic operators for model management, which tackle meta data evolution scenarios, are *Match*, *Compose*, and *TransfGen*. *Match* takes two models as input and returns a mapping between them. *Compose* takes a mapping between models *A* and *B* and a mapping between models *B* and *C*, and returns a mapping between *A* and *C*. *TransfGen* takes a mapping between the models *A* and *B* and returns a new model transformation between *A* and *B*.

The *Match* operator is applied in most model management solutions and often used as a first step for data integration. Hence, it is not surprising that a multitude of approaches has been developed realizing and improving the *Match* operator. At the end of 2007, the *Publication Categorizer on Schema Evolution*[16] documented more than 415 publications in the field of schema evolution. In Section 6.7.1 we give an overview of the use and realization of the *Match* operator in the fields of Dataware and Database Management Systems TS (DBMS TS), Ontology TS, and MDA TS.

To provide solutions for model and model transformation evolution problems, approaches have to support more model management operators than the *Match* operator. In Section 6.7.2 we present and compare approaches that address model and model transformation evolution in the field of MDE.

### 6.7.1 Mapping Approaches

In most use cases, evolution of models and model transformation is triggered by the evolution of metamodels. The reason is that metamodels are the basis for model and model transformation specification. Hence, representing the changes in metamodels through mappings is an important basis for approaches that provide solutions to model and model transformation evolution, since these mappings represent a single source of information about the change in the meta data. The *Match* operator can be used to obtain these mappings.

DBMS TS is probably the research field, where data integration and meta data mapping has been studied for the longest time [2, 154, 166, 189, 190]. Schema matching is used to identify the semantic heterogeneity in order to integrate data in various database application domains. The format and semantics of data is typically specified as meta data. Semantic heterogeneity can be expressed as mappings, which specify the relationships between elements of meta data. Since mapping specification is a tedious, time-consuming, error-prone, and expensive process [247], a variety of solution has been developed to automate the matching process. [247] and [70] provide surveys about automatic schema matching in the DBMS TS.

In the Ontology TS more expressive mapping representations than in the DBMS TS have been proposed to bridge between different ontologies [82, 87, 172, 192]. These approaches have mappings as first class entities. The set of valid mapping constructs, that involve complex axioms such as equivalence and generalization, specify the relationships between concepts of two or more ontologies. Other approaches like [49, 55, 56, 109, 161] support mappings between ontologies by considering them as local domains. [269] provides an overview of approaches for schema and ontology matching.

The concept of mapping and matching meta data is also applied to the MDA TS. The use of model-based correspondences was introduced in [246]. Approaches [44, 88, 90, 325] have been developed to make mappings between metamodels available to MDE

[16] `http://se-pubs.dbs.uni-leipzig.de`

solutions as native constructs. Probably the most prominent one is the model weaving approach [44, 88].

**AMW** The ATLAS Model Weaver (AMW) tool implements the model weaving approach [44, 88]. It enables the representation of correspondences between models in so-called weaving models, from which model transformations can be generated. AMW defines a core and generic weaving model with the help of abstract concepts, which are used to specify links between metamodels, such as a *WModel*, *WLink*, and *WLinkEnd*. Since each concept is defined abstract, concrete links and their semantics have to be defined for Domain Specific Weaving Metamodels (DSWMs). [90] suggests DSWMs for the following application domains: *Composition* - Override, Merge, Delete; *Interoperability* - Equality, SourceToTarget; *Data integration* - Concatenation, Equality, IntToStr; *Traceability* - Origin, Source, Evolution, Modified, Added; *Ontology alignment* - Equivalent, Equality, Resemblance, Proximity.

**Correspondence Models** [89] develops correspondence metamodel extensions to the generic AMW metamodel. The goal is to fully capture different kinds of semantic heterogeneity between tool metamodels. The developed metamodel extensions are *similarity expressions* (equality, equivalence, non equivalence), *mapping expressions* (many-to-one, one-to-many, many-to-many), and *data value expressions*.

**Bridging DSLs** [325] proposes the use of an explicit and formal mapping model to bridge DSLs and UML. It reuses the AMW core weaving language and extends it with *class*, *attribute*, *reference*, *enumeration*, *literal*, and *data* mappings.

**Semaphore** The Model-based Semantic Mapping Framework (Semaphore) [16, 169] also allows to create mappings between (meta)models. The user of Semaphore can use a set of mapping operators (i.e. the weaving links in the AMW approach): *root mapping*, *simple mapping*, *concatenate mapping*, *split mapping*, and *substring mapping*.

### 6.7.2 Comparison of Model and Model Transformation Evolution Approaches

With the evolution of metamodels it is necessary to deal with the evolution of models, model transformations, constraints, editors, etc. in MDE. In the following we provide a comparison of approaches that deal with model and model transformation evolution. Most approaches are based on mappings that specify the delta between the old and the new metamodel. From mappings one obtains a (new) transformation that can be executed on models, that conform to the new metamodel.

#### Categorization

For the comparison we selected a set of categories. We use these categories to classify approaches supporting (meta) data evolution in the MDA TS. Most categories are derived from the work of [35, 65, 146, 247]. Whenever necessary, we introduced other categories. The categories are explained in the following list.

**Evolution Scenarios** We consider the following evolution scenarios:

- *Model Evolution:* In this scenario, a new model transformation is produced to transform models conforming to the old metamodels into models conforming to the new metamodel.
- *Model Transformation Evolution:* In this scenario, for an existing model transformation, which is specified on the basis of old metamodels, a new model transformation for the new metamodels is produced.

**Model Management Operators** This category indicates the model management operators, that are used by the compared approaches to support evolution scenarios. We consider the operators *Match*, *Compose*, and *TransfGen*.

**Mappings** Mappings are used to express correspondences between (meta)models. We distinguish model mapping approaches by the means of the following categories.

- *Instance vs. Schema:* Instance-level approaches consider instance data (i.e. data contents). In extreme cases, no schema information is given at all. Schema-level approaches only consider metamodel information, not instance data.
- *Granularity:* Mapppings can be defined for individual metamodel *elements*, such as attributes or classes[17], or for combinations of elements, such as complex metamodel *structures*.
- *Cardinality:* Mappings may relate one or more elements of one metamodel to one or more elements of another, yielding four cases *1:1*, *1:n*, *m:1*, and *m:n*.
- *Similarity:* Similarity expressions like *equality*, *equivalence*, and *non-equivalence* represent resemblance links between metamodel elements. *Equality* is used when a pair of elements represents exactly the same information. In the case of *equivalence* the linked elements contain similar information, but not exactly the same. However, the translation semantics may be the same as in equality links.
- *Constraint:* Constraints on metamodels and metamodel elements often define data types and value ranges, uniqueness, optionality, relationship types and cardinalities, etc. In constraint-based mapping approaches, constraints can be interpreted as structures, where the topology of structures as well as different element types are used to define mappings.

**MT Language** This category concerns the model transformations languages supported by the approaches, which can be used to realize and execute the evolution scenarios. *Conceptual* indicates the model transformation paradigm the approach can support in principle. The *realized* category is used to enumerate the model transformation languages for which the approach has already been implemented.

**Representation TS** This category considers the Technological Space (TS) that is used to represent *models*, *mappings*, and *model transformation.*

[17] Data value expressions differ from mapping expressions because they also evaluate the model elements, not only the metamodel elements. Data value expressions modify the source model values to make them compatible with the target model.

**Automation** This category is used to express the automation support for the *Match* operator. Automation support of (meta)model matching can be manifold, ranging from the use of linguistic methods (e.g. name matching) and constraint-based approaches to the use of heuristic, reasoning results, etc.

### Approaches for Model and Model Transformation Evolution

A variety of approaches that deal with (meta) data evolution have been developed. Since this thesis is about MDSD, we restrict our comparison to approaches that provide solutions directly applicable to evolution scenarios in the MDA TS. There also exist approaches that use similar techniques in the DBMS TS and the Ontology TS. However, these solutions cannot be applied to models conforming to different metamodels directly. The distance between the conceptual basis of the models and the implementation is too big [91]. Indeed, some of the approaches [151, 251] we discuss in the following provide additional techniques to use the results and benefits of the Ontology TS.

**Model-driven Tool Interoperability** The approach presented in [89] captures different kinds of complex mappings between tool meta data using correspondence models. It uses the correspondence models to automatically produce executable ATL transformations.

The approach for model-driven tool interoperability supports the model evolution scenario and the exchange of models. Model transformation evolution is not supported directly. This could be realized by chaining model transformations. Model-driven tool interoperability realizes the *Match* and the *TransfGen* model management operators. Mappings are defined at schema-level for elements of the metamodel. The approach allows instance level mapping for data values via so-called *DataExpressions*. The correspondence models allows to use *equality*, *equivalence*, and *non-equivalence* mappings. It is possible to specify *1:1*, *1:n*, *m:1*, and *m:n* mappings. The approach for model-driven tool interoperability is independent of any model transformation language and has been realized for ATL. Mappings, models, and model transformations are represented in the MDA TS. The approach does not provide automation support for the *Match* operator.

**Semi-automatic Model Integration** The work described in [91] presents an approach to semi-automate the development of transformations via weaving models of the model weaving approach. It describes an iterative and incremental procedure of weaving link generation, similarity calculation, and weaving link selection.

The semi-automatic model integration approach can be used to realize the model evolution scenario. For model transformation evolution this approach cannot be applied directly; one would need to apply chaining of model transformations. Semi-automatic model integration realizes the *Match* and the *TransfGen* model management operators. Mappings are defined at schema-level for elements of the metamodel. The weaving model allows to use *equality* and *equivalence* to define mappings. Semi-automatic makes use of *1:1* mappings. The approach is independent of any model transformation language and has been realized for ATL. Mappings, models, and model transformations are represented in the MDA TS. The approach provides automation support for the *Match* operator in various ways. It calculates similarity values for element mappings using linguistic matching approaches (e.g. string similarity, dictionary of synonyms, etc.) and

approaches exploiting the structure of the metamodels. Further it selects the best mapping links through link filtering and link rewriting.

**Semaphore** The Model-based Semantic Mapping Framework (Semaphore) [169, 16] supports mappings between domain models and allows the user to specify these mappings DSLs graphically. The key idea is that mappings between different information formats are used to generate the transformations, that actually perform the needed data conversion.

Semaphore supports the model evolution scenario and the exchange of models, but does not support model transformation evolution directly. Semaphore realizes the *Match* and the *TransfGen* model management operators. Mappings are defined at schema-level for elements of the metamodel. It is possible to express *equality* as well as *1:1* and *1:n* mappings. Semaphore has been implemented to produce XSLT transformation code. However, it is independent of any specific model transformation language. Mappings, models, and model transformations can be represented in various TSs. Examples have been implemented for the Structured Document TS (XML TS) and the DBMS TS. Since the approach does not provide automation support for the *Match* operator, mappings have to be specified and adjusted manually when ever changes to the meta data occur.

**Ontology-based Model Transformation** The OntMT [250, 251, 252] approach presented in this thesis lifts metamodels to an ontological level, and derives metamodel mappings from the ontology reasoning results. It interprets metamodel mappings as model transformation modification programs.

OntMT supports both aspects, the model evolution and the model transformation scenario. It realizes the *Match* and the *Compose* model management operator directly. The *TransfGen* operator is realized through a combination of a model transformation bootstrapping algorithm and the application of the *Compose* operator. Mappings are defined at schema-level for elements of the metamodel. The OntMT uses the *equality* relationship to define *1:1* and *1:n* mappings. Constraints of mappings can be defined through a value pattern mechanism. A mapping is only applied, if the value pattern matches e.g. a model transformation rule. The approach currently supports declarative transformation language and has been implemented for QVT Relations. Models are represented both in the MDA TS and the Ontology TS. Mappings and model transformations are represented in the MDA TS. OntMT provides automation support for the *Match* operator in various ways. It makes use of the reasoning result from the Ontology TS. It applies heuristics and the results of the *Compose* operator's execution to generate metamodel mappings and model transformation modification programs respectively.

**ModelCVS** The Semantic Infrastructure for Model-based Tool Integration (ModelCVS) project [151, 152, 193] provides a framework for semi-automatic generation of transformation programs. By representing modelling language concepts explicitly in ontologies, the goal is to derive bridgings between the original metamodels from the mapping between the ontologies. Concrete model transformations are derived from these bridgings (metamodel mappings).

The ModelCVS approach provides support for the model evolution scenario and the exchange of models. Model transformation evolution is not supported directly. ModelCVS realizes the *Match* and the *TransfGen* model management operators. Little can be

found about which metamodel mapping constructs are supported by the ModelCVS approach. However, the experience report in [153] indicates the kind of mapping constructs that are derived from ontology mappings in the ModelCVS approaches. According to this, mappings are defined at schema-level for elements of the metamodel. The *equality* relationship can be used to specify *1:1* mappings. The approach for model-driven tool interoperability is independent of any model transformation language and has been realized for ATL (the *TranfGen* operator has been realized as part of the *Bridging DSLs with UML* approach). Mappings and models are represented in both the MDA TS and the Ontology TS. Model transformations are represented in the MDA TS. The approach provides reasoning support to automate the *Match* operator. On ontological level further techniques from other ontology mapping and matching approaches can be integrated.

**Model Transformation Generation By-Example** The Model Transformation Generation By-Example (MTBE) approach [326] makes use of inter-model mappings representing semantic correspondences between concrete domain models. The inter-model mappings between domain models can be used to generate the model transformation rules, by-example, taking into account the already defined mapping between abstract and concrete syntax elements.

MTBE supports the model evolution scenario and the exchange of models, but does not support model transformation evolution directly. It realizes the *Match* and the *TransfGen* model management operators. Mappings are defined at schema-level for elements of the metamodel and at instance-level for the model elements. The approach allows to use the *equality* relationship to define *1:1* mappings. MTBE supports constraint mappings by so-called *conditional mappings*, where the mapping of elements depends on the values of their attributes. The approach is independent of any model transformation language and has been realized for ATL. Mappings, models, and model transformations are represented in the MDA TS. The approach provides some automation support for the *Match* operator by generating metamodel model mappings from sample model mappings.

**Semi-automatic Approach for Bridging DSLs with UML** [325] presents a semi-automatic approach for bridging DSLs with UML. The approach is based on the manual mapping of domain-specific metamodels and UML and the automatic generation of UML profiles and model transformations.

This bridging approach provides support for the model evolution scenario and the exchange of models. Model transformation evolution is not supported directly. It realizes the *Match* and the *TransfGen* model management operators. The bridging language extends the AMW weaving metamodel. Mappings are defined at schema-level for elements of the metamodel. The approach allows to use the *equality* relationship to define *1:1* mappings. The approach is independent of any model transformation language and has been realized for ATL. Mappings, models, and model transformations are represented in the MDA TS. The approach does not provide automation support for the *Match* operator.

**Model Typing** In [279], the authors introduce model typing as extension of object-oriented typing and propose an algorithm for checking the conformance of model types. It is presented, how model typing permits more flexible reuse of model transformations

across various metamodels while preserving type safety.

This approach improves the reuse of models and model transformations whenever small changes to metamodels occur, like altering the cardinality of an association. In case of major change in models' representation formats, model transformations still have to be modified manually. This stems mainly from the fact that this approach does not support the *Match* or any other model management operator. The approach does not improve (meta) data evolution directly through the manipulation of the this data. Evolution is supported by extending the set of valid input and output models of a model transformations or MDE tools, by comparing not the metamodels but the type of the models; i.e. in evolution scenarios the old model (transformation) is the new model (transformation). Models and model transformations are represented in the MDA TS.

#### Comparison Overview

Table 6.11 provides an overview of the approaches supporting evolution scenarios in the MDA TS. The table uses the categories we presented above for the classification of the discussed approaches. The following notes provide further clarification about the comparison table:

*a)* This approach supports the model transformation scenario only indirectly. The scenario could be realized by chaining model transformations. Depending on the language of the model transformations, which are generated from the mapping specification, chaining has an impact on other model transformation features. For example traceability of the new chained model transformation execution may be difficult to achieve.

*b)* This approach realizes the *TransfGen* operator through a combination of model transformation bootstrapping and the application of the *Compose* operator.

*c)* This approach does not generate or modify model transformations. The old models and model transformations are also the new ones.

## 6.8 Conclusions

In this chapter we presented a variety of contributions that address interoperability and evolution problems in MDSD. The most important ones are: We developed the OntMT approach, which provides means to automatically deal with model and model transformation evolution scenarios. We introduced an architecture for a semantic-enabled modelling and development suite, which provides the basis to support developers and modellers in a sophisticated manner by making use of reasoning results. We described concepts and techniques to realize and implement the OntMT approach. We developed a higher-order model transformation language that allows to modify model transformation and lends itself for automating reuse of model transformations. We showed how the semantics of the higher-order model transformation language and the reasoning results can be used to automate the model management operator *Match* and compute (meta)model mappings.

The approach of OntMT provides technology that fosters interoperability in model exchange and the evolution of model transformations. It integrates ontologies in MDSD

| Category | | Model-driven Tool Interoperability | Semi-autom. Model Integration | Semaphore | OntMT | ModelCVS | MTBE | Model Typing |
|---|---|---|---|---|---|---|---|---|
| **Evolution scenarios** | *Models* | yes | yes | yes | **yes** | yes | yes | yes[c] |
| | *MTs* | no[a] | no[a] | no[a] | **yes** | no[a] | no[a] | yes[c] |
| **Model Mgmt. Operators** | *Match* | yes | yes | yes | **yes** | yes | yes | n/a |
| | *Compose* | no | no | no | **yes** | no | no | n/a |
| | *TransfGen* | yes | yes | yes | **yes**[b] | yes | yes | n/a |
| **Matching** | *Inst. vs. Schema* | schema, instance | schema | schema | **schema** | schema | instance, schema | n/a |
| | *Granularity* | element | element | element | **element** | element | element | n/a |
| | *Cardinality* | 1:1, 1:n, m:1, m:n | 1:1 | 1:1, 1:n | **1:1, 1:n** | 1:1 | 1:1 | n/a |
| | *Similarity* | equality, equivalence, non-equiv. | equality, equivalence | equality | **equality** | equality | equality | n/a |
| | *Constraint* | no | no | no | **value patterns** | no | conditions | n/a |
| **MT language** | *Conceptual* | lang. indep. | lang. indep. | lang. indep. | **declarative** | lang. indep. | lang. indep. | n/a |
| | *Realized* | ATL | ATL | XSLT | **QVT Relations** | ATL | ATL | n/a |
| **Represen-tation TS** | *Mappings* | MDA | MDA | TS indep. | **MDA, Ontology** | MDA, Ontology | MDA | n/a |
| | *Models* | MDA | MDA | TS indep. | **MDA** | MDA, Ontology | MDA | MDA |
| | *MTs* | MDA | MDA | TS indep. | **MDA** | MDA | MDA | MDA |
| **Automation** | | no | simil. calc., filtering, rewriting | no | **reasoning, heuristics** | reasoning | examples | n/a |

Table 6.11: Comparison of approaches supporting MDE evolution scenarios

and makes use of the reasoning capabilities of the Ontology TS. By automated generation of mappings it offers new possibilities for the integration of domain-specific languages and 'legacy' models in a plug&play manner. This makes it easier for new organisations to join collaborations. OntMT also supports organisations evolving their modelling techniques like using new and more advanced versions of modelling languages. It yields more efficient reuse of model transformations and the knowledge that is captured in those transformations. Nevertheless, OntMT uses additional information, which has to be provided by the experts developing metamodels and domain-specific languages.

# Chapter 7

# Conclusions

In this thesis we investigated methods and techniques to improve the practical application of MDSD for efficient and effective CBP enactment. Three problems were identified: the lack of CBP modelling and enactment infrastructures, the absence of adequate methods to select ICT architectures in MDSD, and the evolution of meta data and their dependent artifacts in MDSD, i.e. models and transformations.

## 7.1 Summary

With the overall goal of a broader dissemination and easier adoption of MDSD, we developed solutions that address these problems.

#### CBP Modelling and Enactment Infrastructure

We developed frameworks, tools, and model transformations that facilitate the generation of executable code from high-level, domain-specific models. Thereby, we presented software architecture patterns that enable ICT system coordination in a service-oriented environment. We implemented a set of model transformations, that are based on software architecture patterns and allow automated generation of PIMs from computational independent CBP models. We developed a model and code generation framework that facilitates the generation of executable workflow code from higher-level process models (e.g. PIMs and CIMs) on the basis of the input from modelling and domain experts.

This contribution bridges the semantic gap between domain-specific, high-level business process descriptions and the technologies used to implement them in service-oriented ICT systems. Both domain and IT experts benefit since our contribution provides a set of sensible and customizable transformations they can reuse to improve the development and adjustment of their ICT systems. These experts do not have to develop similar transformations again, what saves them a time consuming, error-prone, and often tedious task. Further, the focus of software development is shifted away from the underlying IT technology towards the ideas and concepts of the problem domain.

#### ICT Architecture Selection

We investigated new decision methods and guidelines for the selection of appropriate ICT architectures. We developed a model for decision support that helps IT architects to derive an appropriate architecture paradigm for a given use case or application domain.

The decision model combines the AHP with scenario-based architecture evaluation techniques. Hence, we further presented scenario descriptions that allow the evaluation and selection of appropriate ICT system coordination architecture paradigms for CBP enactment. We assist IT architects by providing a set of guidelines of how contingencies influence ICT system coordination architecture.

Our decision method, the scenario descriptions, and the set of guidelines help IT architects to select and reuse appropriate ICT system coordination architectures for CBP enactment in a timely manner. This helps to gain productivity wins by reducing the development time and improving the quality of development with existing and tested solutions. Without sufficient decision support people would start redeveloping the solutions themselves.

#### Model and Transformation Evolution

We developed the OntMT approach that facilitates the technical aspects of exchanging models between different enterprises and the reuse and evolution of model transformations. The OntMT approach provides means to automatically deal with model and model transformation evolution scenarios. We described an architecture for a semantic-enabled modelling and development suite. We implemented OntMT as such a semantic-enabled tool and applied OntMT to two real world case studies. We further developed a higher-order model transformation language, that realizes the model management *Compose* operator and lends itself for automating the reuse of model transformations via the modification of model transformations. The OntMT approach is the first appoach that supports the *Compose* operator for the MDA TS (cp. Table 6.11).

OntMT enables organisations to exchange models as well as to reuse existing models and model transformations despite of interoperability challenges. Differences in syntax and semantics of modelling formalisms, that arise from the use of DSLs and the constant evolution of models and modelling languages, can be overcome with as little effort as possible. Organisations and modellers benefit from OntMT since it allows them to exchange, customize, and evolve models and model transformations more efficiently.

## 7.2 Discussion and Outlook

Though MDSD yields advances in the efficiency and the effectiveness of software engineering, it does not provide magic buttons and is not a silver bullet [52, 96] that reduces essential complexity in software development. MDSD reduces development time and improves software quality by providing higher-level abstractions that fit the needs of the developers and improve the communication with the users. However, it is still necessary to specify requirements that unambiguously define the problem as the requirements fidelity at the higher abstraction level determines the potential quality of the systems to develop.

In this thesis we took the usefulness of MDSD as granted and put our focus on methods and techniques to improve the practical application of MDSD for efficient and effective CBP enactment. We developed contributions to three main challenges as described above. There still exist other challenges that have to be addressed, like the support of people involved in model-driven development projects through appropriate guidelines and methodologies. Some of our results can be applied to broader problem areas than the design and enactment of CBPs and the case studies presented in this thesis.

### CBP Modelling and Enactment Infrastructure

For the CIM to PIM transformation we developed software architecture patterns for ICT system generation that can be applied to enact CBPs. We implemented these patterns in the CIM to PIM transformations that allow automated generation of service-oriented, platform independent models (PIM4SOA) from computational independent CBP models. Our solution stands out from other solutions presented under the MDSD label. It not only converts the representation format of cross-organisational business process from EPC to PIM4SOA and leaves the implementation of the software architecture as a manual task to the system architect. Our solutions narrow the gap between high-level business models and ICT system models significantly by encoding software architecture patterns in the model transformations. Having implemented and applied such model transformations for the central, the decentral, and the brokerless approach, we gained insights how CBPs should be ideally modelled in CIMs. We found it important that CBPs are explicitly modelled; otherwise, model transformation results are likely to be of poor quality. The decentral broker architecture relies on the existence of a CBP model to a higher degree than the central broker architecture and the brokerless architecture: the latter can be derived more easily from the process flow; in the former, the appropriate grouping of processes to view processes in a decentral broker must be specified explicitly.

In the context of executable WS-BPEL workflow code, there exist a variety of solutions that provide hardly more than another concrete syntax (graphical instead of textual) for WS-BPEL. These solutions do not narrow the gap between higher-level process descriptions and workflow execution. Tool chains that allow model-driven development and the generation of WS-BPEL code still have restrictions that prevent all process models from being fully transformed. The model and code generation framework presented in this thesis provides a solution that helps to efficiently bridge the gap between higher-level process descriptions and executable workflow code. It allows to achieve time savings compared to other approaches. However, these time savings depend on the complexity of the input graphs (e.g. with or without cycles) and on the experience of the developer. Our model and code generation framework makes it possible for people with no or little experience in code generation and graph transformation to produce workflow code at reasonable time. Future work is to apply the model and code generation framework to other languages like XPDL or OWL-S code.

### ICT Architecture Selection

The evaluation and decision support method enables people to get a better understanding of the influence of contingencies on the overall decisions. Existing approaches do not take into account how the various aspects of organisations' environment influence the measurement and the final decision of the evaluation process. Our experience so far indicates that pairwise comparisons reduce the amount of information that is necessary for decisions. Since people can only deal with information involving simultaneously a small number of facts, pairwise comparisons help evaluators to make better judgements compared to methods where more information needs to be considered. Though pairwise comparisons require more complex calculations than other rating approaches, they promise to provide more exact results.

Future work is the question how decision methods as the one developed in this thesis can be built into existing enterprise modelling frameworks and model-driven IDEs,

to support process modelers and ICT architects in their task of creating and managing executable CBP specifications from business level models. A second area concerns the specification of further scenarios to extend the scope of the evaluation and decision method's application. More fine-grained models and extensions of our decision method need to be developed to support the process down to the platform-specific and code levels.

### Model and Transformation Evolution

The OntMT approach facilitates the exchange of models between different enterprises as well as the reuse and evolution of model transformations. It assumes the existence of an appropriate Reference Ontology (RO). However, developing or agreeing on a RO is a non-trivial task. For example there may exist different versions of (reference) ontologies, what would transfer the problem of heterogeneous models from the MDA TS to the Ontology TS. In those cases techniques for matching and merging ontologies, like linguistic, schema-based, or probabilistic approaches, combined with human intervention have to be applied to obtain a suitable RO. Ontology alignment, matching, and mapping approaches can also be very useful to discover and define bindings from the metamodels to the RO. To provide ontological groundings and to find ROs may require to invest a lot of effort. Hence, the goal is to reuse ROs and ontological groundings with other applications that are part of a semantic-enabled modelling and development suite. Further, a totally automated solution may also have to cope with acceptance problems of software engineers. Software engineers will probably not be willing to give up overall control of model transformations to an automated tool, which makes its choice based on metrics and heuristics. Hence, the majority of application scenarios will be of such a form, that the Sem-MT-Tool makes suggestions with a change and problem history to the software engineer. The engineer has the possibility to accept, correct, or reject the suggestions.

Future research is to apply the techniques of the higher-order transformation language also to imperative model transformation languages. Another task is to extend the correlation algorithm of the Sem-MT-Component for a more sophisticated support of the *Match* model management operator. It further seems promising to apply the concepts developed for the OntMT approach to other meta data evolution problems. From a today's point of view, it will be a great challenge to develop similar techniques for constraint languages like OCL. The techniques of the higher-order transformation language can also be applied to the evolution of editors, which realize a mapping of the abstract syntax to concrete visualizations.

# Bibliography


[1] Model-driven Interoperability. http://www.modelbased.net/mdi/index.html, February 2007.

[2] Serge Abiteboul, Sophie Cluet, and Tova Milo. Correspondence and Translation for Heterogeneous Data. In *Proceedings of the 6th International Conference on Database Theory (ICDT)*, volume 1186, pages 351–363. Springer, January 1997.

[3] AGILE. AGILE project: Agile Software Development of Embedded Systems. http://www.agile-itea.org.

[4] AgilPro. AgiPro project: Agile Business Processes With Service Enabled Applications. http://www.agilpro.eu/.

[5] Aditya Agrawal, Gabor Karsai, and Feng Shi. Graph Transformations on Domain-Specific Models. Technical Report ISIS-03-403, Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA, November 2003.

[6] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison Wesley, 1986.

[7] David H. Akehurst, W. Gareth Howells, and Klaus D. McDonald-Maier. Kent Model Transformation Language. In *Model Transformations in Practice Workshop, MoDELS Conference, Montego Bay, Jamaica*, October 2005.

[8] David H. Akehurst and Stuart Kent. A Relational Approach to Defining Transformations in a Metamodel. In *5th International Conference on The Unified Modeling Language (UML), Dresden, Germany*, volume 2460 of *Lecture Notes in Computer Science*, pages 243–258. Springer-Verlag, September/October 2002.

[9] Scott W. Ambler. Agile Modeling (AM) Home Page: Effective Practices for Modeling and Documentation. http://www.agilemodeling.com.

[10] AMPLE. AMPLE project: Aspect-Oriented, Model-driven Product Line Engineering. http://www.ample-project.net/.

[11] Jim Amsden, Tracy Gardner, Catherine Griffin, and Sridhar Iyengar. Draft UML 1.4 profile for automated business processes with a mapping to BPEL 1.0, version 1.1. *IBM developerworks*, June 2003.

[12] Marc Andries, Gregor Engels, Annegret Habel, Berthold Hoffmann, Hans-Jörg Kreowski, Sabine Kuske, Detlef Plump, Andy Schürr, and Gabriele Taentzer. Graph Transformation for Specification and Programming. *Science of Computer Programming*, 34(1):1–54, April 1999.

[13] AndroMDA. AndroMDA. http://www.andromda.org/.

[14] Assaf Arkin. Business Process Modeling Language (BPML). BPMI.org, November 2002.

[15] ATHENA IP. ATHENA IP European Project: Advanced Technologies for interoperability of Heterogeneous Enterprise Networks and their Applications. http://www.athena-ip.org.

[16] ATHENA IP. Model-based Semantic Mapping Framework (Semaphore). http://modelbased.net/semaphore/.

[17] Colin Atkinson and Thomas Kühne. Model-Driven Development: A Metamodeling Foundation. *IEEE Software*, 20(5):36–41, September 2003.

[18] ATLAS group. KM3: Kernel MetaMetaModel - Manual version 0.3, August 2005.

[19] AToM$^3$. AToM$^3$: A Tool for Multi-Paradigm modeling. http://atom3.cs.mcgill.ca/.

[20] Ralph-Johan Back. On Correct Refinement of Programs. *Journal of Computer and System Sciences*, 23(1):49–68, August 1981.

[21] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley, 2003.

[22] Len Bass and Bonnie E. John. Linking Usability to Software Architecture Patterns Through General Scenarios. *Journal of Systems and Software*, 66(3):187–197, June 2003.

[23] Bernhard Bauer, Florian Lautenbacher, Stephan Roser, Wolf Fischer, and Viviane Schöbel. SPL4AOX – Eine Modellierungssprache als Service-Profil für den MID innovatorAOX 2006 sowie Transformationen nach WSDL und BPEL4WS. May 2006.

[24] Bernhard Bauer, Jörg P. Müller, and Stephan Roser. A Model-Driven Approach to Designing Cross-Enterprise Business Processes. In *On the Move to Meaningful Internet Systems 2004: OTM 2004 Workshops: OTM Confederated International Workshops and Posters, GADA, JTRES, MIOS, WORM, WOSE, PhDS, and INTEROP 2004, Agia Napa, Cyprus, October 25-29, 2004. Proceedings*, volume 3292 of *Lecture Notes in Computer Science*, pages 544–555. Springer, October 2004.

[25] Bernhard Bauer, Jörg P. Müller, and Stephan Roser. Adaptive Design of Cross-Organizational Business Processes Using a Model-Driven Architecture. In *Wirtschaftsinformatik 2005: eEconomy, eGovernment, eSociety, 7. Internationale Tagung Wirtschaftsinformatik 2005, Bamberg, Germany*, pages 103–122. Physica-Verlag, February 2005.

[26] Bernhard Bauer, Jörg P. Müller, and Stephan Roser. A Decentralized Broker Architecture for Collaborative Business Process Modelling and Enactment. In *Enterprise Interoperability - New Challenges and Approaches (I-ESA'06)*, pages 115–126. Springer, April 2006.

[27] Bernhard Bauer, Jörg P. Müller, and Stephan Roser. Model- and Architecture-Driven Development in the Context of Cross-Enterprise Business Process Engineering. In *Proceedings of the IEEE International Conference on Services Computing (SCC'06), Chicago, USA*, pages 119–126. IEEE Computer Society, September 2006.

[28] Bernhard Bauer, Günther Palfinger, Florian Lautenbacher, and Stephan Roser. "AgilPro": Modellierung, Simulation und Ausführung agiler Prozesse. *Objekt Spektrum*, (1):52–59, January/February 2007.

[29] Bernhard Bauer and Stephan Roser. Semantic-enabled Software Engineering and Development. In *INFORMATIK 2006 - Informatik für Menschen Band 2, 1st International Workshop on Applications of Semantic Technologies, AST 2006, Dresden, Germany*, volume P-94 of *Lecture Notes in Informatics*, pages 293–296. Bonner Köllen, October 2006.

[30] Gorka Benguria, Xabier Larrucea, Brian Elvesæter, Tor Neple, Anthony Beardsmore, and Michael Friess. A Platform Independent Model for Service Oriented Architectures. In *Enterprise Interoperability: New Challenges and Approaches, Proceedings Of the 2nd International Conference on Interoperability of Enterprise Systems and Architecture (I-ESA'2006), Bordeaux, France*, pages 23–34. Springer, April 2007.

[31] Douglas W. Bennett. *Designing Hard Software - The Essential Task*. Prentice Hall, 1997.

[32] Tim Berners-Lee and Mark Fischetti. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*. Harper Collins, 1 edition, September 1999.

[33] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, May 2001.

[34] Philip A. Bernstein. Applying Model Management to Classical Meta-Data Problems. In *First Biennial Conference on Innovative Data Systems Research (CIDR), Asilomar, CA, USA*, pages 209–220, January 2003.

[35] Philip A. Bernstein and Sergey Melnik. Model Management 2.0: Manipulating Richer Mappings. In *Proceedings of the ACM SIGMOD international conference on Management of data (SIGMOD), Beijing, China*, pages 1–12. ACM, June 2007.

[36] Jean Bézivin. Model Engineering: from Principles to Platforms, March 2005.

[37] Jean Bézivin. On the unification power of models. *Software and System Modeling*, 4(2):171–188, May 2005.

[38] Jean Bézivin, Fabian Büttner, Martin Gogolla, Frederic Jouault, Ivan Kurtev, and Arne Lindow. Model Transformations? Transformation Models! In *Proceedings of Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006, Genova, Italy*, volume 4199 of *Lecture Notes in Computer Science*, pages 440–453. Springer, October 2006.

[39] Jean Bézivin, Vladan Devedžić, Dragan Djurić, Jean-Marie Favreau, Dragan Gašević, and Frédéric Jouault. An M3-Neutral Infrastructure for Bridging Model Engineering and Ontology Engineering. In *Interoperability of Enterprise Software and Applications (INTEROP-ESA'05)*, pages 159–171, 2005.

[40] Jean Bézivin, Grégoire Dupé, Frédéric Jouault, and Jamal Eddine Rougui. First experiments with the ATL model transformation language: Transforming XSLT into XQuery. In *OOPSLA Workshop on Generative Techniques in the Context of the MDA*, 2003.

[41] Jean Bézivin and Olivier Gerbé. Towards a Precise Definition of the OMG/MDA Framework. In *16th IEEE international conference on Automated software engineering (ASE), San Diego, USA*, pages 273–280. IEEE Computer Society, November 2001.

[42] Jean Bézivin, Guillaume Hillairet, Frédéric Jouault, Ivan Kurtev, and William Piers. Bridging the MS/DSL Tools and the Eclipse Modeling Framework. In *International OOPSLA'05 Workshop on Software Factories at, San Diego, California, USA*, October 2005.

[43] Jean Bézivin, Frédéric Jouault, Peter Rosenthal, and Patrick Valduriez. Modeling in the Large and Modeling in the Small. In *European MDA Workshops: Foundations and Applications, MDAFA 2003 and MDAFA 2004, Twente, The Netherlands, and Linköping, Sweden*, volume 3599 of *Lecture Notes in Computer Science*, pages 33–46. Springer, June 2003/04.

[44] Jean Bézivin, Frédéric Jouault, and Patrick Valduriez. First Experiments with a ModelWeaver. In *Proceedings of the OOPSLA GPCE Workshop on Best Practices for Model Driven Software Development Workshop, Vancouver, Canada*, October 2004.

[45] Alex Birman and John Ritsko. Preface to Service-Oriented Architecture. *IBM Systems Journal*, 44(4):651–652, 2005.

[46] Grady Booch, Alan Brown, Sridhar Iyengar, James Rumbaugh, and Bran Selic. An MDA Manifesto. *MDA Journal*, May 2004.

[47] Stefano Borgo, Aldo Gangemi, Nicola Guarino, Claudio Masolo, and Alessandro Oltramari. OntologyRoadMap. WonderWeb Deliverable D15. `http://wonderweb.semanticweb.org`, December 2002.

[48] Adam Bosworth. Data Routing Rather than Databases: The Meaning of the Next Wave of the Web Revolution to Data Management. In *28th international conference on Very Large Data Bases (VLDB), Hong Kong, China*, page .1. Morgan Kaufmann, August 2002.

[49] Paolo Bouquet, Fausto Giunchiglia, Frank van Harmelen, Luciano Serafini, and Heiner Stuckenschmidt. C-OWL: Contextualizing Ontologies. In *2nd International Semantic Web Conference (ISWC), Florida, USA*, volume 2870 of *Lecture Notes in Computer Science*, pages 164–179. Springer, October 2003.

[50] Peter Braun and Frank Marschall. The Bi-directional Object-Oriented Transformation Language. Technical Report TUM-I0307, Technische Universität München, München, Germany, May 2003.

[51] Saartje Brockmanns and Peter Haase. A Metamodel and UML Profile for Networked Ontologies. In *2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE), Athens, GA, USA*, November 2006.

[52] Jr. Frederick P. Brooks. No Silver Bullet: Essence and Accidents of Software Engineering. *Computer*, 20(4):10–19, April 1987.

[53] Phillip A. Brown and David F. Gibson. A quantified model for facility site selection application to multiplant location problem. *AIIE transactions: industrial engineering research and development*, 4(1):1–10, 1972.

[54] R. Ian Bull and Jean-Marie Favre. Visualization in the Context of Model Driven Engineering. In *Workshop on Model Driven Development of Advanced User Interfaces (MDDAUI) at MoDELS Conference, Montego Bay, Jamaica*, volume 159 of *CEUR Workshop Proceedings*. CEUR-WS.org, October 2005.

[55] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. A Framework for Ontology Integration. In *First Semantic Web Working Symposium, Stanford, USA*, Frontiers in artificial intelligence and applications, pages 303–316. IOS Press, 2002.

[56] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Description Logics for Information Integration. In *Computational Logic: Logic Programming and Beyond*, volume 2408 of *Lecture Notes in Computer Science*, pages 41–60. Springer, 2002.

[57] Paul Clements, Rick Kazman, and Mark Klein. *Evaluating Software Architecture*. Addison-Wesley, 2002.

[58] Edgar F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6):377–387, June 1970.

[59] Edgar F. Codd. *The Relational Model for Database Management: Version 2*. ACM Classic Books Series. Addison-Wesley Longman Publishing Co., Inc., 1990.

[60] Compuware. OptimalJ, Model-driven development for Java. `http://www.compuware.com/products/optimalj/`.

[61] Steve Cook. Domain-Specific Modeling and Model Driven Architecture, January 2004.

[62] Microsoft Corporation. Visual Studio 2005 Team System Modeling Strategy and FAQ. Visual Studio 2005 Technical Articles, May 2005.

[63] Krzysztof Czarnecki and Michal Antkiewicz. Mapping Features to Models: A Template Approach Based on Superimposed Variants. In *4th International Conference on Generative Programming and Component Engineering, Tallinn, Estonia*, volume 3676 of *Lecture Notes in Computer Science*, pages 422–437. Springer, September/October 2005.

[64] Krzysztof Czarnecki and Simon Helsen. Classification of Model Transformation Approaches. In *Proceedings Of the OOPSLA'03 Workshop on the Generative Techniques in the Context Of Model-Driven Architecture, Anaheim, California, USA*, 2003.

[65] Krzysztof Czarnecki and Simon Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3):621–645, July 2006.

[66] Paulo Pinheiro da Silva, Deborah L. McGuinness, and Richard Fikes. A proof markup language for Semantic Web services. *Information Systems*, 31(4):381–395, June/July 2006.

[67] Juan de Lara and Hans Vangheluwe. AToM: A Tool for Multi-Formalism and Meta-Modeling. In *5th International Conference on Fundamental Approaches to Software Engineering (FASE), Grenoble, France*, volume 2306 of *Lecture Notes in Computer Science*, pages 174–188. Springer, April 2002.

[68] Henry Lieberman Dieter Fensel, James A. Hendler and Wolfgang Wahlster, editors. *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. The MIT Press, 2003.

[69] Dragan Djurić, Dragan Gašević, and Vladan Devedžić. Ontology Modeling and MDA. *Journal of Object Technology*, 4(1):109–128, January-February 2005.

[70] AnHai Doan and Alon Y. Halevy. Semantic Integration Research in the Database Community: A Brief Survey. *AI Magazine, Special Issue on Semantic Integration*, 26(1):83–94, March 2005.

[71] Thomas J. Dolan. *Architecture assessment of Information-System Families*. PhD thesis, Technische Universiteit Eindhoven, 2001.

[72] Lex Donaldson. *The Contingency Theory of Organizations*. SAGE Publications, Inc, February 2001.

[73] Eclipse Project. ATLAS MegaModel Management (AM3). `http://www.eclipse.org/gmt/am3/`.

[74] Eclipse Project. ATLAS Model Weaver (AMW). `http://www.eclipse.org/gmt/amw/`.

[75] Eclipse Project. Eclipse Modeling Framework (EMF). `http://www.eclipse.org/modeling/emf/`.

[76] Eclipse Project. Eclipse Validation Framework. `http://www.eclipse.org/modeling/emf/?project=validation#validation`.

[77] Eclipse Project. EMF Ontology Definition Metamodel (EODM). `http://www.eclipse.org/modeling/mdt/?project=eodm`.

[78] Eclipse Project. Generative Modeling Tools (GMT). `http://www.eclipse.org/gmt/`.

[79] Eclipse Project. Java Emitter Templates (JET). `http://www.eclipse.org/modeling/m2t/?project=jet#jet`.

[80] Eclipse Project. Model Driven Development integration (MDDi). `http://www.eclipse.org/mddi/`.

[81] Sven Efftinge, Peter Friese, Arno Haase, Clemens Kadura, Bernd Kolb, Dieter Moroff, Karsten Thoms, and Markus Völter. openArchitectureWare User Guide, Version 4.2, September 2007.

[82] Marc Ehrig, Peter Haase, Mark Hefke, and Nenad Stojanovic. Similarity for Ontologies - A Comprehensive Framework. In *13th European Conference on Information Systems, Regensburg, Germany*, May 2005.

[83] Brian Elvesæter, Axel Hahn, Arne-Jørgen Berre, and Tor Neple. Towards an Interoperability Framework for Model-Driven Development of Software Systems. In *Interoperability of Enterprise Software and Applications, Proceedings of the 1st International Conference on Interoperability of Enterprise Systems and Architecture (I-ESA'2005), Geneva, Switzerland*, pages 409–420. Springer, 2005.

[84] Thomas Erl. *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. Prentice Hall International, 2004.

[85] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall International, 2005.

[86] European Commission. Third Status Report on European Technology Platforms: At the Launch of FP7, March 20007.

[87] Jérôme Euzenat. An API for Ontology Alignment. In *3rd International Semantic Web Conference, Hiroshima, Japan*, volume 3298 of *Lecture Notes in Computer Science*, pages 698–712. Springer, November 2004.

[88] Marcos Didonet Del Fabro, Jean Bézivin, Frédéric Jouault, Erwan Breton, and Guillaume Gueltas. AMW: A Generic Model Weaver. In *1ères Journées sur l'Ingénierie Dirigée par les Modèles*, 2005.

[89] Marcos Didonet Del Fabro, Jean Bézivin, Frédéric Jouault, and Patrick Valduriez. Model-Driven Tool Interoperability: An Application in Bug Tracking. In *5th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE), Montpellier, France*, volume 4275 of *Lecture Notes in Computer Science*, pages 863–881. Springer, October 2006.

[90] Marcos Didonet Del Fabro, Jean Bézivin, and Patrick Valduriez. Weaving Models with the Eclipse AMW plugin. In *Eclipse Modeling Symposium, Eclipse Summit Europe 2006, Esslingen, Germany*, 2006.

[91] Marcos Didonet Del Fabro and Patrick Valduriez. Semi-automatic Model Integration using Matching Transformations and Weaving Models. In *22nd Annual ACM symposium on Applied computing (SAC) - Model Transformation Track, Seoul, Korea*, pages 963–970. ACM, 2007.

[92] Jean-Marie Favre. CacOphoNy: Metamodel-Driven Architecture Reconstruction. In *11th Working Conference on Reverse Engineering (WCRE), Delft, The Netherlands*, pages 204–213. IEEE Computer Society, November 2004.

[93] Jean-Marie Favre. Foundations of Meta-Pyramids: Languages vs. Metamodels – Episode ii: Story of Thotus the Baboon1. In Jean Bezivin and Reiko Heckel, editors, *Language Engineering for Model-Driven Software Development*, number 04101 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2005. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany.

[94] Jean-Marie Favre. Foundations of Model (Driven) (Reverse) Engineering : Models – Episode i: Stories of The Fidus Papyrus and of The Solarus. In Jean Bezivin and Reiko Heckel, editors, *Language Engineering for Model-Driven Software Development*, number 04101 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2005. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany.

[95] Wolf Fischer and Stephan Roser. Inference for an Application of Semantic-enabled Software Development. 2008, forthcoming.

[96] David Flater. Impact of Model-Driven Standards. In *35th Annual Hawaii International Conference on System Sciences (HICSS)*, volume 9, pages 3706–3714. IEEE Computer Society, January 2002.

[97] Franck Fleurey, Zoé Drey, Didier Vojtisek, Cyril Faucher, and Vincent Mahé. Kermeta language – Reference manual, June 2007.

[98] Joachim H. Frank, Tracy A. Gardner, and Simon K. Johnston. Business Process Definition Metamodel - Concepts and Overview, April 2004.

[99] David S. Frankel. *Model Driven Architecture - Applying MDA™ to Enterprise Computing*. John Wiley & Sons, Inc., New York, NY, USA, 2003.

[100] Fujaba. Fujaba Tool Suite. `http://www.fujaba.de/`.

[101] Erich Gamma, Richard Helm, and Ralph E. Johnson. *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley Longman, 1995.

[102] Andrés García-Camino, Pablo Noriega, and Juan-Antonio Rodríguez-Aguilar. Implementing Norms in Electronic Institutions. In *Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), Utrecht, The Nederlands*, pages 667–673. ACM Press, July 2005.

[103] Tracy Gardner, Catherine Griffin, Jana Koehler, and Rainer Hauser. A review of OMG MOF 2.0 Query / Views / Transformations Submissions and Recommendations towards the final Standard. In *1st MetaModelling for MDA Workshop, Kings Manor, York, England*, pages 178–197, November 2003.

[104] Tracy Gardner and Larry Yusuf. Explore model-driven development (MDD) and related approaches: A closer look at model-driven development and other industry initiatives. *IBM developerworks*, March 2006.

[105] Dragan Gašević, Dragan Djurić, and Vladan Devedžić. *Model Driven Architecture and Ontology Development*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[106] Dragan Gašević and Marek Hatala. Searching Web Resources Using Ontology Mappings. In *K-CAP 2005 Workshop on Integrating Ontologies, Banff, Canada*, volume 156 of *CEUR Workshop Proceedings*. CEUR-WS.org, October 2005.

[107] Anna Gerber, Michael Lawley, Kerry Raymond, Jim Steel, and Andrew Wood. Transformation: The Missing Link of MDA. In *Proceedings of Graph Transformation: First International Conference (ICGT'02) Barcelona, Spain*, volume 2505 of *Lecture Notes in Computer Science*, pages 90–105. Springer, October 2002.

[108] Parviz Ghandforoush, Philip Y. Huang, and Bernard W. Taylor. A mulit-criteria decision model for the selection of a computerized manufacturing control system. *International Journal of Production Research*, 23(1):117–128, 1985.

[109] Chiara Ghidini and Luciano Serafini. Distributed First Order Logic - Revised Semantics. Technical report, ITC-irst, January 2005.

[110] Interactive Objects Software GmbH. ArcStyler. `http://www.arcstyler.com`.

[111] Asunción Gómez-Pérez and Oscar Corcho. Ontology Languages for the Semantic Web. *IEEE Intelligent Systems*, 17(1):54–60, January/February 2002.

[112] Mathias Götz, Stephan Roser, Florian Lautenbacher, and Bernhard Bauer. Using Token Analysis to Transform Graph-Oriented Process Models to BPEL. 2008, forthcoming.

[113] Jeff Gray, Yuehua Lin, and Jing Zhang. Automating Change Evolution in Model-Driven Engineering. *IEEE Computer (Special issue on Model-Driven Engineering)*, 39(2):51–58, Februaber 2006.

[114] Jack Greenfield. Bare-Naked Languages or What Not to Model. *The Architecture Journal*, October 2006.

[115] Jack Greenfield and Keith Short. Software factories: assembling applications with patterns, models, frameworks and tools. In *OOPSLA'03: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 16–27, New York, NY, USA, 2003. ACM Press.

[116] Jack Greenfield, Keith Short, Steve Cook, and Stuart Kent. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley Publishing Inc., 2004.

[117] Ulrike Greiner, Sonia Lippe, Timo Kahl, Jörg Ziemann, and Frank-Walter Jäkel. Designing and Implementing Cross-Organizational Business Processes - Description and Application of a Modelling Framework. In *Enterprise Interoperability - New Challenges and Approaches (I-ESA'06)*, pages 137–147. Springer, April 2006.

[118] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43(5-6):907–928, November/December 1995.

[119] Nicola Guarino. Understanding, building and using ontologies. *International Journal of Human-Computer Studies*, 46(2-3):293–310, February/March 1997.

[120] Nicola Guarino. Formal Ontology and Information Systems. In *Proceedings of the 1st International Conference on Formal Ontology in Information Systems (FOIS), Trento, Italy*, pages 3–15. IOS Press, June 1998.

[121] Michael Guttmann. A Response to Steve Cook. *MDA Journal*, February 2004.

[122] Brent Hailpern and Peri Tarr. Model-driven development: The good, the bad, and the ugly. *IBM Systems Journal*, 45(3):451–461, July 2006.

[123] Michael Hammer and James Champy. *Reengineering the Corporation: A Manifesto for Business Revolution*. Harper Business, New York, 1993.

[124] David Harel and Bernhard Rumpe. Meaningful Modeling: What's the Semantics of "Semantics"? *IEEE Computer*, 37(10):64–2, October 2004.

[125] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105, March 2004.

[126] Ian Horrocks and Peter Patel-Schneider. Reducing OWL entailment to description logic satisfiability. *Journal of Web Semantics*, 1(4):345–357, October 2004.

[127] IBM. Rational Software Architect. `http://www-306.ibm.com/software/awdtools/architect/swarchitect/index.html`.

[128] IBM. WebSphere. `http://www-306.ibm.com/software/websphere/`.

[129] IBM, BEA Systems, Microsoft, SAP AG, and Siebel Systems. Business Process Execution Language for Web Services version 1.1, May 2003.

[130] IBM alphaWorks. Model Transformation Framework (MTF). `http://www.alphaworks.ibm.com/tech/mtf`.

[131] IDEAS – Thematic Network. A Gap Analysis. Deliverables D3.4, D3.5, D3.6, May 2003.

[132] IDEAS – Thematic Network. The Vision for 2010. Deliverable D2.4, May 2003.

[133] Institute for Software Integrated Systems (ISIS). The Generic Modeling Environment. `http://www.isis.vanderbilt.edu/projects/gme/`.

[134] Internet Engineering Task Force (IETF). Uniform Resource Identifiers (URI): Generic Syntax. RfC 2396, August 1998.

[135] IRISA-Triskell. Kermeta: Triskell Metamodeling Kernel. Kermeta.

[136] Igor Ivkovic and Kostas Kontogiannis. Tracing Evolution Changes of Software Artifacts through Model Synchronization. In *20th IEEE International Conference on Software Maintenance, Washington, DC, USA*, pages 252–261. IEEE Computer Society, September 2004.

[137] JAMDA. JAMDA Java Model Driven Architecture. `http://sourceforge.net/projects/jamda`.

[138] Jena. Jena – A Semantic Web Framework for Java. `http://jena.sourceforge.net/`.

[139] Nicholas R. Jennings, Katia Sycara, and Michael Wooldridge. A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

[140] Nick R. Jennings, Tim J. Norman, Peyman Faratin, Paul O'Brien, and Brian Odgers. Autonomous Agents for Business Process Management. *International Journal of Applied Artificial Intelligence*, 14(2):145–189, 2000.

[141] Nick R. Jennings, Tim J. Norman, Peyman Faratin, Paul O'Brien, Brian Odgers, and James L. Alty. Implementing a Business Process Management System using ADEPT: A Real-World Case Study. *International Journal of Applied Artificial Intelligence*, 14(5):421–463, 2000.

[142] Richard Johnson, David Pearson, and Keshav Pingali. The Program Structure Tree: Computing Control Regions in Linear Time. In *ACM SIGPLAN 1994 conference on Programming language design and implementation (PLDI), Orlando, Florida, United States*, pages 171–185. ACM Press, 1994.

[143] Simon Johnston. UML 2.0 Profile for Software Services. *IBM developerworks*, April 2005.

[144] Frédéric Jouault and Jean Bézivin. KM3: a DSL for Metamodel Specification. In *8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS), Bologna, Italy*, volume 4037 of *Lecture Notes in Computer Science*, pages 171–185. Springer, June 2006.

[145] Frédéric Jouault and Ivan Kurtev. Transforming Models with ATL. In *Model Transformations in Practice Workshop at MoDELS 2005, Montego Bay, Jamaica*, volume 3844 of *Lecture Notes in Computer Science*, pages 128–138. Springer, October 2005.

[146] Frédéric Jouault and Ivan Kurtev. On the interoperability of model-to-model transformation languages. *Science of Computer Programming*, 68(3):114–137, October 2007.

[147] Jürgen Jung. Meta-Modelling Support for a General Process Modelling Tool. In *5th OOPSLA Workshop on Domain-Specific Modeling*, 2005.

[148] Matjaz B. Juric. A Hands-on Introduction to BPEL. Technical report, Oracle.

[149] Audris Kalnins, Janis Barzdins, and Edgars Celms. Model Transformation Language MOLA. In *European MDA Workshops: Foundations and Applications, MDAFA 2003 and MDAFA 2004, Twente, The Netherlands, and Linköping, Sweden*, volume 3599 of *Lecture Notes in Computer Science*, pages 62–76. Springer, June 2003/04.

[150] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Nowak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90- TR-21, Software Engineereing Institute, Carnegie Mellon University, Pittsburgh, PA, USA, 1990.

[151] Gerti Kappel, Elisabeth Kapsammer, Horst Kargl, Gerhard Kramler, Thomas Reiter, Werner Retschitzegger, Wieland Schwinger, and Manuel Wimmer. 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 (MoDELS/UML), Genova, Italy*, volume 4199 of *Lecture Notes in Computer Science*, pages 528–542. Springer, October 2006.

[152] Gerti Kappel, Elisabeth Kapsammer, Horst Kargl, Gerhard Kramler, Thomas Reiter, Werner Retschitzegger, Wieland Schwinger, and Manuel Wimmer. On Model and Ontologies - A Layered Approach for Model-based Tool Integration. In *Proceedings of Modellierung 2006, Innsbruck, Austria*, Lecture Notes in Informatics. GI, March 2006.

[153] Gerti Kappel, Horst Kargl, Gerhard Kramler, Andrea Schauerhuber, Martina Seidl, Michael Strommer, and Manuel Wimmer. Matching Metamodels with Semantic Systems - An Experience Report. In *Model Management und Metadaten-Verwaltung - BTW 2007 Workshop, Aachen, Germany*, March 2007.

[154] Vipul Kashyap and Amit Sheth. Semantic and schematic similarities between database objects: a context-based approach. *The VLDB Journal*, 5(4):276–304, December 1996.

[155] Alfons Kemper and Andre Eickler. *Datenbanksysteme – Eine Einführung*. Oldenbourg, 6 edition, March 2006.

[156] Bartosz Kiepuszewski, Arthur H.M. ter Hofstede, and Wil M.P. van der Aalst. Fundamentals of control flow in workflows. *Acta Informatica*, 39(3):143–209, March 2003.

[157] Ralf Klein, Florian Kupsch, and August-Wilhelm Scheer. Modellierung interorganisationaler Prozesse mit Ereignisgesteuerten Prozessketten. In *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, volume 178. Scheer, 2004.

[158] Anneke Kleppe, Jos Warmer, and Wim Bast. *MDA Explained, The Model Driven Architecture: Practice and Promise.* Addison-Wesley, April 2003.

[159] Alexander Königs. Model Transformation with Triple Graph Grammars. In *Proceedings of Model Transformations in Practice Workshop, MoDELS Conference, Montego Bay, Jamaica*, October 2005.

[160] Ivan Kurtev, Jean Bézivin, and Mehmet Aksit. Technological Spaces: An Initial Appraisal. In *International Federated Conference (DOA, ODBASE, CoopIS), Industrial Track, Irvine, CA, USA*, 2002.

[161] Oliver Kutz, Carsten Lutz, Frank Wolter, and Michael Zakharyaschev. *E*-connections of abstract description systems. *Artificial Intelligence*, 156(1):1–73, June 2004.

[162] Grant Larsen. Model-driven development: Assets and reuse. *IBM Systems Journal*, 45(3):541–553, July 2006.

[163] Michael Lawley and Jim Steel. Practical Declarative Model Transformation with Tefkat. In *Proceedings of Model Transformations in Practice Workshop, MoDELS Conference, Montego Bay, Jamaica*, volume 3844 of *Lecture Notes in Computer Science*, pages 139–150. Springer, October 2005.

[164] Akos Ledeczi, Miklos Maroti, Arpad Bakay, Gabor Karsai, Jason Garrett, Charles Thomason, Greg Nordstrom, Jonathan Sprinkle, and Peter Volgyesi. The Generic Modeling Environment. In *Proceedings of the IEEE Workshop on Intelligent Signal Processing (WISP'01), Budapest, Hungary*. IEEE, May 2001.

[165] Christine Legner and Kristin Wende. Towards an Excellence Framework for Business Interoperability. In *19th Bled eConference "eValues", Slovenia*, June 2006.

[166] Maurizio Lenzerini. Data Integration: A Theoretical Perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS)*, pages 233–246. ACM, 2002.

[167] Frank Leymann. Web Services Flow Language (WSFL 1.0). IBM, May 2001.

[168] Frank Leymann, Dieter Roller, and Marc-Thomas Schmidt. Web services and business process management. *IBM Systems Journal*, 41(2):198–211, 2002.

[169] Andreas Limyr, Tor Neple, Arne-Jørgen Berre, and Brian Elvesæter. Semaphore - A Model-Based Semantic Mapping Framework. In *Business Process Management Workshop, BPM 2006 International Workshops, BPD, BPI, ENEI, GPWW, DPM, semantics4ws, Vienna, Austria*, volume 4103 of *Lecture Notes in Computer Science*, pages 275–284. Springer, September 2006.

[170] Sonia Lippe, Ulrike Greiner, and Alistair Barros. A Survey on State of the Art to Facilitate Modelling of Cross-Organisational Business Processes. In *XML4BPM 2005, Proceedings of the 2nd GI Workshop XML4BPM – XML Interchange Formats for Business Process Management at 11th GI Conference BTW 2005, Karlsruhe, Germany*, pages 7–22, http://wi.wu-wien.ac.at/˜mendling/XML4BPM2005/xml4bpm-2005-proceedings-lippe.pdf, March 2005.

[171] Duen-Ren Liu and Minxin Shen. Modeling Workflows with a Process-View Approach. In *DASFAA '01: Proceedings of the 7th International Conference on Database Systems for Advanced Applications, Hong Kong, China*, pages 260–267. IEEE Computer Society, April 2001.

[172] Alexander Maedche, Boris Motik, Nuno Silva, and Raphael Volz. MAFRA - A MApping FRAmework for Distributed Ontologies. In *13th European Conference on Knowledge Engineering and Knowledge Management (EKAW), Siquenca, Spain*, volume 2473 of *Lecture Notes in Computer Science*, pages 235–250. Springer, October 2002.

[173] MagicDraw. MagicDraw: Architecture Made Simple. `http://www.magicdraw.com/`.

[174] Thomas W. Malone. Modeling Coordination in Organizations and Markets. *Management Science*, 33(10):1317–1332, October 1987.

[175] Keith Mantell. From UML to BPEL - Model Driven Architecture in a Web service world. *IBM developerworks*, September 2005.

[176] Manyeta. Codagen Architect. `http://www.manyeta.com/`.

[177] Frank Marschall and Peter Braun. Model Transformations for the MDA with BOTL. In *Workshop on Model Driven Architecture: Foundations and Applications, Enschede, The Netherlands*, number TR-CTIT-03-27 in CTIT Technical Report, pages 25–36. University of Twente, June 2003.

[178] Martin Matula. NetBeans Metadata Repository, March 2003.

[179] Paul T. Maurer. ActiveBPEL 3.0 from Active Endpoints, Inc. *SOA World Magazine*, pages 22–23, December 2006.

[180] Stephen J. Mellor and Marc J. Balcer. *Executable UML: A Foundation for Model-Driven Architecture*. Addison Wesley, May 2002.

[181] Jan Mendling, Kristian Lassen, and Uwe Zdun. Transformation Strategies between Block-Oriented and Graph-Oriented Process Modelling Languages. In *Multikonferenz Wirtschaftsinformatik (MKWI)*, volume 2, pages 297–312. GITO-Verlag, 2006.

[182] Tom Mens and Pieter Van Gorp. A Taxonomy of Model Transformation. *Electronic Notes in Theoretical Computer Science*, 152:125–142, September 2005. International Workshop on Graph and Model Transformation (GraMoT), Tallinn, Estonia.

[183] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and How to Develop Domain-Specific Languages. *ACM Computing Surveys (CSUR)*, 37(4):316–344, December 2005.

[184] Kai Mertins and Roland Jochem. Integrated Enterprise Modeling: Method and Tool. *SIGGROUP Bulletin*, 18(2):63–66, August 1997.

[185] MetaCase. Domain-Specific Modeling with MetaEdit+. `http://www.metacase.com/`.

[186] Microsoft. Domain-Specific Language Tools. `http://msdn.microsoft.com/vstudio/DSLTools/`.

[187] MID. Modellierungsplattform Innovator. `http://www.mid.de/Innovator.html`.

[188] George A. Miller. The Magical Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information. *The Psychological Review*, 63:81–97, 1956.

[189] Renée J. Miller, Mauricio A. Hernéndez, Laura M. Haas, Lingling Yan, C. T. Howard Ho, Ronald Fagin, and Lucian Popa. The Clio Project: Managing Heterogeneity. *SIGMOD Rec.*, 30(1):78–83, March 2001.

[190] Tova Milo and Sagit Zohar. Using Schema Matching to Simplify Heterogeneous Data Translation. In *Proceedings of the 24rd International Conference on Very Large Data Bases (VLDB*, pages 122–133. Morgan Kaufmann Publishers Inc., August 1998.

[191] Michele Missikoff, Federica Schiappelli, and Francesco Taglino. A Controlled Language for Semantic Annotation and Interoperability in e-Business Applications. 2003.

[192] Prasenjit Mitra, Gio Wiederhold, and Martin L. Kersten. A Graph-Oriented Model for Articulation of Ontology Interdependencies. In *Proceedings of the 7th International Conference on Extending Database Technology (EDBT)*, volume 1777 of *Lecture Notes in Computer Science*, pages 86–100. Springer-Verlag, March 2000.

[193] ModelCVS. ModelCVS Project: A Semantic Infrastructure for Model-based Tool Integration. `http://www.modelcvs.org/`.

[194] MODELPLEX. MODELPLEX project: MODELling solution for comPLEX software system. `http://www.modelplex-ist.org/`.

[195] MODELWARE. MODELWARE project: MODELling solution for softWARE systems. `http://www.modelware-ist.org/`.

[196] MOFScript. MOFScript. `http://www.eclipse.org/gmt/mofscript/`.

[197] MOMOCS. MOMOCS project: MOdel driven MOdernisation of Complex Systems. `http://www.momocs.org/`.

[198] Peter D. Mosses. *Action semantics*. Cambridge University Press, New York, NY, USA, 1992.

[199] Pierre-Alain Muller, Franck Fleurey, and Jean-Marc Jézéquel. Weaving Executability into Object-Oriented Metalanguages. In *8th International Conference on Model Driven Engineering Languages and Systems (MoDELS), Montego Bay, Jamaica*, volume 3713 of *Lecture Notes in Computer Science*, pages 264–278. Springer, October 2005.

[200] Pierre-Alain Muller, Philippe Studer, Frédéric Fondement, and Jean Bézivin. Platform independent Web application modeling and development with Netsilon. *Software and System Modeling*, 4(4):424–442, June 2005.

[201] NESSI. Networked European Software and Services Initiative. `http://www.nessi-europe.com/Nessi`.

[202] NetBeans. Metadata Repository (MDR). `http://mdr.netbeans.org/`.

[203] OASIS. ebXML Business Process Specification Schema Technical Specification v2.0.4. ebxmlbp-v2.0.4-Spec-os-en, December 2006.

[204] OASIS. Reference Model for Service Oriented Architecture, Committee Draft 1.0. wd-soa-rm-cd1, February 2006.

[205] OASIS. Web Services Business Process Execution Language Version 2.0. wsbpel-primer, May 2007.

[206] oAW. openArchitectureWare (oaw). http://www.openarchitectureware.org/.

[207] Daniel Oberle. *Semantic Management of Middleware*, volume 1 of *Semantic Web and Beyond*. Springer, 2005.

[208] OMG. OMG Model Driven Architecture. http://www.omg.org/mda/.

[209] OMG. Common Warehouse Metamodel (CWM) Specification, version 1.0. ad/01-02-01, February 2001.

[210] OMG. Meta Object Facility (MOF) Specification, version 1.4. formal/02-04-03, April 2002.

[211] OMG. Request for Proposal: MOF 2.0 Query / View / Transformations RFP. ad/2002-04-10, April 2002.

[212] OMG. MDA Guide Version 1.0. omg/2003-05-01, May 2003.

[213] OMG. MDA Guide Version 1.0.1. omg/2003-06-01, June 2003.

[214] OMG. Response to the MOF 2.0 Query/Views/Transformations RFP (ad/2002-04-10). ad/2003-08-05, August 2003.

[215] OMG. Revised submission for MOF 2.0 Query / Views / Transformations RFP, Version 1.1. ad/03-08-08, August 2003.

[216] OMG. A Definition of MDA. ormsc/04-08-02, August 2004.

[217] OMG. Addendum to the minutes of ORMSC Meeting - Orlando, FL (June 2004). ormsc/04-06-01, June 2004.

[218] OMG. Business Process Definition MetaModel (BPDM), Revised Submission to BEI RFP bei/2003-01-06. bei/2004-08-03, August 2004.

[219] OMG. MOF Model to Text Transformation Language, Request For Proposal. ad/2004-04-07, April 2004.

[220] OMG. UML Profile for Enterprise Distributed Object Computing (EDOC). http://www.omg.org/technology/documents/formal/edoc.htm, 2004.

[221] OMG. MOF 2.0/XMI Mapping Specification, Version 2.1. formal/05-09-01, September 2005.

[222] OMG. MOF QVT Final Adopted Specification. ptc/05-11-01, November 2005.

[223] OMG. Software Process Engineering Metamodel Specification, version 1.1. formal/05-01-06, January 2005.

[224] OMG. Business Motivation Model (BMM) Specification. dtc/2006-08-03, August 2006.

[225] OMG. Business Process Definition MetaModel (BPDM), final submission. bmi/2006-11-03, December 2006.

[226] OMG. Business Process Modeling Notation Specification, Final Adopted Specification. dtc/06-02-01, February 2006.

[227] OMG. MDA Guide Draft 0.1.5. ormsc/06-09-03, September 2006.

[228] OMG. Meta Object Facility (MOF) Core Specification, version 2.0. formal/06-01-01, January 2006.

[229] OMG. Object Constraint Language, OMG Available Specification, Version 2.0. formal/06-05-01, May 2006.

[230] OMG. Ontology Definition Metamodel, Sixth Revised Submission. ad/2006-05-01, May 2006.

[231] OMG. Revised submission for MOF Model to Text Transformation Language RFP (ad/2004-04-07), version 1.3. ad/2006-04-03, April 2006.

[232] OMG. UML Profile and Metamodel for Services (UPMS), Request For Proposal. soa/2006-09-09, September 2006.

[233] OMG. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification - Final Adopted Specification. ptc/07-07-07, July 2007.

[234] OMG. Ontology Definition Metamodel, Adopted Specification. ptc/2007-09-09, November 2007.

[235] OMG. UML Profile and Metamodel for Services - for Heterogeneous Architectures (UPMS-HA). ad/2007-06-02, June 2007.

[236] OMG. UML Profile and Metamodel for Services, Initial Submission. ad/2007-06-03, June 2007.

[237] OMG. Unified Modeling Language: Infrastructure, version 2.1.1. formal/07-02-06, February 2007.

[238] OMG. Unified Modeling Language: Superstructure, version 2.1.1. formal/07-02-05, February 2007.

[239] OMG. White paper: Architecture-Driven Modernization – Transforming the Enterprise. admtf/07-12-01, December 2007.

[240] Oracle. *Oracle® BPEL Process Manager, Developer's Guide*, October 2005.

[241] Chun Ouyang, Marlon Dumas, Arthur H.M. ter Hofstede, and Wil M.P. van der Aalst. Pattern-based translation of BPMN process models to BPEL web services. *International Journal of Web Services Research (JWSR)*, 5(1):42–62, 2007.

[242] Terence Parr. ANTLR 3.0. `http://antlr.org/`, 2007.

[243] Terence Parr. *The Definitive ANTLR Reference: Building Domain-Specific Language*. Pragmatic Bookshelf, May 2007.

[244] Octavian Patrascoiu. YATL:Yet Another Transformation Language. In *1st European MDA Workshop, MDA-IA, Twente, The Netherlands*, pages 83–90, January 2004.

[245] Pellet. Pellet: The Open Source OWL DL Reasoner. `http://pellet.owldl.com/`.

[246] Rachel A. Pottinger and Philip A. Bernstein. Merging Models Based on Given Correspondences. In *Proceedings of the 29th international conference on Very large data bases (VLDB), Berlin, Germany*, pages 862–873. VLDB Endowment, 2003.

[247] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, December 2001.

[248] Stephan Roser. Modellgetriebene Geschäftsprozessautomatisierung. Technical Report 2005-09, Institute of Computer Science, Univeristy of Augsburg, April 2005.

[249] Stephan Roser and Bernhard Bauer. A Categorization of Collaborative Business Process Modeling Techniques. In *Workshop on Service oriented Solutions for Cooperative Organizations (SoS4CO) in 7th International IEEE Conference on E-Commerce Technology, Munich, Germany*, pages 43–51, Washington, DC, USA, July 2005. IEEE Computer Society.

[250] Stephan Roser and Bernhard Bauer. Ontology-Based Model Transformation. In *Satellite Events at the MoDELS 2005 Conference, MoDELS 2005 International Workshops, Doctoral Symposium, Educators Symposium, Montego Bay, Jamaica*, volume 3844 of *Lecture Notes in Computer Science*, pages 355–356. Springer, October 2005.

[251] Stephan Roser and Bernhard Bauer. An Approach to Automatically Generated Model Transformations Using Ontology Engineering Space. In *2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE), Athens, GA, USA*, November 2006.

[252] Stephan Roser and Bernhard Bauer. Automatic Generation and Evolution of Model Transformations Using Ontology Engineering Space. *Journal on Data Semantics, Springer LNCS*, 2008, forthcoming.

[253] Stephan Roser, Florian Lautenbacher, and Bernhard Bauer. Generation of Workflow Code from DSMs. In J. Sprinkle, J. Gray, M. Rossi, and J.-P. Tolvanen, editors, *7th OOPSLA Workshop on Domain-Specific Modeling, Montreal, Canada*, number 38 in Computer Science and Information System Reports. University of Jyväskylä, October 2007.

[254] Stephan Roser, Florian Lautenbacher, and Bernhard Bauer. MDSD light for ERP. In *23rd Annual ACM symposium on Applied computing (SAC) - Enterprise Information Systems Track, Fortaleza, Ceará, Brazil*, March 2008.

[255] Jeff Rothenberg. *Artificial intelligence, simulation & modeling*, chapter The Nature of Modeling, pages 75–92. John Wiley & Sons, Inc., New York, NY, USA, 1989.

[256] Thomas L. Saaty. *The Analytic Hierarchy Process*. McGraw-Hill, New York, 1980.

[257] Thomas L. Saaty. How to make a decision: The Analytic Hierarchy Process. *European Journal of Operational Research*, 48(1):9–26, September 1990.

[258] Thomas L. Saaty. How to make a decision: The Analytic Hierarchy Process. *Interfaces*, 24(6):19–43, November-December 1994.

[259] Thomas L. Saaty. *Decision Making for Leaders*. RWS Publications, 3rd edition, 1999.

[260] SAP Research. Maestro. `http://www.athena-ip.org`, 2006.

[261] August-Wilhelm Scheer. *ARIS – Vom Geschäftsprozess zum Anwendungssystem*. Springer, 3 edition, 1998.

[262] Douglas C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, February 2006.

[263] Karsten A. Schulz and Maria E. Orlowska. Facilitating cross-organisational workflows with a workflow view approach. *Data & Knowledge Engineering*, 51(1):109–147, April 2004.

[264] SECSE. SECSE project: Service Centric System Engineering. `http://secse.eng.it/`.

[265] Ed Seidewitz. What models mean. *IEEE Software*, 20(5):26–32, September/October 2003.

[266] Shane Sendall and Wojtek Kozaczynski. Model Transformation: The Heart and Soul of Model-Driven Software Development. *IEEE Software*, 20(5):42–45, September/October 2003.

[267] Luciano Serafini, Heiner Stuckenschmidt, and Holger Wache. A Formal Investigation of Mapping Language for Terminological Knowledge. In *19th International Joint Conference on Artificial Intelligence (IJCAI), Edinburgh, Scotland*, pages 576–581. Professional Book Center, July/August 2005.

[268] SERIOUS. SERIOUS project: Software Evolution, Refactoring, Improvement of Operational & Usable Systems. `http://lore.cmi.ua.ac.be/serious`.

[269] Pavel Shvaiko and Jérôme Euzenat. A Survey of Schema-Based Matching Approaches. *Journal on Data Semantics IV*, 3730:146–171, 2005.

[270] Charles C. Snow, Raymond E. Miles, and Henry J. Coleman. Managing 21st Century Network Organizations. *Organizational Dynamics*, 20(3):5–20, 1992.

[271] Arnor Solberg, Robert France, and Raghu Reddy. Navigating the MetaMuddle. In *4th Workshop in Software Model Engineering (WiSME'05), Montego Bay, Jamaica*, 2005.

[272] Sourceforge Project. Ontology-based Model Transformation (OntMT). `http://sourceforge.net/projects/ontmt/`.

[273] Sourceforge Project. QVT Relations Parser. `http://sourceforge.net/projects/qvtparser/`.

[274] Sourceforge Project. Token Analysis. `http://sourceforge.net/projects/tokenanalysis/`.

[275] Sourceforge Project. Workflow Generation Framework. `http://sourceforge.net/projects/wf-codegen`.

[276] Günter Spur, Kai Mertins, and Roland Jochem. *Integrated Enterprise Modelling*. Beuth Verlag, 1996.

[277] Fabian Stäber, Giorgio Sobrito, Jörg P. Müller, Udo Bartlang, and Thomas Friese. Interoperability challenges and solutions in Automotive Collaborative Product Development. In *Enterprise Interoperability II: New Challenges and Approaches (I-ESA'07)*, pages 709–720. Springer, March 2007.

[278] Tomas Stahl and Markus Völter. *Model-Driven Software Development: Technology, Engineering, Management*. Wiley, May 2006.

[279] Jim Steel and Jean-Marc Jézéquel. Model Typing for Improving Reuse in Model-Driven Engineering. In *8th International Conference on Model Driven Engineering Languages and Systems (MoDELS), Montego Bay, Jamaica*, volume 3713 of *Lecture Notes in Computer Science*, pages 84–96. Springer, October 2005.

[280] Paul A. Strassmann. Is Outsourcing Profitable? Lecture at George Mason University, March 2006.

[281] StringTemplate. StringTemplate. `http://www.stringtemplate.org/`.

[282] Gerson Sunyé, Damien Polleta, Yves Le Traon, and Jean-Marc Jézéquel. Refactoring UML Models. In *4th International Conference on The Unified Modeling Language Conference, Modeling Languages, Concepts, and Tools, Toronto, Canada*, volume 2185 of *Lecture Notes in Computer Science*, pages 134–148. Springer, October 2001.

[283] Peter Swithinbank, Mandy Chessell, Tracy Gardner, Catherine Griffin, Jessica Man, Helen Wylie, and Larry Yusuf. Patterns: Model-Driven Development Using IBM Rational Software Architect. IBM Redbook SG24-7105-00b, December 2005.

[284] Janos Sztipanovits and Gabor Karsai. Model-Integrated Computing. *Computer*, 30(4):110–111, April 1997.

[285] Gabriele Taentzer. AGG: A Graph Transformation Environment for Modeling and Validation of Software. In *2nd International Workshop on Application of Graph Transformations with Industrial Relevance (AGTIVE), Charlottesville, VA, USA*, volume 3062 of *Lecture Notes in Computer Science*, pages 446–453. Springer, September 2003.

[286] Satish Thatte. XLANG: Web Services for Business Process Design. Microsoft, 2001.

[287] D. A. Thomas and B. M. Barry. Model-Driven Development: the Case for Domain-Oriented Programming. In *Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 2–7, October 2003.

[288] Juha-Pekka Tolvanen. Making Model-Based Code Generation Work. *Embedded Systems Europe*, 8(60):36–38, August/September 2004.

[289] Juha-Pekka Tolvanen and Steven Kelly. Domänenspezifische modellierung. *Objektspektrum*, 4:30–34, July/August 2004.

[290] Laurence Tratt. The MT Model Transformation Language. In *21st Annual ACM symposium on Applied computing (SAC) - Model Transformation Track, Dijon, France*, pages 1296–1303. ACM, April 2006.

[291] TRDDC. ModelMorf: A Model Transformer. `http://www.tcs-trddc.com/ModelMorf/index.htm`.

[292] UML 2 Semantics Project. UML 2 Semantics Project. `http://www.cs.queensu.ca/~stl/internal/uml2/index.html`.

[293] Will M.P. van der Aalst and Arthur H.M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.

[294] Jussi Vanhatalo, Hagen Völzer, and Frank Leymann. Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. In *5th International Conference Service-Oriented Computing (ICSOC), Vienna, Austria*, volume 4749 of *Lecture Notes in Computer Science*, pages 43–55. Springer, September 2007.

[295] Dóniel Varró and Andrós Pataricza. Generic and Meta-transformations for Model Transformation Engineering. In *7th International Conference on the Unified Modeling Language, Lisbon, Portugal*, volume 3273 of *Lecture Notes in Computer Science*, pages 290–304. Springer, October 2004.

[296] Dóniel Varró, Gergely Varró, and Andrós Pataricza. Designing the automatic transformation of visual languages. *Science of Computer Programming*, 44(2):205–227, August 2002.

[297] Velocity. The Apache Velocity Project. `http://velocity.apache.org/`.

[298] Eelco Visser. A survey of rewriting strategies in program transformation systems. *Electronic Notes in Theoretical Computer Science, 1st International Workshop on Reduction Strategies in Rewriting and Programming (WRS'01), Utrecht, The Netherlands*, 57:109–143, December 2001.

[299] Didier Vojtisek and Jean-Marc Jézéquel. MTL and Umlaut NG: Engine and Framework for Model Transformation, 2004.

[300] W3C. Rule Interchange Format (RIF) Working Group. `http://www.w3.org/2005/rules/wiki/RIF_Working_Group`.

[301] W3C. XSL Transformations (XSLT), Version 1.0. `http://www.w3.org/TR/xslt`, November 1999.

[302] W3C. Web Services Description Language (WSDL) 1.1. `http://www.w3.org/TR/wsdl`, March 2001.

[303] W3C. Web Service Choreography Interface (WSCI) 1.0. http://www.w3.org/TR/wsci, August 2002.

[304] W3C. OWL-S: Semantic Markup for Web Services. http://www.w3.org/Submission/OWL-S, November 2004.

[305] W3C. OWL Web Ontology Language for Services (OWL-S). http://www.w3.org/Submission/2004/07/, July 2004.

[306] W3C. OWL Web Ontology Language Semantics and Abstract Syntax. http://www.w3.org/TR/owl-semantics/, February 2004.

[307] W3C. RDF Primer. http://www.w3.org/TR/rdf-primer/, February 2004.

[308] W3C. RDF Semantics. http://www.w3.org/TR/rdf-mt/, February 2004.

[309] W3C. RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema/, February 2004.

[310] W3C. Resource Description Framework (RDF): Concepts and Abstract Syntax. http://www.w3.org/TR/rdf-concepts/, February 2004.

[311] W3C. SKOS Mapping Vocabulary Specification. http://www.w3.org/2004/02/skos/mapping/spec/, November 2004.

[312] W3C. Web Services Architecture, W3C Working Group Note. http://www.w3.org/TR/ws-arch/, February 2004.

[313] W3C. Web Services Choreography Description Language Version 1.0. http://www.w3.org/TR/ws-cdl-10/, November 2005.

[314] W3C. Extensible Markup Language (XML) 1.0 (Fourth Edition). http://www.w3.org/TR/xml/, September 2006.

[315] W3C. SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/, November 2007.

[316] W3C. XML Path Language (XPath) 2.0. http://www.w3.org/TR/xpath20/, January 2007.

[317] Ueli Wahli, Larissa Leybovich, Eric Prevost, Russell Scher, Andre Venancio, Sascha Wiederkom, and Neil MacKinnon. Business Process Management: Modeling through Monitoring Using WebSphere V6 Products. IBM Redbook SG24-7148-00, April 2006.

[318] John J. Wallis and C. North Douglas. *Measuring the transactions sector in the American economy*, volume volume of *series*. University of Chicago Press, Chicago, Long-term factors in American economic growth edition, 1986.

[319] WfMC. Process Definition Interface – XML Process Definition Language. WFMC-TC-1025, October 2005.

[320] Stephen White. Best Practices for Using WebSphere Business Modeler and Monitor. IBM Redpaper REDP-4159-00, April 2006.

[321] Ian Wilkie, Adrian King, Mike Clarke, Chas Weaver, Chris Raistrick, and Paul Francis. *UML ASL Reference Guide for ASL Language Level 2.5 – Manual Revision D*. Kennedy Carter, 2003.

[322] Oliver E. Williamson. *Markets and Hierarchies: Analysis and Antitrust Implications*. Free Press, 1975.

[323] Oliver E. Williamson. Transaction Cost Economics. *Handbook of Industrial Organization*, 1:135–182, 1989.

[324] Edward D. Willink. UMLX: A Graphical Transformation Language for MDA. In *18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, Anaheim, CA*, pages 13–24, 2003.

[325] Manuel Wimmer, Andrea Schauerhuber, Michael Strommer, Wieland Schwinger, and Gerti Kappel. A Semi-automatic Approach for Bridging DSLs with UML. In J. Sprinkle, J. Gray, M. Rossi, and J.-P. Tolvanen, editors, *7th OOPSLA Workshop on Domain-Specific Modeling, Montreal, Canada*, number 38 in Computer Science and Information System Reports. University of Jyväskylä, October 2007.

[326] Manuel Wimmer, Michael Strommer, Horst Kargl, and Gerhard Kramler. Towards Model Transformation Generation By-Example. In *HICSS-40 Hawaii International Conference on System Sciences, Hawaii, USA*, page 285b. IEEE Computer Society, January 2007.

[327] Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, 1993.

[328] Xactium. XMF Mosaic. `http://www.xactium.com/`.

[329] XDoclet. XDoclet Attribute Oriented Programming. `http://xdoclet.sourceforge.net/xdoclet/index.html`.

[330] Jing Zhang, Yuehua Lin, and Jeff Gray. *Model-Driven Software Development*, volume 1, chapter Generic and Domain-Specific Model Refactoring Using a Model Transformation Engine, pages 199–218. Springer, Heidelberg, Germany, 2005.

# Acronyms

**Abstract Syntax TS** Grammarware TS

**ADM** Architecture-Driven Modernization

**AGG** Attributed Graph Grammar

**AGILE** Agile Software Development of Embedded Systems

**AgilPro** Agile Business Processes With Service Enabled Applications

**AHP** Analytic Hierarchy Process

**AJAX** Asynchronous JavaScript and XML

**AM3** ATLAS MegaModel Management

**AMPLE** Aspect-Oriented, Model-driven Product Line Engineering

**AMW** Atlas ModelWeaver

**ANTLR** ANother Tool for Language Recognition

**AOSD** Aspect-oriented Software Development

**API** Application Programming Interface

**ASL** Action Specification Language

**ATHENA IP** Advanced Technologies for Interoperability of Heterogeneous Enterprise Networks and their Applications

**ATL** Atlas Transformation Language

**AToM$^3$** A Tool for Multi-formalism and Meta-Modeling

**ARIS** Architecture of Integrated Information Systems

**B2B** Business-To-Business

**BMM** Business Motivation Model

**BPDM** Business Process Definition Metamodel

**BPEL4WS** Business Process Execution Language for Web Services

**BPML** Business Process Modeling Language

**BPMN** Business Process Modeling Notation

**BPSS** Business Process Specification Schema

**BOTL** Bidirectional Object-oriented Transformation Language

**CASE** Computer-aided Software Engineering

**CBP** Cross-organisational Business Process

**CIM** Computation Independent Model

**CL** Common Logic

**CORBA** Common Object Request Broker Architecture

**C-OWL** Context OWL

**CRM** Customer Relationship Management

**C-SAW** Constraint-Specification Aspect Weaver

**CWM** Common Warehouse Metamodel

**DBMS TS** Dataware and Database Management Systems TS

**DL** Description Logic

**DSL** Domain Specific Language

**DSM** Domain Specific Model

**DSWM** Domain Specific Weaving Metamodel

**EAI** Enterprise Application Integration

**ebXML** eBusiness eXtensible Markup Language

**EDOC** Enterprise Distributed Object Computing

**EMF** Eclipse Modeling Framework

**EMOF** Essential MOF

**EODM** EMF Ontology Definition Metamodel

**EPC** Event-driven Process Chain

**eEPC** extended Event-driven Process Chain

**ERP** Enterprise Resource Planning

**ES** Elementary Service

**ETP** European Technology Platform

**Fujaba** From UML to Java And Back Again

**GReAT** Graph Rewriting and Transformation language

**GME** Generic Modeling Environment

**GMT** Generative Modeling Tools

**GPL** General Public License

**ICT** Information and Communication Technology

**IT** Information Technology

**IDE** Integrated Development Environment

**IDEAS** Interoperability Development for Enterprise Application and Software

**IEM** Integrated Enterprise Modelling

**J2EE** Java 2 Platform Enterprise Edition

**Jamda** Java Model Driven Architecture

**jBPM** JBoss workflow engine

**JET** Java Emitter Templates

**JMI** Java Metadata Interface

**KM3** Kernel MetaMetaModel

**LHS** left-hand side

**LiMo** AgilPro Light Modeler

**MAS** Multi-agent Systems

**MDA** Model Driven Architecture

**MDA TS** Modelware and Model-based Technology TS

**MDD** Model Driven Development

**MDDi** Model Driven Development integration

**MDE** Model Driven Engineering

**MDR** Metadata Repository

**MDSD** Model Driven Software Development

**MDSE** Model Driven Software Engineering

**MIC** Model Integrated Computing

**MO2GO** Method for Object Oriented Business Process Optimization

**ModelCVS** Semantic Infrastructure for Model-based Tool Integration

**MOLA** MOdel transformation LAnguage

**MT** Model Transformation

**MTBE** Model Transformation Generation By-Example

**MTL** Model Transformation Language

**MTF** Model Transformation Framework

**MODELPLEX** MODELling solution for comPLEX software systems

**MODELWARE** MODELling solution for softWARE systems

**MOF** Meta Object Facility

**MOMOCS** MOdel driven MOdernisation of Complex Systems

**NESSI** Networked European Software and Services Initiative

**OASIS** Organization for the Advancement of Structured Information Standards

**oAW** openArchitectureWare

**OCL** Object Constraint Language

**ODM** Ontology Definition Metamodel

**OEM** Original Equipment Manufacturer

**OMG** Object Management Group

**OntMT** Ontology-based Model Transformation

**Ontology TS** Ontologyware and Ontology Engineering TS

**OTE** Object Template Expression

**OWL** Web Ontology Language

**OWL-S** Web Ontology Language for Services

**P2P** peer-to-peer

**PIM** Platform Independent Model

**PIM4SOA** Platform Independent Model for Service Oriented Architectures

**PM** Platform Model

**PO** Purchasing Organisation

**PP** Private Process

**PSM** Platform Specific Model

**PTI** Property Template Item

**QVT** Query, Views, and Transformation

**RDF** Resource Description Framework

**RDF(S)** RDF Schema

**RHS** right-hand side

**RIF** Rule Interchange Format

**RO** Reference Ontology

**SECSE** Service Centric System Engineering

**Semaphore** Model-based Semantic Mapping Framework

**SERIOUS** Software Evolution, Refactoring, Improvement of Operational & Usable Systems

**SESE** Single-Entry-Single-Exit Component

**SKOS** Simple Knowledge Organisation Systems

**SMAIL** Semantic Mediation and Application Interoperability Language

**SME** Small and Medium Enterprise

**SOA** Service Oriented Architecture

**SOR** Statement of Requirements

**SPARQL** SPARQL Protocol and RDF Query Language

**SPEM** Software Process Engineering Metamodel

**SPL** Software Product Line

**SPL4AOX** Service Modeling Language for MID innovatorAOX

**SU** Supplier

**TBox** terminological component / vocabulary in knowledge base

**TM** Topic Maps

**TS** Technological Space

**UML** Unified Modeling Language

**UPMS** UML Profile and Metamodel for Services

**UPMS–HA** UML Profile and Metamodel for Services – for Heterogeneous Architectures

**URI** Uniform Resource Identifier

**VIATRA** VIsual Automated model TRAnsformations

**VP** View Process

**W3C** World Wide Web Consortium

**WS-BPEL** Web Services Business Process Execution Language

**WSA** Web Service Architecture

**WS–CDL** Web Services Choreography Description Language

**WSDL** Web Services Description Language

**WSFL** Web Services Flow Language

**WCSI** Web Service Choreography Interface

**XMI** XML Metadata Interchange

**XML** Extensible Markup Language

**XML TS** Structured Document TS

**XPath** XML Path Language

**XPDL** XML Process Definition Language

**XSLT** Extensible Stylesheet Language Transformation

**YATL** Yet Another Transformation Language

**YAWL** Yet Another Workflow Language

# List of Figures

# List of Tables

# Listings

# Appendix A

# CBP Enactment

Listing A.1: AgilPro case study generated code

```
<?xml version="1.0" encoding="UTF-8"?>
<process name="CreateOffer"
  xmlns="http://schemas.xmlsoap.org
              /ws/2003/03/business-process/"
  xmlns:acc="urn:samples:account"
  xmlns:agi="urn:samples:agilpro"
  xmlns:atm="urn:samples:atm"
  xmlns:bpel="http://schemas.xmlsoap.org
              /ws/2003/03/business-process/"
  xmlns:tic="urn:samples:ticket"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  targetNamespace="urn:samples:atm">
 <partnerLinks>
  <!-- relationship with the ATM -->
  <partnerLink myRole="FrontEnd" name="atm"
   partnerLinkType="atm:Atm-Front"/>
  <!-- relationship with the ticket issuer -->
  <partnerLink name="ticket" partnerRole="TicketIssuer"
   partnerLinkType="atm:Front-Ticket" />
  <!-- relationship with the account system -->
  <partnerLink name="account" partnerRole="AccountSystem"
   partnerLinkType="atm:Front-Account" />
  <!-- relationship with the agilpro framework -->
  <partnerLink name="agilpro" partnerRole="AgilproIssuer"
   partnerLinkType="atm:Front-Agilpro" />
 </partnerLinks>
 <variables>
  <!-- ATM connection request -->
  <variable name="connectReq" messageType="atm:connectRequest"/>
  <!-- ticket creation request -->
  <variable name="ticketReq"
   messageType="tic:ticketRequest" name="ticketReq"/>
  <!-- ticket number wrapper -->
  <variable name="ticketMsg"
   messageType="tic:ticketMessage"/>
  <!-- ATM connection flag -->
  <variable name="connected" type="xsd:boolean"/>
  <!-- customer session flag -->
  <variable name="logged" type="xsd:boolean"/>
  <variable name="connectToAgilproReq"
```

```
    messageType="agi:connectToAgilproRequest"/>
   <variable name="setApplicationReq"
    messageType="agi:setApplicationRequest"/>
   <variable name="setValueToObjectReq"
    messageType="agi:setValueToObjectRequest"/>
   <variable name="getValueFromObjectReq"
    messageType="agi:getValueFromObjectRequest"/>
   <variable name="getValueFromObjectRes"
    messageType="agi:getValueFromObjectResponse"/>
   <variable name="startActionReq"
    messageType="agi:startActionRequest"/>
   <variable name="endActionReq"
    messageType="agi:endActionRequest"/>
   <variable name="nextActionReq"
    messageType="agi:nextActionRequest"/>
   <variable name="checkGuardReq"
    messageType="agi:checkGuardRequest"/>
   <variable name="checkGuardRes"
    messageType="agi:checkGuardResponse"/>
   <variable name="disconnectAgilproReq"
    messageType="agi:disconnectAgilproRequest"/>
   <variable name="disconnectAgilproRes"
    messageType="agi:disconnectAgilproResponse"/>
  </variables>
  <correlationSets>
   <!-- conversation with a connected ATM -->
   <correlationSet name="atmInteraction"
    properties="atm:ticketId"/>
  </correlationSets>
  <!-- structure of process -->
  <sequence name="mainSequence">
   <!-- receive a connection request -->
   <receive name="connectIn" createInstance="yes"
     portType="atm:FrontEnd" operation="connect"
     partnerLink="atm" variable="connectReq"/>
   <!-- generate a ticket number -->
   <invoke name="createTicket" partnerLink="ticket"
     portType="tic:TicketIssuer" operation="createTicket"
     inputVariable="ticketReq"  outputVariable="ticketMsg">
    <correlations>
     <correlation initiate="yes" pattern="in"
      set="atmInteraction"/>
    </correlations>
   </invoke>
   <!-- initialize the status flags -->
   <assign name="initConnection">
    <copy>
     <from expression="true()"/>
     <to variable="connected"/>
    </copy>
    <copy>
     <from expression="false()"/>
     <to variable="logged"/>
    </copy>
   </assign>
   <!-- send the ticket number back to the ATM -->
   <reply name="connectOut"
     portType="atm:FrontEnd" operation="connect"
     partnerLink="atm" variable="ticketMsg">
```

```
    <correlations>
     <correlation set="atmInteraction"/>
    </correlations>
   </reply>
   <!-- handle the ATM connection -->
   <scope name="connectionUnit"
    variableAccessSerializable="no">
    <variables>
     <!-- customer log on request -->
     <variable name="logOnReq"
      messageType="atm:logOnRequest"/>
     <!-- connection status response -->
     <variable name="statusRsp"
      messageType="atm:statusResponse"/>
    </variables>
    <sequence name="START">
     <receive
       portType="atm:FrontEnd" operation="connectToAgilpro"
       partnerLink="atm" variable="connectToAgilproReq">
      <correlations>
       <correlation set="atmInteraction"/>
      </correlations>
     </receive>
     <assign name="copy_Ticketnumber">
      <copy>
       <from part="Ticketnumber" variable="connectToAgilproReq"/>
       <to part="Ticketnumber" variable="nextActionReq"/>
      </copy>
     </assign>
     <scope name="Defaultname">
      <sequence>
       <empty/>
      </sequence>
     </scope>
     <scope name="CreateOffer">
      <sequence>
       <empty/>
       <assign name="set_App_OfferManagement">
        <copy>
         <from part="Ticketnumber" variable="nextActionReq"/>
         <to part="Ticketnumber" variable="setApplicationReq"/>
        </copy>
        <copy>
         <from expression="string(
          'eu.emundo.agilpro.fw.fe.erp.intf.AngebotUi')"/>
         <to part="JavaClassIN" variable="setApplicationReq"/>
        </copy>
        <copy>
         <from expression="string('Assistant')"/>
         <to part="RoleIN" variable="setApplicationReq"/>
        </copy>
        <copy>
         <from expression="string('CreateOffer')"/>
         <to part="ActionNameIN" variable="setApplicationReq"/>
        </copy>
       </assign>
       <invoke name="setApplication_OfferManagement"
         portType="agi:AgilproIssuer" operation="setApplication"
         partnerLink="agilpro" inputVariable="setApplicationReq">
```

```
160          <correlations>
              <correlation pattern="out" set="atmInteraction"/>
             </correlations>
            </invoke>
            <assign name="set_DTO_Offerheader">
165          <copy>
              <from part="Ticketnumber" variable="nextActionReq"/>
              <to part="Ticketnumber" variable="setValueToObjectReq"/>
             </copy>
             <copy>
170          <from expression="string(
              'eu.emundo.agilpro.fw.fe.erp.intf.AngebotPanelEn')"/>
              <to part="DataTypeIN" variable="setValueToObjectReq"/>
             </copy>
             <copy>
175          <from expression="string('TAB_ANGEBOT_KOPFDATEN')"/>
              <to part="ValueIN" variable="setValueToObjectReq"/>
             </copy>
             <copy>
              <from expression="string('Offerheader')"/>
180          <to part="NameIN" variable="setValueToObjectReq"/>
             </copy>
             <copy>
              <from expression="number('-1')"/>
              <to part="HashCodeIN" variable="setValueToObjectReq"/>
185          </copy>
            </assign>
            <invoke name="setValueToObject_Offerheader"
              portType="agi:AgilproIssuer"
              operation="setValueToObject"
190           partnerLink="agilpro"
              inputVariable="setValueToObjectReq">
             <correlations>
              <correlation pattern="out" set="atmInteraction"/>
             </correlations>
195         </invoke>
            <assign name="set_DTO_OfferID">
             <copy>
              <from part="Ticketnumber" variable="nextActionReq"/>
              <to part="Ticketnumber" variable="setValueToObjectReq"/>
200          </copy>
             <copy>
              <from expression="string('ID')"/>
              <to part="DataTypeIN" variable="setValueToObjectReq"/>
             </copy>
205          <copy>
              <from expression="string('0')"/>
              <to part="ValueIN" variable="setValueToObjectReq"/>
             </copy>
             <copy>
210          <from expression="string('OfferID')"/>
              <to part="NameIN" variable="setValueToObjectReq"/>
             </copy>
             <copy>
              <from expression="number('-1')"/>
215           <to part="HashCodeIN" variable="setValueToObjectReq"/>
             </copy>
            </assign>
            <invoke name="setValueToObject_OfferID"
```

```
        portType="agi:AgilproIssuer"
        operation="setValueToObject"
        partnerLink="agilpro"
        inputVariable="setValueToObjectReq">
       <correlations>
        <correlation pattern="out" set="atmInteraction"/>
       </correlations>
      </invoke>
      <assign name="startAction_CreateOffer">
       <copy>
        <from part="Ticketnumber" variable="nextActionReq"/>
        <to part="Ticketnumber" variable="startActionReq"/>
       </copy>
      </assign>
      <invoke name="startAction_CreateOffer"
        portType="agi:AgilproIssuer" operation="startAction"
        partnerLink="agilpro" inputVariable="startActionReq">
       <correlations>
        <correlation pattern="out" set="atmInteraction"/>
       </correlations>
      </invoke>
      <receive>
        partnerLink="atm" variable="nextActionReq"
        portType="atm:FrontEnd" operation="nextAction">
       <correlations>
        <correlation set="atmInteraction"/>
       </correlations>
      </receive>
      <assign name="endAction_CreateOffer">
       <copy>
        <from part="Ticketnumber" variable="nextActionReq"/>
        <to part="Ticketnumber" variable="endActionReq"/>
       </copy>
      </assign>
      <invoke name="endAction_CreateOffer"
        portType="agi:AgilproIssuer" operation="endAction"
        partnerLink="agilpro" inputVariable="endActionReq">
       <correlations>
        <correlation pattern="out" set="atmInteraction"/>
       </correlations>
      </invoke>
      <assign name="get_DTO_Offer">
       <copy>
        <from part="Ticketnumber"
          variable="nextActionReq"/>
        <to part="Ticketnumber"
         variable="getValueFromObjectReq"/>
       </copy>
       <copy>
        <from expression="string(
         'eu.emundo.agilpro.fw.fe.dto.AngebotDTO')"/>
        <to part="DataTypeIN" variable="getValueFromObjectReq"/>
       </copy>
       <copy>
        <from expression="string('102')"/>
        <to part="ValueIN" variable="getValueFromObjectReq"/>
       </copy>
       <copy>
        <from expression="string('Offer')"/>
```

```
        <to part="NameIN" variable="getValueFromObjectReq"/>
       </copy>
      </assign>
      <invoke name="getValueFromObject_Offer"
        portType="agi:AgilproIssuer"
        operation="getValueFromObject"
        partnerLink="agilpro"
        inputVariable="getValueFromObjectReq"
        outputVariable="getValueFromObjectRes">
       <correlations>
        <correlation pattern="out" set="atmInteraction"/>
       </correlations>
      </invoke>
     </sequence>
    </scope>
    <scope name="AddOfferPosition">
     <sequence>
      <empty/>
      <assign name="set_App_OfferManagement">
       <copy>
        <from part="Ticketnumber" variable="nextActionReq"/>
        <to part="Ticketnumber" variable="setApplicationReq"/>
       </copy>
       <copy>
        <from expression="string(
         'eu.emundo.agilpro.fw.fe.erp.intf.AngebotUi')"/>
        <to part="JavaClassIN" variable="setApplicationReq"/>
       </copy>
       <copy>
        <from expression="string('Assistant')"/>
        <to part="RoleIN" variable="setApplicationReq"/>
       </copy>
       <copy>
        <from expression="string('AddOfferPosition')"/>
        <to part="ActionNameIN" variable="setApplicationReq"/>
       </copy>
      </assign>
      <invoke name="setApplication_OfferManagement"
        portType="agi:AgilproIssuer"
        operation="setApplication"
        partnerLink="agilpro"
        inputVariable="setApplicationReq">
       <correlations>
        <correlation pattern="out" set="atmInteraction"/>
       </correlations>
      </invoke>
      <assign name="set_DTO_Offer">
       <copy>
        <from part="Ticketnumber" variable="nextActionReq"/>
        <to part="Ticketnumber" variable="setValueToObjectReq"/>
       </copy>
       <copy>
        <from expression="string(
         'eu.emundo.agilpro.fw.fe.dto.AngebotDTO')"/>
        <to part="DataTypeIN" variable="setValueToObjectReq"/>
       </copy>
       <copy>
        <from expression="string('102')"/>
        <to part="ValueIN" variable="setValueToObjectReq"/>
```

```
        </copy>
        <copy>
         <from expression="string('Offer')"/>
         <to part="NameIN" variable="setValueToObjectReq"/>
        </copy>
        <copy>
         <from expression="number('-1')"/>
         <to part="HashCodeIN" variable="setValueToObjectReq"/>
        </copy>
       </assign>
       <invoke name="setValueToObject_Offer"
        portType="agi:AgilproIssuer"
        operation="setValueToObject"
        partnerLink="agilpro"
        inputVariable="setValueToObjectReq">
        <correlations>
         <correlation pattern="out" set="atmInteraction"/>
        </correlations>
       </invoke>
       <assign name="set_DTO_Offerposition">
        <copy>
         <from part="Ticketnumber" variable="nextActionReq"/>
         <to part="Ticketnumber" variable="setValueToObjectReq"/>
        </copy>
        <copy>
         <from expression="string(
          'eu.emundo.agilpro.fw.fe.erp.intf.AngebotPanelEn')"/>
         <to part="DataTypeIN" variable="setValueToObjectReq"/>
        </copy>
        <copy>
         <from expression="string('TAB_ANGEBOT_POSITIONEN')"/>
         <to part="ValueIN" variable="setValueToObjectReq"/>
        </copy>
        <copy>
         <from expression="string('Offerposition')"/>
         <to part="NameIN" variable="setValueToObjectReq"/>
        </copy>
        <copy>
         <from expression="number('-1')"/>
         <to part="HashCodeIN" variable="setValueToObjectReq"/>
        </copy>
       </assign>
       <invoke name="setValueToObject_Offerposition"
        partnerLink="agilpro"
        operation="setValueToObject"
        portType="agi:AgilproIssuer"
        inputVariable="setValueToObjectReq">
        <correlations>
         <correlation pattern="out" set="atmInteraction"/>
        </correlations>
       </invoke>
       <assign name="startAction_AddOfferPosition">
        <copy>
         <from part="Ticketnumber" variable="nextActionReq"/>
         <to part="Ticketnumber" variable="startActionReq"/>
        </copy>
       </assign>
       <invoke name="startAction_AddOfferPosition"
        portType="agi:AgilproIssuer"
```

```
        operation="startAction"
        partnerLink="agilpro"
        inputVariable="startActionReq">
       <correlations>
400     <correlation pattern="out" set="atmInteraction"/>
       </correlations>
      </invoke>
      <receive
        partnerLink="atm" variable="nextActionReq"
405     portType="atm:FrontEnd" operation="nextAction">
       <correlations>
        <correlation set="atmInteraction"/>
       </correlations>
      </receive>
410   <assign name="endAction_AddOfferPosition">
       <copy>
        <from part="Ticketnumber" variable="nextActionReq"/>
        <to part="Ticketnumber" variable="endActionReq"/>
       </copy>
415   </assign>
      <invoke name="endAction_AddOfferPosition"
        portType="agi:AgilproIssuer"
        operation="endAction"
        partnerLink="agilpro"
420     inputVariable="endActionReq">
       <correlations>
        <correlation pattern="out" set="atmInteraction"/>
       </correlations>
      </invoke>
425  </sequence>
    </scope>
    <assign name="checkConditions">
     <copy>
      <from part="Ticketnumber" variable="nextActionReq"/>
430   <to part="Ticketnumber" variable="checkGuardReq"/>
     </copy>
     <copy>
      <from expression="string('Offer')"/>
      <to part="DataIN" variable="checkGuardReq"/>
435  </copy>
     <copy>
      <from expression="string('Offer')"/>
      <to part="AttributeIN" variable="checkGuardReq"/>
     </copy>
440  <copy>
      <from expression="string('&gt;=')"/>
      <to part="OperationIN" variable="checkGuardReq"/>
     </copy>
     <copy>
445   <from expression="string('1000')"/>
      <to part="ValueIN" variable="checkGuardReq"/>
     </copy>
    </assign>
    <invoke name="invokeGuardCheck"
450   portType="agi:AgilproIssuer" operation="checkGuard"
      partnerLink="agilpro"
      inputVariable="checkGuardReq"
      outputVariable="checkGuardRes">
     <correlations>
```

```
455       <correlation pattern="out" set="atmInteraction"/>
        </correlations>
      </invoke>
      <switch name="Defaultname">
       <case condition="bpel:getVariableData('checkGuardRes',
460                                            'ValueOUT')=1">
        <sequence>
         <scope name="CheckOfferConditions">
          <sequence>
           <empty/>
465        <assign name="set_App_PDFViewer">
            <copy>
             <from part="Ticketnumber"
              variable="nextActionReq"/>
             <to part="Ticketnumber"
470           variable="setApplicationReq"/>
            </copy>
            <copy>
             <from expression="string(
              'eu.emundo.agilpro.fw.fe.intf.AcrobatUi')"/>
475          <to part="JavaClassIN" variable="setApplicationReq"/>
            </copy>
            <copy>
             <from expression="string('HeadofSalesDepartment')"/>
             <to part="RoleIN" variable="setApplicationReq"/>
480         </copy>
            <copy>
             <from expression="string('CheckOfferConditions')"/>
             <to part="ActionNameIN"
              variable="setApplicationReq"/>
485         </copy>
           </assign>
           <invoke name="setApplication_PDFViewer"
            portType="agi:AgilproIssuer"
            operation="setApplication"
490         partnerLink="agilpro"
            portType="agi:AgilproIssuer"
            inputVariable="setApplicationReq">
            <correlations>
             <correlation pattern="out" set="atmInteraction"/>
495         </correlations>
           </invoke>
           <assign name="set_DTO_Offer">
            <copy>
             <from part="Ticketnumber"
500           variable="nextActionReq"/>
             <to part="Ticketnumber"
              variable="setValueToObjectReq"/>
            </copy>
            <copy>
505          <from expression="string(
              'eu.emundo.agilpro.fw.fe.dto.AngebotDTO')"/>
             <to part="DataTypeIN"
              variable="setValueToObjectReq"/>
            </copy>
510         <copy>
             <from expression="string('102')"/>
             <to part="ValueIN" variable="setValueToObjectReq"/>
            </copy>
```

```
          <copy>
515        <from expression="string('Offer')"/>
           <to part="NameIN" variable="setValueToObjectReq"/>
          </copy>
          <copy>
           <from expression="number('-1')"/>
520        <to part="HashCodeIN"
            variable="setValueToObjectReq"/>
          </copy>
         </assign>
         <invoke name="setValueToObject_Offer"
525       portType="agi:AgilproIssuer"
          operation="setValueToObject"
          partnerLink="agilpro"
          portType="agi:AgilproIssuer"
          inputVariable="setValueToObjectReq">
530       <correlations>
           <correlation pattern="out" set="atmInteraction"/>
          </correlations>
         </invoke>
         <assign name="set_DTO_Offerprint">
535       <copy>
           <from part="Ticketnumber"
            variable="nextActionReq"/>
           <to part="Ticketnumber"
            variable="setValueToObjectReq"/>
540       </copy>
          <copy>
           <from expression="string('filename')"/>
           <to part="DataTypeIN"
            variable="setValueToObjectReq"/>
545       </copy>
          <copy>
           <from expression="string(
            '../agilio/demodaten/angebot5005.pdf')"/>
           <to part="ValueIN" variable="setValueToObjectReq"/>
550       </copy>
          <copy>
           <from expression="string('Offerprint')"/>
           <to part="NameIN" variable="setValueToObjectReq"/>
          </copy>
555       <copy>
           <from expression="number('-1')"/>
           <to part="HashCodeIN"
            variable="setValueToObjectReq"/>
          </copy>
560      </assign>
         <invoke name="setValueToObject_Offerprint"
           portType="agi:AgilproIssuer"
           operation="setValueToObject"
           partnerLink="agilpro"
565        inputVariable="setValueToObjectReq">
          <correlations>
           <correlation pattern="out" set="atmInteraction"/>
          </correlations>
         </invoke>
570      <assign name="startAction_CheckOfferConditions">
          <copy>
           <from part="Ticketnumber" variable="nextActionReq"/>
```

```
          <to part="Ticketnumber" variable="startActionReq"/>
         </copy>
        </assign>
        <invoke name="startAction_CheckOfferConditions"
          portType="agi:AgilproIssuer"
          operation="startAction"
          partnerLink="agilpro"
          inputVariable="startActionReq">
         <correlations>
          <correlation pattern="out" set="atmInteraction"/>
         </correlations>
        </invoke>
        <receive
          portType="atm:FrontEnd" operation="nextAction"
          partnerLink="atm" variable="nextActionReq">
         <correlations>
          <correlation set="atmInteraction"/>
         </correlations>
        </receive>
        <assign name="endAction_CheckOfferConditions">
         <copy>
          <from part="Ticketnumber" variable="nextActionReq"/>
          <to part="Ticketnumber" variable="endActionReq"/>
         </copy>
        </assign>
        <invoke name="endAction_CheckOfferConditions"
          portType="agi:AgilproIssuer"
          operation="endAction"
          partnerLink="agilpro"
          inputVariable="endActionReq">
         <correlations>
          <correlation pattern="out" set="atmInteraction"/>
         </correlations>
        </invoke>
       </sequence>
      </scope>
      <empty/>
     </sequence>
    </case>
    <case condition="bpel:getVariableData('checkGuardRes',
                                          'ValueOUT')=0">
     <sequence>
      <empty/>
     </sequence>
    </case>
   </switch>
   <scope name="AddOfferToB2BPortal">
    <sequence>
     <empty/>
     <assign name="set_App_WebBrowser">
      <copy>
       <from part="Ticketnumber" variable="nextActionReq"/>
       <to part="Ticketnumber" variable="setApplicationReq"/>
      </copy>
      <copy>
       <from expression="string(
       'eu.emundo.agilpro.fw.fe.intf.BrowserUi')"/>
       <to part="JavaClassIN" variable="setApplicationReq"/>
      </copy>
```

```
        <copy>
         <from expression="string('Assistant')"/>
         <to part="RoleIN" variable="setApplicationReq"/>
        </copy>
        <copy>
         <from expression="string('AddOfferToB2BPortal')"/>
         <to part="ActionNameIN" variable="setApplicationReq"/>
        </copy>
       </assign>
       <invoke name="setApplication_WebBrowser"
        portType="agi:AgilproIssuer"
        operation="setApplication"
        partnerLink="agilpro"
        inputVariable="setApplicationReq">
        <correlations>
         <correlation pattern="out" set="atmInteraction"/>
        </correlations>
       </invoke>
       <assign name="set_DTO_SellerPortal">
        <copy>
         <from part="Ticketnumber" variable="nextActionReq"/>
         <to part="Ticketnumber" variable="setValueToObjectReq"/>
        </copy>
        <copy>
         <from expression="string('URL')"/>
         <to part="DataTypeIN" variable="setValueToObjectReq"/>
        </copy>
        <copy>
         <from expression="string(
          '\Projekte\AgilPro\eclipse_workspace\Agilio
          \demodaten\click2procure.html')"/>
         <to part="ValueIN" variable="setValueToObjectReq"/>
        </copy>
        <copy>
         <from expression="string('SellerPortal')"/>
         <to part="NameIN" variable="setValueToObjectReq"/>
        </copy>
        <copy>
         <from expression="number('-1')"/>
         <to part="HashCodeIN" variable="setValueToObjectReq"/>
        </copy>
       </assign>
       <invoke name="setValueToObject_SellerPortal"
        portType="agi:AgilproIssuer"
        operation="setValueToObject"
        partnerLink="agilpro"
        inputVariable="setValueToObjectReq">
        <correlations>
         <correlation pattern="out" set="atmInteraction"/>
        </correlations>
       </invoke>
       <assign name="set_DTO_Offer">
        <copy>
         <from part="Ticketnumber" variable="nextActionReq"/>
         <to part="Ticketnumber" variable="setValueToObjectReq"/>
        </copy>
        <copy>
         <from expression="string(
          'eu.emundo.agilpro.fw.fe.dto.AngebotDTO')"/>
```

```
        <to part="DataTypeIN" variable="setValueToObjectReq"/>
       </copy>
       <copy>
        <from expression="string('102')"/>
        <to part="ValueIN" variable="setValueToObjectReq"/>
       </copy>
       <copy>
        <from expression="string('Offer')"/>
        <to part="NameIN" variable="setValueToObjectReq"/>
       </copy>
       <copy>
        <from expression="number('-1')"/>
        <to part="HashCodeIN" variable="setValueToObjectReq"/>
       </copy>
      </assign>
      <invoke name="setValueToObject_Offer"
        portType="agi:AgilproIssuer"
        operation="setValueToObject"
        partnerLink="agilpro"
        inputVariable="setValueToObjectReq">
       <correlations>
        <correlation pattern="out" set="atmInteraction"/>
       </correlations>
      </invoke>
      <assign name="startAction_AddOfferToB2BPortal">
       <copy>
        <from part="Ticketnumber" variable="nextActionReq"/>
        <to part="Ticketnumber" variable="startActionReq"/>
       </copy>
      </assign>
      <invoke name="startAction_AddOfferToB2BPortal"
        portType="agi:AgilproIssuer"
        operation="startAction"
        partnerLink="agilpro"
        inputVariable="startActionReq">
       <correlations>
        <correlation pattern="out" set="atmInteraction"/>
       </correlations>
      </invoke>
      <receive variable="nextActionReq"
        portType="atm:FrontEnd" operation="nextAction"
        partnerLink="atm">
       <correlations>
        <correlation set="atmInteraction"/>
       </correlations>
      </receive>
      <assign name="endAction_AddOfferToB2BPortal">
       <copy>
        <from part="Ticketnumber" variable="nextActionReq"/>
        <to part="Ticketnumber" variable="endActionReq"/>
       </copy>
      </assign>
      <invoke name="endAction_AddOfferToB2BPortal"
        portType="agi:AgilproIssuer"
        operation="endAction"
        partnerLink="agilpro"
        inputVariable="endActionReq">
       <correlations>
        <correlation pattern="out" set="atmInteraction"/>
```

```
      </correlations>
     </invoke>
    </sequence>
   </scope>
   <scope name="Defaultname">
    <sequence>
     <empty/>
    </sequence>
   </scope>
   <assign>
    <copy>
     <from part="Ticketnumber" variable="nextActionReq"/>
     <to part="Ticketnumber" variable="disconnectAgilproReq"/>
    </copy>
   </assign>
   <invoke name="disconnectAgilpro"
     portType="agi:AgilproIssuer"
     operation="disconnectAgilpro"
     partnerLink="agilpro"
     inputVariable="disconnectAgilproReq"
     outputVariable="disconnectAgilproRes">
    <correlations>
     <correlation pattern="out" set="atmInteraction"/>
    </correlations>
   </invoke>
  </sequence>
 </scope>
</sequence>
</process>
```

# Appendix B

# CBP Architecture Evaluation

## B.1 Scenarios Descriptions

| **Scenario 1 – Modification of CBPs** | |
|---|---|
| *Source* | Management |
| *Stimulus* | Due to the constant and rapid change in business existing CBPs have to be adapted to the new business models. |
| *Environment* | Design-time |
| *Artifact* | Cross-organisational business process |
| *Response* | The necessary changes in order to enact the new CBP affect a minimal number of existing modules. Necessary change of existing modules should have no side-effects on other processes (e.g CBPs). |
| *Response Measure* | *Brokerless:* up to n ESs of the partners are affected<br>*Central broker:* the central broker is affected<br>*Decentral broker:* VPs of the respective partner(s) are affected |

Table B.1: Scenario 1: modification of CBPs

| **Scenario 2: Change of partners in CBP** | |
|---|---|
| *Source* | Management |
| *Stimulus* | New partners have to get involved in the CPB which can have different ESs compared to the previous partners. |
| *Environment* | Design-time |
| *Artifact* | Cross-organisational business process |
| *Response* | Minimal effort is needed for integrating the new partner in the CBP. The ESs of the other partners have not to be changed. Minimal and only local (only a small defined part of the CBP has to be altered) changes to the CBP are necessary while the semantics of the CBP is maintained. |
| *Response Measure* | *Brokerless:* multiple ESs (of partners) are affected<br>*Central broker:* central CBP process realization or ESs of new partners has to be adjusted<br>*Decentral broker:* integration affects only VP of the new partner |

Table B.2: Scenario 2: change of partners in CBP

| **Scenario 3: Incremental development of CBPs** | |
|---|---|
| *Source* | Management (, Architects) |
| *Stimulus* | An organisation, which wishes to participate in a CBP, asks for an incremental development and enactment of CBPs. A minimal system shall be designed and enacted early, while further functions are added later. |
| *Environment* | Design-time |
| *Artifact* | Cross-organisational business process |
| *Response* | The processes (especially ESs) of the other collaboration parterns are not affected by the incremental CBP development. Changes to the already implemented part of CBP are as small as possible when developing a new increment. |
| *Response Measure* | *Brokerless:* multiple ESs of partners are affected by each change<br>*Central broker:* ESs are not affected, but central CBP realization<br>*Decentral broker:* only the own VP has to be adjusted |

Table B.3: Scenario 3: incremental development of CBPs

| **Scenario 4: Reuse of CBPs** | |
|---|---|
| *Source* | Management, Architects |
| *Stimulus* | Wish that CBP descriptions or parts of them can be reused in different contexts. An organisation participates in various similar CBPs and can reuse its parts of its CBP implementations. |
| *Environment* | Design-time |
| *Artifact* | Cross-organisational business process |
| *Response* | Number of processes (CBP realizations) which have to be supported by an organisation have to be minimized. |
| *Response Measure* | *Brokerless:* Normally weak; only good, if ESs can be reused (degree of standardization)<br>*Central broker:* Good, but CBP cannot be decoupled; thus only the whole CBP can be reused<br>*Decentral broker:* CBP(VPs) are decoupled and does not depend on ESs |

Table B.4: Scenario 4: reuse of CBPs

| **Scenario 5: Change of CBP protocol specification** | |
|---|---|
| *Source* | Management, Architects |
| *Stimulus* | The protocol specification of a CBP changes (data types and/or control flow). |
| *Environment* | Design-time (and run-time) |
| *Artifact* | Cross-organisational business process |
| *Response* | There should be no or minimal side-effects or changes to an organisation's internal ESs. |
| *Response Measure* | *Brokerless:* ESs have to be adjusted<br>*Central broker:* The central broker process has to be adjusted<br>*Decentral broker:* Dependent where the changes occur, one or more VPs have to be adjusted |

Table B.5: Scenario 5: change of CBP protocol specification

| **Scenario 6: Change of elementary services** | |
|---|---|
| *Source* | Management, Architects |
| *Stimulus* | An elementary service of an enterprise changes. This can comprise the functionality but also the behaviour (represented by the ES's abstract process) of the ES. |
| *Environment* | Design-time |
| *Artifact* | Elementary service |
| *Response* | There should be no side-effects to the ESs of the other partners. The changes to the CBPs should be minimal and as local as possible (only a small defined part of the CBP has to be altered). |
| *Response Measure* | *Brokerless:* All processes using a replaced ES need to get the knowledge about the new identity of the new ES. In the case of changes to one process's syntax or semantics, the other ESs have to be further adjusted<br>*Central broker:* The central broker hides the identity of ESs from other ESs. Changes have to be made only in the central broker<br>*Decentral broker:* The decentral broker hides the identity of ESs from the other ESs and other organisations. Changes have to be made only to one VP of the decentral broker |

Table B.6: Scenario 6: change of elementary services

| **Scenario 7: Privacy of internal ESs related data** | |
|---|---|
| *Source* | Management |
| *Stimulus* | Wish that no information about organisational internal data structures is exposed to partners. |
| *Environment* | Design-time (and run-time) |
| *Artifact* | Data structures used by ESs |
| *Response* | Only data structures are exposed which are needed or defined by the CBP protocol. Specific organisational internal data structures are not to be exhibited. |
| *Response Measure* | *Brokerless:* This approach is the better, the more the data structures, which have to be exchanged, are standardized<br>*Central broker:* This approach depends on who is responsible to control the broker<br>*Decentral broker:* Internal data can be hidden by VPs |

Table B.7: Scenario 7: privacy of internal ESs related data

| **Scenario 8: Privacy of internal CBPs realizations** | |
|---|---|
| *Source* | Management |
| *Stimulus* | Wish that partners cannot see how an organisation has realized its part of the CBP. |
| *Environment* | Design-time (and run-time) |
| *Artifact* | Structure of ESs |
| *Response* | Organisations participating an CBP get no or minimal information about how the other organisations realize their part of the CBP and structure their ESs. |
| *Response Measure* | *Brokerless:* This approach works better, the more the process is standardized<br>*Central broker:* It is likely that internal realization is exhibited to the broker<br>*Decentral broker:* Internal CBP realizations can be hidden by VPs |

Table B.8: Scenario 8: privacy of internal CBPs realizations

| **Scenario 9: Reuse of elementary services** | |
|---|---|
| *Source* | Management |
| *Stimulus* | Wish that elementary services can easily be reused in different CBP and in different contexts, respectively. |
| *Environment* | Design-time |
| *Artifact* | Elementary service |
| *Response* | Minimal effort has to be made to use ESs in different CBPs and contexts. There has to be maintained only a small number of variants for one ES, so that its implemenation can be easily replaced by other applications. |
| *Response Measure* | *Brokerless:* Weak, since an ES has to fit to other ESs in different contexts<br>*Central broker:* Central broker process can be used for integrating ESs<br>*Decentral broker:* VPs can be used for integration ESs. Integration can be realized with in the organisation (its VP) which wants to reuse ESs |

Table B.9: Scenario 9: reuse of elementary services

| **Scenario 10: Change of ES's interfaces** | |
|---|---|
| *Source* | (Management,) Architects |
| *Stimulus* | The interfaces (e.g. data types) of an organisation's ESs have to be altered. |
| *Environment* | Design-time (and run-time) |
| *Artifact* | Elementary service |
| *Response* | There should be no side-effects to the ESs of the other partners. The changes to the CBPs should be minimal and as local as possible (only a small defined part of the CBP has to be altered). |
| *Response Measure* | *Brokerless:* All processes using an replaced ES need to be adjusted to the new syntax (interface) of the new process<br>*Central broker:* Changes have to be made only in the central broker<br>*Decentral broker:* Changes have to be made only to one VP of the decentral broker |

Table B.10: Scenario 10: change of ES's interfaces

| **Scenario 11: Development of CBP variants** | |
|---|---|
| *Source* | Management |
| *Stimulus* | Wish to include and exclude various ESs or to choose different versions of ESs to customize a CBP for specific collaborations. |
| *Environment* | Design- and build-time |
| *Artifact* | CBP design process and models |
| *Response* | Including or excluding certain ESs has no side effects on the other ESs and CBPs. A minimal number of redundant models has to be maintained during design. |
| *Response Measure* | *Brokerless:* many variants of ESs have to be maintained; variant of one partner can cause variants for other partners; nevertheless for short process with low complexity it may be an efficienty way to realize small variations<br>*Central broker:* ESs are decoupled; new CBP variant is necesarry for each partner wishing an own variant<br>*Decentral broker:* ESs and CBP(VPs) are decoupled as much as possible |

Table B.11: Scenario 11: development of CBP variants

| **Scenario 12: Bottle-neck** | |
|---|---|
| *Source* | Management, System Administrators |
| *Stimulus* | Wish that the systems enacting the CBPs should be able to cope with a high number of process instances, high traffic volume, and a high number of messages exchanged between the process partners. |
| *Environment* | Run-time |
| *Artifact* | CBP & ES |
| *Response* | For relevant load and traffic volumes there exists no bottle-neck. |
| *Response Measure* | *Brokerless:* In principle this approach is good, since the communication is decentralized<br>*Central broker:* Due to the centralized broker component, there is the risk of a communication and data transfer bottle-neck<br>*Decentral broker:* This approach has no central component, but provides a single component for each organicsation and CBP |

Table B.12: Scenario 12: bottle-neck

| **Scenario 13: Security overhead** | |
|---|---|
| *Source* | Management, System Administrators |
| *Stimulus* | Security relevant attacks on the CBP and the ESs (inspection and alterning of exchange messages, intrusion, DoS) have to be repelled. |
| *Environment* | Run-time |
| *Artifact* | CBP & ES |
| *Response* | It is possible to undertake efficient actions to successfully prevent such attacks. The action have minimal effects on the performance of the system enacting the CBPs and ESs. |
| *Response Measure* | *Brokerless:* In this approach all messages have to be encrypted, since there are no private communication links<br>*Central broker:* All messages to the central broker have to be encrypted; with the central broker it may be easiser to manage the encryption keys<br>*Decentral broker:* Only message exchanged with other organisations have to be encrypted. For messages between VP and the ESs this may be not necessary |

Table B.13: Scenario 13: security overhead

| **Scenario 14: Versioning** | |
|---|---|
| *Source* | Management, System Administrators |
| *Stimulus* | The various processes and process versions, which are developed due to constant change of CBP, can be easily handled. |
| *Environment* | Development- & Run-time |
| *Artifact* | CBP & ES |
| *Response* | Effort undertaken to handle multiple versions of process and especially of process instances is minimal. |
| *Response Measure* | *Brokerless:* This totally decentralized approach requires powerful versioning mechnisms<br>*Central broker:* In a central component versioning is not so complex like in decentralized environments<br>*Decentral broker:* This approach needs a decentralized versioning mechnisms, but has to deal with much less decentralized entities as the first approach |

Table B.14: Scenario 14: versioning

| **Scenario 15: Monitoring** | |
|---|---|
| *Source* | Management, System Administrators |
| *Stimulus* | Wish to monitor status and progress of the running process instances. |
| *Environment* | Run-time |
| *Artifact* | CBP & ES |
| *Response* | Possibility to monitor the running processes instances with minimal effort. |
| *Response Measure* | *Brokerless:* In this totally decentralized approach monitoring of the CBP progress can be difficult<br>*Central broker:* In this approach only one process instance has to be monitored in order to retrieve the status of CBP execution<br>*Decentral broker:* This approach needs to monitor the result of multiple instances, but does have to collect data from less instances like the first approach |

Table B.15: Scenario 15: monitoring

## B.2 Influences of Contingencies

| | | Scenario 1 | Scenario 2 | Scenario 3 | Scenario 4 | Scenario 5 |
|---|---|---|---|---|---|---|
| Internal Contingencies | Coll. topology | initiator ⇈ | power of players ⇈ | n/a | n/a | n/a |
| | Product | complexity ⇈ | complexity ⇈ | n/a | complexity/ specifity ↑↓ | n/a |
| | Service flow | n/a | n/a | n/a | n/a | ⇈ |
| | Process | n/a | length ⇈ | # process instances ⇈ | length ⇈ | length ⇈ |
| External Contingencies | Standardization | ↑↓ | ↑↓ | ↑↓ | ↑↓ | ↑↓ |
| | Maturity | ↑↓ | ↑↓ | ↑↓ | ↑↓ | ↑↓ |
| | Bus. semantics | n/a | ↑↓ | ↑↓ | ↑↓ | ↑↓ |
| | Legislation | n/a | n/a | n/a | n/a | n/a |

Table B.16: Influence of contingencies on scenario ratings 1-5

| | | Scenario 6 | Scenario 7 | Scenario 8 | Scenario 9 | Scenario 10 |
|---|---|---|---|---|---|---|
| Internal Contingencies | Coll. topology | n/a | power of players ⇈ | n/a | n/a | n/a |
| | Product | n/a | n/a | n/a | complexity ↑↓ | n/a |
| | Service flow | ⇈ | n/a | n/a | n/a | n/a |
| | Process | n/a | n/a | n/a | n/a | n/a |
| External Contingencies | Standardization | ↑↓ | ↑↓ | ↑↓ | ↑↓ | ↑↓ |
| | Maturity | ↑↓ | n/a | n/a | ↑↓ | n/a |
| | Bus. semantics | ↑↓ | ↑↓ | n/a | ↑↓ | ↑↓ |
| | Legislation | n/a | n/a | n/a | n/a | n/a |

Table B.17: Influence of contingencies on scenario ratings 6-10

| | | Scenario 11 | Scenario 12 | Scenario 13 | Scenario 14 | Scenario 15 |
|---|---|---|---|---|---|---|
| Internal Contingencies | Coll. topology | n/a | n/a | n/a | n/a | n/a |
| | Product | compl. ⇈ | n/a | n/a | n/a | n/a |
| | Service flow | n/a | ⇈ | ⇈ | ⇈ | ⇈ |
| | Process | length ⇈ | # process instances ⇈ | # process instances ⇈ | # process instances ⇈ | # process instances ⇈ |
| External Contingencies | Standardization | ↑↓ | n/a | n/a | ↑↓ | n/a |
| | Maturity | n/a | n/a | n/a | ↑↓ | n/a |
| | Bus. semantics | n/a | n/a | n/a | ↑↓ | n/a |
| | Legislation | n/a | n/a | ⇈ | n/a | ⇈ |

Table B.18: Influence of contingencies on scenario ratings 11-15

## B.3 Virtual Enterprise Scenario

### Determining the Qualitative Measure - Weighting Subjective Factors

| | mod. | pri. | reuse | int. | eff. | man. | $v_i$ |
|---|---|---|---|---|---|---|---|
| modifiability | 1 | 7 | 3 | $\frac{1}{3}$ | 3 | 3 | 0.21 |
| privacy | $\frac{1}{7}$ | 1 | $\frac{1}{4}$ | $\frac{1}{9}$ | $\frac{1}{5}$ | $\frac{1}{5}$ | 0.03 |
| reuse | $\frac{1}{3}$ | 4 | 1 | $\frac{1}{5}$ | 1 | 1 | 0.10 |
| interoperability | 3 | 9 | 5 | 1 | 5 | 5 | 0.45 |
| efficiency | $\frac{1}{3}$ | 5 | 1 | $\frac{1}{5}$ | 1 | 1 | 0.10 |
| managability | $\frac{1}{3}$ | 5 | 1 | $\frac{1}{5}$ | 1 | 1 | 0.10 |
| | | | | | | | $\lambda_{max} = 6.172, CR = 0.028$ |

Table B.19: Priority comparison matrix for the first level factors

| | sc. 1 | sc. 2 | sc. 3 | sc. 6 | sc. 11 | $v_i$ |
|---|---|---|---|---|---|---|
| scenario 1 | 1 | 3 | 7 | $\frac{1}{5}$ | $\frac{1}{3}$ | 0.14 |
| scenario 2 | $\frac{1}{3}$ | 1 | 5 | $\frac{1}{5}$ | $\frac{1}{5}$ | 0.08 |
| scenario 3 | $\frac{1}{5}$ | $\frac{1}{7}$ | 1 | $\frac{1}{9}$ | $\frac{1}{9}$ | 0.03 |
| scenario 6 | 5 | 5 | 9 | 1 | 3 | 0.47 |
| scenario 11 | 3 | 5 | 9 | $\frac{1}{3}$ | 1 | 0.27 |
| | | | | | | $\lambda_{max} = 5.392, CR = 0.088$ |

Table B.20: Priority comparison matrix for second level factor modifiability

| | sc. 7 | sc. 8 | $v_i$ |
|---|---|---|---|
| scenario 7 | 1 | $\frac{1}{5}$ | 0.17 |
| scenatio 8 | 5 | 1 | 0.83 |
| | | | $\lambda_{max} = 2.000, CR = 0.000$ |

Table B.21: Priority comparison matrix for second level factor privacy

| | sc. 4 | sc. 9 | sc. 11 | $v_i$ |
|---|---|---|---|---|
| scenario 4 | 1 | 1 | 3 | 0.43 |
| scenario 9 | 1 | 1 | 3 | 0.43 |
| scenario 11 | $\frac{1}{3}$ | $\frac{1}{3}$ | 1 | 0.14 |
| $\lambda_{max} = 3.000, CR = 0.000$ | | | | |

Table B.22: Priority comparison matrix for second level factor reuse

| | sc. 5 | sc. 10 | $v_i$ |
|---|---|---|---|
| scenario 5 | 1 | $\frac{1}{5}$ | 0.17 |
| scenatio 10 | 5 | 1 | 0.83 |
| $\lambda_{max} = 2.000, CR = 0.000$ | | | |

Table B.23: Priority comparison matrix for second level factor interoperability

| | sc. 12 | sc. 13 | $v_i$ |
|---|---|---|---|
| scenario 12 | 1 | $\frac{1}{3}$ | 0.25 |
| scenatio 13 | 3 | 1 | 0.72 |
| $\lambda_{max} = 2.000, CR = 0.000$ | | | |

Table B.24: Priority comparison matrix for second level factor efficiency

| | sc. 14 | sc. 15 | $v_i$ |
|---|---|---|---|
| scenario 14 | 1 | 2 | 0.67 |
| scenatio 15 | $\frac{1}{2}$ | 1 | 0.33 |
| $\lambda_{max} = 2.000, CR = 0.000$ | | | |

Table B.25: Priority comparison matrix for second level factor manageability

### Determining the Qualitative Measure - Rating the Scenarios

| scenario 1 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | $\frac{1}{7}$ | $\frac{1}{7}$ | 0.14 |
| Cen-Br. | 7 | 1 | 1 | 1.00 |
| Dec-Br. | 7 | 1 | 1 | 1.00 |
| | | | $\lambda_{max} = 3.000, CR = 0.000$ | |

Table B.26: Rating scenario 1

| scenario 2 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | $\frac{1}{5}$ | $\frac{1}{7}$ | 0.11 |
| Cen-Br. | 5 | 1 | $\frac{1}{3}$ | 0.44 |
| Dec-Br. | 7 | 3 | 1 | 1.00 |
| | | | $\lambda_{max} = 3.065, CR = 0.062$ | |

Table B.27: Rating scenario 2

| scenario 3 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | $\frac{1}{5}$ | $\frac{1}{9}$ | 0.08 |
| Cen-Br. | 5 | 1 | $\frac{1}{5}$ | 0.30 |
| Dec-Br. | 9 | 5 | 1 | 1.00 |
| | | | $\lambda_{max} = 3.117, CR = 0.113$ | |

Table B.28: Rating scenario 3

| scenario 4 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | 1 | $\frac{1}{4}$ | 0.25 |
| Cen-Br. | 1 | 1 | $\frac{1}{4}$ | 0.25 |
| Dec-Br. | 4 | 4 | 1 | 1.00 |
| | | | $\lambda_{max} = 3.032, CR = 0.031$ | |

Table B.29: Rating scenario 4

| scenario 5 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | $\frac{1}{7}$ | $\frac{1}{7}$ | 0.14 |
| Cen-Br. | 7 | 1 | 1 | 1.00 |
| Dec-Br. | 7 | 1 | 1 | 1.00 |
| | | | $\lambda_{max} = 3.000, CR = 0.000$ | |

Table B.30: Rating scenario 5

| scenario 6 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | $\frac{1}{4}$ | $\frac{1}{5}$ | 0.16 |
| Cen-Br. | 4 | 1 | $\frac{1}{3}$ | 0.46 |
| Dec-Br. | 5 | 3 | 1 | 1.00 |
| | | | $\lambda_{max} = 3.039, CR = 0.038$ | |

Table B.31: Rating scenario 6

| scenario 7 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | 2 | $\frac{1}{5}$ | 0.24 |
| Cen-Br. | $\frac{1}{2}$ | 1 | $\frac{1}{6}$ | 0.14 |
| Dec-Br. | 5 | 6 | 1 | 1.00 |
| | | | $\lambda_{max} = 3.014, CR = 0.013$ | |

Table B.32: Rating scenario 7

| scenario 8 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | 1 | $\frac{1}{5}$ | 0.20 |
| Cen-Br. | 1 | 1 | $\frac{1}{5}$ | 0.20 |
| Dec-Br. | 5 | 5 | 1 | 1.00 |
| | | | $\lambda_{max} = 3.000, CR = 0.000$ | |

Table B.33: Rating scenario 8

| scenario 9 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | $\frac{1}{4}$ | $\frac{1}{5}$ | 0.17 |
| Cen-Br. | 4 | 1 | $\frac{1}{5}$ | 0.59 |
| Dec-Br. | 5 | 2 | 1 | 1.00 |
| | | | $\lambda_{max} = 3.025, CR = 0.024$ | |

Table B.34: Rating scenario 9

| scenario 10 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | $\frac{1}{3}$ | $\frac{1}{5}$ | 0.17 |
| Cen-Br. | 3 | 1 | $\frac{1}{3}$ | 0.41 |
| Dec-Br. | 5 | 3 | 1 | 1.00 |
| | | | $\lambda_{max} = 3.039, CR = 0.038$ | |

Table B.35: Rating scenario 10

| scenario 11 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | $\frac{1}{5}$ | $\frac{1}{9}$ | 0.08 |
| Cen-Br. | 7 | 1 | $\frac{1}{5}$ | 0.30 |
| Dec-Br. | 9 | 5 | 1 | 1.00 |
| | | | $\lambda_{max} = 3.117, CR = 0.113$ | |

Table B.36: Rating scenario 11

| scenario 12 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | 3 | 1 | 1.00 |
| Cen-Br. | $\frac{1}{3}$ | 1 | $\frac{1}{3}$ | 0.33 |
| Dec-Br. | 1 | 3 | 1 | 1.00 |
| | | | $\lambda_{max} = 3.000, CR = 0.000$ | |

Table B.37: Rating scenario 12

| scenario 13 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | $\frac{1}{5}$ | $\frac{1}{3}$ | 0.17 |
| Cen-Br. | 5 | 1 | 3 | 1.00 |
| Dec-Br. | 3 | $\frac{1}{3}$ | 1 | 0.41 |
| | | | $\lambda_{max} = 3.039, CR = 0.038$ | |

Table B.38: Rating scenario 13

| scenario 14 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | $\frac{1}{7}$ | $\frac{1}{6}$ | 0.12 |
| Cen-Br. | 7 | 1 | 2 | 1.00 |
| Dec-Br. | 6 | $\frac{1}{2}$ | 1 | 0.60 |
| | | | $\lambda_{max} = 3.032, CR = 0.031$ | |

Table B.39: Rating scenario 14

| scenario 15 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | $\frac{1}{7}$ | $\frac{1}{5}$ | 0.11 |
| Cen-Br. | 7 | 1 | 3 | 1.00 |
| Dec-Br. | 5 | $\frac{1}{3}$ | 1 | 0.44 |
| | | | $\lambda_{max} = 3.064, CR = 0.062$ | |

Table B.40: Rating scenario 15

### Determining the Quantitative Measure

$i = 0$ – brokerless architecture
$i = 1$ – central broker architecture
$i = 2$ – decentral broker architecture
discount rate $d = 5.0\%$
number of discounting periods $N = 10$

| **Software Licence** | | | | | | |
|---|---|---|---|---|---|---|
| *Purchase Software* | $C_{purchaseSWBr}$ | = | 0 | $n_{purchaseSWBr_0}$ | = | 0 |
| | | | | $n_{purchaseSWBr_1}$ | = | 1 |
| | | | | $n_{purchaseSWBr_2}$ | = | 3 |
| | $C_{purchaseSWES}$ | = | 500 | $n_{purchaseSWES_0}$ | = | 90 |
| | | | | $n_{purchaseSWES_1}$ | = | 90 |
| | | | | $n_{purchaseSWES_2}$ | = | 90 |
| *Annual Licences* | $C_{annualSWBr}$ | = | 3000 | $n_{annualSWBr_0}$ | = | 0 |
| | | | | $n_{annualSWBr_1}$ | = | 1 |
| | | | | $n_{annualSWBr_2}$ | = | 3 |
| | $C_{annualSWES}$ | = | 0 | $n_{annualSWES_0}$ | = | 90 |
| | | | | $n_{annualSWES_1}$ | = | 90 |
| | | | | $n_{annualSWES_2}$ | = | 90 |
| $FV^D_{0_1} = FV_{0_{11_0}} + \sum_{t=0}^{N-1} \frac{FV_{0_{12_t}}}{(1+d)^t}$ | | = | 45000 | | | |
| $FV^D_{1_1} = FV_{1_{11_0}} + \sum_{t=0}^{N-1} \frac{FV_{1_{12_t}}}{(1+d)^t}$ | | = | 69323 | | | |
| $FV^D_{2_1} = FV_{2_{11_0}} + \sum_{t=0}^{N-1} \frac{FV_{2_{12_t}}}{(1+d)^t}$ | | = | 117970 | | | |

| **Hardware** | | | | | | |
|---|---|---|---|---|---|---|
| *Purchase Hardware* | $C_{purchaseHWBr}$ | = | 20000 | $n_{purchaseHWBr_0}$ | = | 0 |
| | | | | $n_{purchaseHWBr_1}$ | = | 1 |
| | | | | $n_{purchaseHWBr_2}$ | = | 3 |
| | $C_{purchaseHWES}$ | = | 5000 | $n_{purchaseHWES_0}$ | = | 15 |
| | | | | $n_{purchaseHWES_1}$ | = | 15 |
| | | | | $n_{purchaseHWES_2}$ | = | 15 |
| *Lease Hardware* | $C_{leasingHWBr}$ | = | 0 | $n_{leasingHWBr_0}$ | = | 0 |
| | | | | $n_{leasingHWBr_1}$ | = | 1 |
| | | | | $n_{leasingHWBr_2}$ | = | 3 |
| | $C_{leasingHWES}$ | = | 0 | $n_{leasingHWES_0}$ | = | 15 |
| | | | | $n_{leasingHWES_1}$ | = | 15 |
| | | | | $n_{leasingHWES_2}$ | = | 15 |
| $FV^D_{0_2} = FV_{0_{21_0}} + \sum_{t=0}^{N-1} \frac{FV_{0_{22_t}}}{(1+d)^t}$ | | = | 75000 | | | |
| $FV^D_{1_2} = FV_{1_{21_0}} + \sum_{t=0}^{N-1} \frac{FV_{1_{22_t}}}{(1+d)^t}$ | | = | 95000 | | | |
| $FV^D_{2_2} = FV_{2_{21_0}} + \sum_{t=0}^{N-1} \frac{FV_{2_{22_t}}}{(1+d)^t}$ | | = | 135000 | | | |

## Labour

| | | | | | | |
|---|---|---|---|---|---|---|
| *Setup & maintenance* | $C_{setup_0}$ | = | 10000 | $C_{maintenance_0}$ | = | 10000 |
| | $C_{setup_1}$ | = | 25000 | $C_{maintenance_1}$ | = | 20000 |
| | $C_{setup_2}$ | = | 30000 | $C_{maintenance_2}$ | = | 30000 |
| *Change Broker* | $C_{changeBr_0}$ | = | 0 | $n_{changeBr_0}$ | = | 0 |
| | $C_{changeBr_1}$ | = | 2500 | $n_{changeBr_1}$ | = | 1 |
| | $C_{changeBr_2}$ | = | 1000 | $n_{changeBr_2}$ | = | 2.7 |
| *Change ES* | $C_{changeES_0}$ | = | 200 | $n_{changeES_0}$ | = | 60 |
| | $C_{changeES_1}$ | = | 0 | $n_{changeES_1}$ | = | 0 |
| | $C_{changeES_2}$ | = | 0 | $n_{changeES_2}$ | = | 0 |
| *Frequency of Change* | $n_{change}$ | = | 50 | | | |

$$FV^D_{0_3} = FV_{0_{31_0}} + \sum_{t=0}^{N-1} \frac{FV_{0_{32_t}}}{(1+d)^t} + \sum_{t=0}^{N-1} \frac{FV_{0_{33_t}}}{(1+d)^t} = 4955771$$

$$FV^D_{1_3} = FV_{1_{31_0}} + \sum_{t=0}^{N-1} \frac{FV_{1_{32_t}}}{(1+d)^t} + \sum_{t=0}^{N-1} \frac{FV_{1_{33_t}}}{(1+d)^t} = 1200634$$

$$FV^D_{2_3} = FV_{2_{31_0}} + \sum_{t=0}^{N-1} \frac{FV_{2_{32_t}}}{(1+d)^t} + \sum_{t=0}^{N-1} \frac{FV_{i_{33_t}}}{(1+d)^t} = 1367791$$

## B.4 SME Scenario

### Determining the Qualitative Measure - Weighting Subjective Factors

| | mod. | pri. | reuse | int. | eff. | man. | $v_i$ |
|---|---|---|---|---|---|---|---|
| modifiability | 1 | 4 | $\frac{1}{2}$ | 3 | $\frac{1}{2}$ | $\frac{1}{2}$ | 0.16 |
| privacy | $\frac{1}{4}$ | 1 | $\frac{1}{5}$ | $\frac{1}{2}$ | $\frac{1}{5}$ | $\frac{1}{5}$ | 0.04 |
| reuse | 2 | 5 | 1 | 4 | 1 | 1 | 0.25 |
| interoperability | $\frac{1}{3}$ | 2 | $\frac{1}{4}$ | 1 | $\frac{1}{4}$ | $\frac{1}{4}$ | 0.06 |
| efficiency | 2 | 5 | 1 | 4 | 1 | 1 | 0.25 |
| managability | 2 | 5 | 1 | 4 | 1 | 1 | 0.25 |
| | | | | | | | $\lambda_{max} = 6.056, CR = 0.009$ |

Table B.41: Priority comparison matrix for the first level factors

| | sc. 1 | sc. 2 | sc. 3 | sc. 6 | sc. 11 | $v_i$ |
|---|---|---|---|---|---|---|
| scenario 1 | 1 | $\frac{1}{9}$ | 1 | $\frac{1}{5}$ | $\frac{1}{3}$ | 0.05 |
| scenario 2 | 9 | 1 | 9 | 5 | 7 | 0.59 |
| scenario 3 | 1 | $\frac{1}{9}$ | 1 | $\frac{1}{5}$ | $\frac{1}{3}$ | 0.05 |
| scenario 6 | 5 | $\frac{1}{5}$ | 5 | 1 | 3 | 0.21 |
| scenario 11 | 3 | $\frac{1}{7}$ | 3 | $\frac{1}{3}$ | 1 | 0.11 |
| | | | | | | $\lambda_{max} = 5.199, CR = 0.045$ |

Table B.42: Priority comparison matrix for second level factor modifiability

| | sc. 7 | sc. 8 | $v_i$ |
|---|---|---|---|
| scenario 7 | 1 | 1 | 0.50 |
| scenatio 8 | 1 | 1 | 0.50 |
| | | | $\lambda_{max} = 2.000, CR = 0.000$ |

Table B.43: Priority comparison matrix for second level factor privacy

| | sc. 4 | sc. 9 | sc. 11 | $v_i$ |
|---|---|---|---|---|
| scenario 4 | 1 | 1 | 5 | 0.45 |
| scenario 9 | 1 | 1 | 5 | 0.45 |
| scenario 11 | $\frac{1}{5}$ | $\frac{1}{5}$ | 1 | 0.09 |
| $\lambda_{max} = 3.000, CR = 0.000$ | | | | |

Table B.44: Priority comparison matrix for second level factor reuse

| | sc. 5 | sc. 10 | $v_i$ |
|---|---|---|---|
| scenario 5 | 1 | $\frac{1}{4}$ | 0.20 |
| scenatio 10 | 4 | 1 | 0.80 |
| $\lambda_{max} = 2.000, CR = 0.000$ | | | |

Table B.45: Priority comparison matrix for second level factor interoperability

| | sc. 12 | sc. 13 | $v_i$ |
|---|---|---|---|
| scenario 12 | 1 | 5 | 0.83 |
| scenatio 13 | $\frac{1}{5}$ | 1 | 0.17 |
| $\lambda_{max} = 2.000, CR = 0.000$ | | | |

Table B.46: Priority comparison matrix for second level factor efficiency

| | sc. 14 | sc. 15 | $v_i$ |
|---|---|---|---|
| scenario 14 | 1 | $\frac{1}{5}$ | 0.17 |
| scenatio 15 | 5 | 1 | 0.83 |
| $\lambda_{max} = 2.000, CR = 0.000$ | | | |

Table B.47: Priority comparison matrix for second level factor manageability

### Determining the Qualitative Measure - Rating the Scenarios

| scenario 1 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | $\frac{1}{2}$ | $\frac{1}{2}$ | 0.50 |
| Cen-Br. | 2 | 1 | 1 | 1.00 |
| Dec-Br. | 2 | 1 | 1 | 1.00 |
| | | | $\lambda_{max} = 3.000, CR = 0.000$ | |

Table B.48: Rating scenario 1

| scenario 2 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | $\frac{1}{2}$ | $\frac{1}{2}$ | 0.36 |
| Cen-Br. | 2 | 1 | $\frac{1}{3}$ | 0.50 |
| Dec-Br. | 2 | 3 | 1 | 1.00 |
| | | | $\lambda_{max} = 3.136, CR = 0.131$ | |

Table B.49: Rating scenario 2

| scenario 3 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | $\frac{1}{2}$ | $\frac{1}{2}$ | 0.50 |
| Cen-Br. | 2 | 1 | 1 | 1.00 |
| Dec-Br. | 2 | 1 | 1 | 1.00 |
| | | | $\lambda_{max} = 3.000, CR = 0.000$ | |

Table B.50: Rating scenario 3

| scenario 4 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | 3 | 2 | 1.00 |
| Cen-Br. | $\frac{1}{3}$ | 1 | $\frac{1}{2}$ | 0.30 |
| Dec-Br. | $\frac{1}{2}$ | 2 | 1 | 0.55 |
| | | | $\lambda_{max} = 3.009, CR = 0.009$ | |

Table B.51: Rating scenario 4

| scenario 5 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | $\frac{1}{3}$ | $\frac{1}{5}$ | 0.17 |
| Cen-Br. | 3 | 1 | $\frac{1}{3}$ | 0.41 |
| Dec-Br. | 5 | 3 | 1 | 1.00 |
| | | | $\lambda_{max} = 3.039, CR = 0.038$ | |

Table B.52: Rating scenario 5

| scenario 6 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | $\frac{1}{5}$ | $\frac{1}{5}$ | 0.20 |
| Cen-Br. | 5 | 1 | 1 | 1.00 |
| Dec-Br. | 5 | 1 | 1 | 1.00 |
| | | | $\lambda_{max} = 3.000, CR = 0.000$ | |

Table B.53: Rating scenario 6

| scenario 7 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | 1 | 1 | 1.00 |
| Cen-Br. | 1 | 1 | 1 | 1.00 |
| Dec-Br. | 1 | 1 | 1 | 1.00 |
| | | | $\lambda_{max} = 3.000, CR = 0.000$ | |

Table B.54: Rating scenario 7

| scenario 8 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | 1 | 1 | 1.00 |
| Cen-Br. | 1 | 1 | 1 | 1.00 |
| Dec-Br. | 1 | 1 | 1 | 1.00 |
| | | | $\lambda_{max} = 3.000, CR = 0.000$ | |

Table B.55: Rating scenario 8

| scenario 9 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | $\frac{1}{3}$ | $\frac{1}{5}$ | 0.17 |
| Cen-Br. | 3 | 1 | $\frac{1}{3}$ | 0.41 |
| Dec-Br. | 5 | 3 | 1 | 1.00 |
| | | | $\lambda_{max} = 3.039, CR = 0.038$ | |

Table B.56: Rating scenario 9

| scenario 10 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | $\frac{1}{2}$ | $\frac{1}{2}$ | 0.40 |
| Cen-Br. | 5 | 1 | $\frac{1}{2}$ | 0.64 |
| Dec-Br. | 5 | 2 | 1 | 1.00 |
| | | | $\lambda_{max} = 3.054, CR = 0.052$ | |

Table B.57: Rating scenario 10

| scenario 11 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | $\frac{1}{2}$ | $\frac{1}{2}$ | 0.40 |
| Cen-Br. | 2 | 1 | $\frac{1}{2}$ | 0.64 |
| Dec-Br. | 2 | 2 | 1 | 1.00 |
| | | | $\lambda_{max} = 3.054, CR = 0.052$ | |

Table B.58: Rating scenario 11

| scenario 12 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | 7 | 3 | 1.00 |
| Cen-Br. | $\frac{1}{7}$ | 1 | $\frac{1}{4}$ | 0.12 |
| Dec-Br. | $\frac{1}{3}$ | 4 | 1 | 0.40 |
| | | | $\lambda_{max} = 3.032, CR = 0.031$ | |

Table B.59: Rating scenario 12

| scenario 13 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | $\frac{1}{2}$ | $\frac{1}{2}$ | 0.50 |
| Cen-Br. | 2 | 1 | 1 | 1.00 |
| Dec-Br. | 2 | 1 | 1 | 1.00 |
| | | | $\lambda_{max} = 3.000, CR = 0.000$ | |

Table B.60: Rating scenario 13

| scenario 14 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | $\frac{1}{3}$ | $\frac{1}{2}$ | 0.30 |
| Cen-Br. | 3 | 1 | 2 | 1.00 |
| Dec-Br. | 2 | $\frac{1}{2}$ | 1 | 0.55 |
| | | | $\lambda_{max} = 3.009, CR = 0.009$ | |

Table B.61: Rating scenario 14

| scenario 15 | Wo-Br. | Cen-Br. | Dec-Br. | $v_i^{id}$ |
|---|---|---|---|---|
| Wo-Br. | 1 | $\frac{1}{5}$ | $\frac{1}{3}$ | 0.17 |
| Cen-Br. | 5 | 1 | 3 | 1.00 |
| Dec-Br. | 3 | $\frac{1}{3}$ | 1 | 0.41 |
| | | | $\lambda_{max} = 3.039, CR = 0.038$ | |

Table B.62: Rating scenario 15

### Determining the Quantitative Measure

$i = 0$ – brokerless architecture
$i = 1$ – central broker architecture
$i = 2$ – decentral broker architecture
discount rate $d = 5.0\%$
number of discounting periods $N = 10$

| **Software Licence** | | | | | | |
|---|---|---|---|---|---|---|
| *Purchase Software* | $C_{purchaseSWBr}$ | = | 0 | $n_{purchaseSWBr_0}$ | = | 0 |
| | | | | $n_{purchaseSWBr_1}$ | = | 1 |
| | | | | $n_{purchaseSWBr_2}$ | = | 4 |
| | $C_{purchaseSWES}$ | = | 5000 | $n_{purchaseSWES_0}$ | = | 20 |
| | | | | $n_{purchaseSWES_1}$ | = | 20 |
| | | | | $n_{purchaseSWES_2}$ | = | 20 |
| *Annual Licences* | $C_{annualSWBr}$ | = | 3000 | $n_{annualSWBr_0}$ | = | 0 |
| | | | | $n_{annualSWBr_1}$ | = | 1 |
| | | | | $n_{annualSWBr_2}$ | = | 4 |
| | $C_{annualSWES}$ | = | 0 | $n_{annualSWES_0}$ | = | 20 |
| | | | | $n_{annualSWES_1}$ | = | 20 |
| | | | | $n_{annualSWES_2}$ | = | 20 |

$FV^D_{0_1} = FV_{0_{11_0}} + \sum_{t=0}^{N-1} \frac{FV_{0_{12_t}}}{(1+d)^t} = 100000$

$FV^D_{1_1} = FV_{1_{11_0}} + \sum_{t=0}^{N-1} \frac{FV_{1_{12_t}}}{(1+d)^t} = 124323$

$FV^D_{2_1} = FV_{2_{11_0}} + \sum_{t=0}^{N-1} \frac{FV_{2_{12_t}}}{(1+d)^t} = 197294$

| **Hardware** | | | | | | |
|---|---|---|---|---|---|---|
| *Purchase Hardware* | $C_{purchaseHWBr}$ | = | 20000 | $n_{purchaseHWBr_0}$ | = | 0 |
| | | | | $n_{purchaseHWBr_1}$ | = | 1 |
| | | | | $n_{purchaseHWBr_2}$ | = | 4 |
| | $C_{purchaseHWES}$ | = | 5000 | $n_{purchaseHWES_0}$ | = | 20 |
| | | | | $n_{purchaseHWES_1}$ | = | 20 |
| | | | | $n_{purchaseHWES_2}$ | = | 20 |
| *Lease Hardware* | $C_{leasingHWBr}$ | = | 0 | $n_{leasingHWBr_0}$ | = | 0 |
| | | | | $n_{leasingHWBr_1}$ | = | 1 |
| | | | | $n_{leasingHWBr_2}$ | = | 4 |
| | $C_{leasingHWES}$ | = | 0 | $n_{leasingHWES_0}$ | = | 20 |
| | | | | $n_{leasingHWES_1}$ | = | 20 |
| | | | | $n_{leasingHWES_2}$ | = | 20 |

$FV^D_{0_2} = FV_{0_{21_0}} + \sum_{t=0}^{N-1} \frac{FV_{0_{22_t}}}{(1+d)^t} = 100000$

$FV^D_{1_2} = FV_{1_{21_0}} + \sum_{t=0}^{N-1} \frac{FV_{1_{22_t}}}{(1+d)^t} = 120000$

$FV^D_{2_2} = FV_{2_{21_0}} + \sum_{t=0}^{N-1} \frac{FV_{2_{22_t}}}{(1+d)^t} = 180000$

| **Labour** | | | | | | |
|---|---|---|---|---|---|---|
| *Setup & maintenance* | $C_{setup_0}$ | = | 10000 | $C_{maintenance_0}$ | = | 10000 |
| | $C_{setup_1}$ | = | 25000 | $C_{maintenance_1}$ | = | 15000 |
| | $C_{setup_2}$ | = | 20000 | $C_{maintenance_2}$ | = | 10000 |
| *Change Broker* | $C_{changeBr_0}$ | = | 0 | $n_{changeBr_0}$ | = | 0 |
| | $C_{changeBr_1}$ | = | 2000 | $n_{changeBr_1}$ | = | 1 |
| | $C_{changeBr_2}$ | = | 1000 | $n_{changeBr_2}$ | = | 4 |
| *Change ES* | $C_{changeES_0}$ | = | 2 | $n_{changeES_0}$ | = | 200 |
| | $C_{changeES_1}$ | = | 0 | $n_{changeES_1}$ | = | 0 |
| | $C_{changeES_2}$ | = | 0 | $n_{changeES_2}$ | = | 0 |
| *Frequency of Change* | $n_{change}$ | = | 3 | | | |

$$FV^D_{0_3} = FV_{0_{31_0}} + \sum_{t=0}^{N-1} \frac{FV_{0_{32_t}}}{(1+d)^t} + \sum_{t=0}^{N-1} \frac{FV_{0_{33_t}}}{(1+d)^t} = 100808$$

$$FV^D_{1_3} = FV_{1_{31_0}} + \sum_{t=0}^{N-1} \frac{FV_{1_{32_t}}}{(1+d)^t} + \sum_{t=0}^{N-1} \frac{FV_{1_{33_t}}}{(1+d)^t} = 195264$$

$$FV^D_{2_3} = FV_{2_{31_0}} + \sum_{t=0}^{N-1} \frac{FV_{2_{32_t}}}{(1+d)^t} + \sum_{t=0}^{N-1} \frac{FV_{i_{33_t}}}{(1+d)^t} = 198372$$

# Appendix C

# Ontology-based Model Transformation

## C.1 Library Example

Listing C.1: QVT model transformation *Mt:Ma→Mc* that is generated for the library example

```
transformation MMa2MMb(ma:MMa; mc:MMc) {

  key MMc::Library{name};
  key MMc::Book{name};
  key MMc::Novel{name};
  key MMc::Journal{name};
  key MMc::Scientist{name};
  key MMc::StoryWriter{name};

  top relation LibraryToLibrary {
    n: String;
    checkonly domain ma c:Library {name=n};
    enforce domain mc c_:Library {name=n};
  }

  top relation NovelToNovel {
    i, n: String;
    checkonly domain ma c:Novel {ns=l:Library{},name=n,index=i} ;
    enforce domain mc c_:Novel {ns=l_:Library{},name=n,NULL} ;
    when {
      LibraryToLibrary(l,l_);
    }
    where {
      AuthorsToAuthors(c,c_);
    }
  }

  top relation JournalToJournal {
    i, n, t: String;
    checkonly domain ma c:Journal {ns=l:Library{},name=n,
      topic=t,index=i} ;
    enforce domain mc c_:Journal {ns=l_:Library{},name=n,
      topic=t,NULL} ;
    when {
```

```
      LibraryToLibrary(l,l_);
    }
    where {
      AuthorsToAuthors_1(c,c_);
      AuthorsToAuthors_2(c,c_);
    }
  }

  top relation AuthorToAuthor_StoryWriter {
    n: String;
    checkonly domain ma c:Author {ns=l:Library{}, name=n,
      profession='StoryWriter'} ;
    enforce domain mc c_:StoryWriter {ns=l_:Library{}, name=n} ;
    when {
      LibraryToLibrary(l,l_);
    }
  }

  top relation AuthorToAuthor_Scientist {
    n: String;
    checkonly domain ma c:Author {ns=l:Library{}, name=n,
      profession='Scientist'} ;
    enforce domain mc c_:Scientist {ns=l_:Library{}, name=n} ;
    when {
      LibraryToLibrary(l,l_);
    }
  }

  relation AuthorsToAuthors_1 {
    n: String;
    checkonly domain ma c:Book {authors=a:Author{name=n}} ;
    enforce domain mc c_:Book {journalAuthors=a_:Scientist{name=n}};
  }

  relation AuthorsToAuthors_2 {
    n: String;
    checkonly domain ma c:Book {authors=a:Author{name=n}} ;
    enforce domain mc c_:Book {novelAuthors=a_:StoryWriter{name=n}};
  }
}
```

## C.2 Reasoning Rules

| | |
|---|---|
| *Equality symmetry* | $C \equiv D \rightarrow D \equiv C$ |
| | If C equals D, then D equals C |
| | $C \equiv D \Leftrightarrow C^{\mathcal{I}} = D^{\mathcal{I}} \Leftrightarrow D^{\mathcal{I}} = C^{\mathcal{I}} \Leftrightarrow D \equiv C$ |
| *Overlap symmetry* | $C \; o \; D \rightarrow D \; o \; C$ |
| | If C overlaps with D, then D overlaps with C |
| | $C \; o \; D \Leftrightarrow C^{\mathcal{I}} \; O \; D^{\mathcal{I}} \Leftrightarrow D^{\mathcal{I}} \; O \; C^{\mathcal{I}} \Leftrightarrow D \; o \; C$ |
| *Possible overlap symmetry* | $C \; \Theta \; D \rightarrow D \; \Theta \; C$ |
| | If C has a possible overlap with D, then D has a possible overlap with C |
| | $C \; \Theta \; D \Leftrightarrow C^{\mathcal{I}} \; \theta \; D^{\mathcal{I}} \Leftrightarrow D^{\mathcal{I}} \; \theta \; C^{\mathcal{I}} \Leftrightarrow D \; \Theta \; C$ |

Table C.1: Rules for symmetry

| | |
|---|---|
| *Containment right inverse* | $C \sqsubseteq D \rightarrow D \sqsupseteq C$ |
| | If C is more specific than D, then D is less specific than C |
| | $C \sqsubseteq D \Leftrightarrow C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \Leftrightarrow D^{\mathcal{I}} \supseteq C^{\mathcal{I}} \Leftrightarrow D \sqsupseteq C$ |
| *Containment left inverse* | $C \sqsupseteq D \rightarrow D \sqsubseteq C$ |
| | If C is less specific than D, then D is more specific than C |
| | $C \sqsupseteq D \Leftrightarrow C^{\mathcal{I}} \supseteq D^{\mathcal{I}} \Leftrightarrow D^{\mathcal{I}} \subseteq C^{\mathcal{I}} \Leftrightarrow D \sqsubseteq C$ |

Table C.2: Rules for inverse

| | |
|---|---|
| *Equality transitivity* | $C \equiv D \sqcap D \equiv E \rightarrow C \equiv E$ |
| | If C equals D and D equals E, then C equals E |
| | $C \equiv D \sqcap D \equiv E \Leftrightarrow$<br>$C^{\mathcal{I}} = D^{\mathcal{I}} \wedge D^{\mathcal{I}} = E^{\mathcal{I}} \Rightarrow$<br>$C^{\mathcal{I}} = E^{\mathcal{I}} \Leftrightarrow$<br>$C \equiv E$ |
| *Containment right transitivity* | $C \sqsubseteq D \sqcap D \sqsubseteq E \rightarrow C \sqsubseteq E$ |
| | If C is more specific than D and D is more specific than E, then C is also more specific than E |
| | $C \sqsubseteq D \sqcap D \sqsubseteq E \Leftrightarrow$<br>$C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \wedge D^{\mathcal{I}} \subseteq E^{\mathcal{I}} \Rightarrow$<br>$C^{\mathcal{I}} \subseteq E^{\mathcal{I}} \Leftrightarrow$<br>$C \sqsubseteq E$ |
| *Containment left transitivity* | $C \sqsupseteq D \sqcap D \sqsupseteq E \rightarrow C \sqsupseteq E$ |
| | If C is less specific than D and D is less specific than E, then C is also less specific than E |
| | $C \sqsupseteq D \sqcap D \sqsupseteq E \Leftrightarrow$<br>$C^{\mathcal{I}} \supseteq D^{\mathcal{I}} \wedge D^{\mathcal{I}} \supseteq E^{\mathcal{I}} \Rightarrow$<br>$C^{\mathcal{I}} \supseteq E^{\mathcal{I}} \Leftrightarrow$<br>$C \sqsupseteq E$ |
| *Overlap (Pseudo-) transitivity* | $C \; o \; D \sqcap D \; o \; E \rightarrow C \; \Theta \; E$ |
| | If C overlaps with D and D overlaps with E, then there is a possibility that C also overlaps with E |
| | $C \; o \; D \sqcap D \; o \; E \Leftrightarrow$<br>$C^{\mathcal{I}} \; O \; D^{\mathcal{I}} \wedge D^{\mathcal{I}} \; O \; E^{\mathcal{I}} \Rightarrow$<br>$C^{\mathcal{I}} \; \theta \; E^{\mathcal{I}} \Leftrightarrow$<br>$C \; \Theta \; E$ |
| *Possible overlap transitivity* | $C \; \Theta^{P_1} \; D \sqcap D \; \Theta^{P_2} \; E \rightarrow C \; \Theta^{P_r} \; E$ |
| | If C possibly overlaps with D (probability $P_1$) and D possibly overlaps with E (probability $P_2$), then C might also overlap with E. The resulting probability $P_r$ is $P_1 * P_2$. |
| | $C \; \Theta^{P_1} \; D \sqcap D \; \Theta^{P_2} \; E \Leftrightarrow$<br>$C^{\mathcal{I}} \; \theta^{P_1} \; D^{\mathcal{I}} \wedge D^{\mathcal{I}} \; \theta^{P_2} \; E^{\mathcal{I}} \Rightarrow$<br>$C^{\mathcal{I}} \; \theta^{P_r} \; E^{\mathcal{I}} \Leftrightarrow$<br>$C \; \Theta^{P_r} \; E$ |

Table C.3: Rules for transitivity

| | |
|---|---|
| *Containment right 1* | $C \sqsubseteq D \sqcap D \equiv E \rightarrow C \sqsubseteq E$ |
| | If C is more specific than D and D equals E, then C is also more specific than E. |
| | $C \sqsubseteq D \sqcap D \equiv E \Leftrightarrow$<br>$C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \wedge D^{\mathcal{I}} = E^{\mathcal{I}} \Rightarrow$<br>$C^{\mathcal{I}} \subseteq E^{\mathcal{I}} \Leftrightarrow$<br>$C \sqsubseteq E$ |
| *Containment right 2* | $C \equiv D \sqcap D \sqsubseteq E \rightarrow C \sqsubseteq E$ |
| | If C equals D and D is more specific than E, then C is also more specific than E. |
| | $C \equiv D \sqcap D \sqsubseteq E \Leftrightarrow$<br>$C^{\mathcal{I}} = D^{\mathcal{I}} \wedge D^{\mathcal{I}} \subseteq E^{\mathcal{I}} \Rightarrow$<br>$C^{\mathcal{I}} \subseteq E^{\mathcal{I}} \Leftrightarrow$<br>$C \sqsubseteq E$ |
| *Containment left 1* | $C \sqsupseteq D \sqcap D \equiv E \rightarrow C \sqsupseteq E$ |
| | If C is less specific than D and D equals E, then C is also less specific than E. |
| | $C \sqsupseteq D \sqcap D \equiv E \Leftrightarrow$<br>$C^{\mathcal{I}} \supseteq D^{\mathcal{I}} \wedge D^{\mathcal{I}} = E^{\mathcal{I}} \Rightarrow$<br>$C^{\mathcal{I}} \supseteq E^{\mathcal{I}} \Leftrightarrow$<br>$C \sqsupseteq E$ |
| *Containment left 2* | $C \equiv D \sqcap D \sqsupseteq E \rightarrow C \sqsupseteq E$ |
| | If C equals D and D is less specific than E, then C is also less specific than E. |
| | $C \equiv D \sqcap D \sqsupseteq E \Leftrightarrow$<br>$C^{\mathcal{I}} = D^{\mathcal{I}} \wedge D^{\mathcal{I}} \supseteq E^{\mathcal{I}} \Rightarrow$<br>$C^{\mathcal{I}} \supseteq E^{\mathcal{I}} \Leftrightarrow$<br>$C \sqsupseteq E$ |

Table C.4: Rules for equality reduction, part 1

| | |
|---|---|
| *Overlap 1* | $C\ o\ D \sqcap D \equiv E \rightarrow C\ o\ E$ |
| | If C overlaps with D and D equals E, then C also overlaps with E. |
| | $C\ o\ D \sqcap D \equiv E \Leftrightarrow$<br>$C^{\mathcal{I}}\ O\ D^{\mathcal{I}} \wedge D^{\mathcal{I}} = E^{\mathcal{I}} \Rightarrow$<br>$C^{\mathcal{I}}\ O\ E^{\mathcal{I}} \Leftrightarrow$<br>$C\ o\ E$ |
| *Overlap 2* | $C \equiv D \sqcap D\ o\ E \rightarrow C\ o\ E$ |
| | If C equals with D and D overlaps E, then C also overlaps with E. |
| | $C \equiv D \sqcap D\ o\ E \Leftrightarrow$<br>$C^{\mathcal{I}} = D^{\mathcal{I}} \wedge D^{\mathcal{I}}\ O\ E^{\mathcal{I}} \Rightarrow$<br>$C^{\mathcal{I}}\ O\ E^{\mathcal{I}} \Leftrightarrow$<br>$C\ o\ E$ |
| *Possible overlap 1* | $C\ \Theta\ D \sqcap D \equiv E \rightarrow C\ \Theta\ E$ |
| | If C has a possible overlaps with D and D equals E, then C also has a possible overlap with E. |
| | $C\ \Theta\ D \sqcap D \equiv E \Leftrightarrow$<br>$C^{\mathcal{I}}\ \theta\ D^{\mathcal{I}} \wedge D^{\mathcal{I}} = E^{\mathcal{I}} \Rightarrow$<br>$C^{\mathcal{I}}\ \theta\ E^{\mathcal{I}} \Leftrightarrow$<br>$C\ \Theta\ E$ |
| *Possible overlap 2* | $C \equiv D \sqcap D\ \Theta\ E \rightarrow C\ \Theta\ E$ |
| | If C equals D and D has a possible overlaps with E, then C also has a possible overlap with E. |
| | $C \equiv D \sqcap D\ \Theta\ E \Leftrightarrow$<br>$C^{\mathcal{I}} = D^{\mathcal{I}} \wedge D^{\mathcal{I}}\ \theta\ E^{\mathcal{I}} \Rightarrow$<br>$C^{\mathcal{I}}\ \theta\ E^{\mathcal{I}} \Leftrightarrow$<br>$C\ \Theta\ E$ |

Table C.5: Rules for equality reduction, part 2

| | |
|---|---|
| *Containment in general* | $C \sqsubseteq D \sqcap D \sqsupseteq E \rightarrow C\ \Theta\ E$ |
| | If C is more specific than D and D is less specific than E, C and E have a possible overlap. |
| | $C \sqsubseteq D \sqcap D \sqsupseteq E \Leftrightarrow$<br>$C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \wedge D^{\mathcal{I}} \supseteq E^{\mathcal{I}} \Rightarrow$<br>$C^{\mathcal{I}}\ \theta\ E^{\mathcal{I}} \Leftrightarrow$<br>$C\ \Theta\ E$ |
| *Containment of special* | $C \sqsupseteq D \sqcap D \sqsubseteq E \rightarrow C\ o\ E$ |
| | If C is less specific than D and D is more specific than E, then C and E have a overlap. |
| | $C \sqsupseteq D \sqcap D \sqsubseteq E \Leftrightarrow$<br>$C^{\mathcal{I}} \supseteq D^{\mathcal{I}} \wedge D^{\mathcal{I}} \subseteq E^{\mathcal{I}} \Rightarrow$<br>$C^{\mathcal{I}}\ O\ E^{\mathcal{I}} \Leftrightarrow$<br>$C\ o\ E$ |

Table C.6: Containments

| | |
|---|---|
| *Containment right and overlap* | $C \sqsubseteq D \sqcap D \; o \; E \rightarrow C \; \Theta \; E$ |
| | If C is more specific than D and D overlaps with E, C and E have a possible overlap. |
| | $C \sqsubseteq D \sqcap D \; o \; E \Leftrightarrow$<br>$C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \wedge D^{\mathcal{I}} \; O \; E^{\mathcal{I}} \Rightarrow$<br>$C^{\mathcal{I}} \; \theta \; E^{\mathcal{I}} \Leftrightarrow$<br>$C \; \Theta \; E$ |
| *Overlap and containment right* | $C \; o \; D \sqcap D \sqsubseteq E \rightarrow C \; o \; E$ |
| | If C overlaps with D and D is more specific than E, then C and E also overlap. |
| | $C \; o \; D \sqcap D \sqsubseteq E \Leftrightarrow$<br>$C^{\mathcal{I}} \; O \; D^{\mathcal{I}} \wedge D^{\mathcal{I}} \subseteq E^{\mathcal{I}} \Rightarrow$<br>$C^{\mathcal{I}} \; O \; E^{\mathcal{I}} \Leftrightarrow$<br>$C \; o \; E$ |
| *Containment left and overlap* | $C \sqsupseteq D \sqcap D \; o \; E \rightarrow C \; o \; E$ |
| | If C is less specific than D and D overlaps with E, C and E also overlap. |
| | $C \sqsupseteq D \sqcap D \; o \; E \Leftrightarrow$<br>$C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \wedge D^{\mathcal{I}} \; O \; E^{\mathcal{I}} \Rightarrow$<br>$C^{\mathcal{I}} \; O \; E^{\mathcal{I}} \Leftrightarrow$<br>$C \; o \; E$ |
| *Overlap and containment left* | $C \; o \; D \sqcap D \sqsupseteq E \rightarrow C \; \Theta \; E$ |
| | If C overlaps with D and D is less specific than E, then C and E have a possible overlap. |
| | $C \; o \; D \sqcap D \sqsupseteq E \Leftrightarrow$<br>$C^{\mathcal{I}} \; O \; D^{\mathcal{I}} \wedge D^{\mathcal{I}} \subseteq E^{\mathcal{I}} \Rightarrow$<br>$C^{\mathcal{I}} \; \theta \; E^{\mathcal{I}} \Leftrightarrow$<br>$C \; \Theta \; E$ |

Table C.7: Containment with overlap

| | |
|---|---|
| *X equals A* | $X \equiv A \sqcap A \equiv \bigcup_{i=1}^{n} A_i \rightarrow X \sqsupseteq A_1 \sqcap \ldots \sqcap X \sqsupseteq A_n$ |
| | If X equals A and A is an union of classes $A_1$ to $A_n$, then X also contains every class $A_1$ to $A_n$. |
| | $X \equiv A \sqcap A \equiv \bigcup_{i=1}^{n} A_i \Leftrightarrow$<br>$X^{\mathcal{I}} = A^{\mathcal{I}} \wedge A^{\mathcal{I}} = A_1^{\mathcal{I}} \cup \ldots \cup A_n^{\mathcal{I}} \Rightarrow$<br>$X^{\mathcal{I}} \supseteq A_1^{\mathcal{I}} \wedge \ldots \wedge X^{\mathcal{I}} \supseteq A_n^{\mathcal{I}} \Leftrightarrow$<br>$X \sqsupseteq A_1 \sqcap \ldots \sqcap X \sqsupseteq A_n$ |
| *X contained in A* | $X \sqsubseteq A \sqcap A \equiv \bigcup_{i=1}^{n} A_i \rightarrow X \,\Theta\, A_1 \sqcap \ldots \sqcap X \,\Theta\, A_n$ |
| | If X is more specific than A and A is an union of classes $A_1$ to $A_n$, X has a possible overlap with classes $A_1$ to $A_n$. |
| | $X \sqsubseteq A \sqcap A \equiv \bigcup_{i=1}^{n} A_i \Leftrightarrow$<br>$X^{\mathcal{I}} \subseteq A^{\mathcal{I}} \wedge A^{\mathcal{I}} = A_1^{\mathcal{I}} \cup \ldots \cup A_n^{\mathcal{I}} \Rightarrow$<br>$X^{\mathcal{I}} \,\theta\, A_1^{\mathcal{I}} \wedge \ldots \wedge X^{\mathcal{I}} \,\theta\, A_n^{\mathcal{I}} \Leftrightarrow$<br>$X \,\Theta\, A_1 \sqcap \ldots \sqcap X \,\Theta\, A_n$ |
| *X contains A* | $X \sqsupseteq A \sqcap A \equiv \bigcup_{i=1}^{n} A_i \rightarrow X \sqsupseteq A_1 \sqcap \ldots \sqcap X \sqsupseteq A_n$ |
| | If X is less specific than A and A is an union of classes $A_1$ to $A_n$, X contains classes $A_1$ to $A_n$. |
| | $X \sqsupseteq A \sqcap A \equiv \bigcup_{i=1}^{n} A_i \Leftrightarrow$<br>$X^{\mathcal{I}} \supseteq A^{\mathcal{I}} \wedge A^{\mathcal{I}} = A_1^{\mathcal{I}} \cup \ldots \cup A_n^{\mathcal{I}} \Rightarrow$<br>$X^{\mathcal{I}} \supseteq A_1^{\mathcal{I}} \wedge \ldots \wedge X^{\mathcal{I}} \supseteq A_n^{\mathcal{I}} \Leftrightarrow$<br>$X \sqsupseteq A_1 \sqcap \ldots \sqcap X \sqsupseteq A_n$ |
| *X overlaps A* | $X \; O \; A \sqcap A \equiv \bigcup_{i=1}^{n} A_i \rightarrow X \,\Theta\, A_1 \sqcap \ldots \sqcap X \,\Theta\, A_n$ |
| | If X overlaps with A and A is an union of classes $A_1$ to $A_n$, X has a possible overlap with classes $A_1$ to $A_n$. |
| | $X \; o \; A \sqcap A \equiv \bigcup_{i=1}^{n} A_i \Leftrightarrow$<br>$X^{\mathcal{I}} \; O \; A^{\mathcal{I}} \wedge A^{\mathcal{I}} = A_1^{\mathcal{I}} \cup \ldots \cup A_n^{\mathcal{I}} \Rightarrow$<br>$X^{\mathcal{I}} \,\theta\, A_1^{\mathcal{I}} \wedge \ldots \wedge X^{\mathcal{I}} \,\theta\, A_n^{\mathcal{I}} \Leftrightarrow$<br>$X \,\Theta\, A_1 \sqcap \ldots \sqcap X \,\Theta\, A_n$ |
| *X possible overlap A* | $X \;\; \Theta \;\; A \sqcap A \equiv \bigcup_{i=1}^{n} A_i \rightarrow X \,\Theta\, A_1 \sqcap \ldots \sqcap X \,\Theta\, A_n$ |
| | If X has a possible overlap with A and A is an union of classes $A_1$ to $A_n$, X has a possible overlap with classes $A_1$ to $A_n$. |
| | $X \,\Theta\, A \sqcap A \equiv \bigcup_{i=1}^{n} A_i \Leftrightarrow$<br>$X^{\mathcal{I}} \,\theta\, A^{\mathcal{I}} \wedge A^{\mathcal{I}} = A_1^{\mathcal{I}} \cup \ldots \cup A_n^{\mathcal{I}} \Rightarrow$<br>$X^{\mathcal{I}} \,\theta\, A_1^{\mathcal{I}} \wedge \ldots \wedge X^{\mathcal{I}} \,\theta\, A_n^{\mathcal{I}} \Leftrightarrow$<br>$X \,\Theta\, A_1 \sqcap \ldots \sqcap X \,\Theta\, A_n$ |

Table C.8: $A \equiv \bigcup_{i=1}^{n} A_i \wedge 1 \leqslant m \leqslant n$, part 1

| | |
|---|---|
| $A_m$ *equals* $X$ | $A_m \equiv X \sqcap A \equiv \bigcup_{i=1}^{n} A_i \rightarrow A \sqsupseteq X$ |
| | If $A_m$ equals X and A is an union of classes $A_1$ to $A_n$ ($A_m$ is one of these classes), A contains X |
| | $A_m \equiv X \sqcap A \equiv \bigcup_{i=1}^{n} A_i \Leftrightarrow$<br>$A_m^{\mathcal{I}} = X^{\mathcal{I}} \wedge A^{\mathcal{I}} = A_1^{\mathcal{I}} \cup \ldots \cup A_m^{\mathcal{I}} \cup \ldots \cup A_n^{\mathcal{I}} \Rightarrow$<br>$A^{\mathcal{I}} \supseteq X^{\mathcal{I}} \Leftrightarrow$<br>$A \sqsupseteq X$ |
| $A_m$ *is contained in* $X$ | $A_m \sqsubseteq X \sqcap A \equiv \bigcup_{i=1}^{n} A_i \rightarrow A \; o \; X$ |
| | If $A_m$ is more specific than X and A is an union of classes $A_1$ to $A_n$ ($A_m$ is one of these classes), A overlaps with X |
| | $A_m \sqsubseteq X \sqcap A \equiv \bigcup_{i=1}^{n} A_i \Leftrightarrow$<br>$A_m^{\mathcal{I}} \subseteq X^{\mathcal{I}} \wedge A^{\mathcal{I}} = A_1^{\mathcal{I}} \cup \ldots \cup A_m^{\mathcal{I}} \cup \ldots \cup A_n^{\mathcal{I}} \Rightarrow$<br>$A^{\mathcal{I}} \; O \; X^{\mathcal{I}} \Leftrightarrow$<br>$A \; o \; X$ |
| $A_m$ *contains* $X$ | $A_m \sqsupseteq X \sqcap A \equiv \bigcup_{i=1}^{n} A_i \rightarrow A \sqsupseteq X$ |
| | If $A_m$ is less specific than X and A is an union of classes $A_1$ to $A_n$ ($A_m$ is one of these classes), A is also less specific than X |
| | $A_m \sqsupseteq X \sqcap A \equiv \bigcup_{i=1}^{n} A_i \Leftrightarrow$<br>$A_m^{\mathcal{I}} \supseteq X^{\mathcal{I}} \wedge A^{\mathcal{I}} = A_1^{\mathcal{I}} \cup \ldots \cup A_m^{\mathcal{I}} \cup \ldots \cup A_n^{\mathcal{I}} \Rightarrow$<br>$A^{\mathcal{I}} \supseteq X^{\mathcal{I}} \Leftrightarrow$<br>$A \sqsupseteq X$ |
| $A_m$ *overlaps* $X$ | $A_m \; o \; X \sqcap A \equiv \bigcup_{i=1}^{n} A_i \rightarrow A \; o \; X$ |
| | If $A_m$ overlaps with X and A is an union of classes $A_1$ to $A_n$ ($A_m$ is one of these classes), A also overlaps with X |
| | $A_m \; o \; X \sqcap A \equiv \bigcup_{i=1}^{n} A_i \Leftrightarrow$<br>$A_m^{\mathcal{I}} \; O \; X^{\mathcal{I}} \wedge A^{\mathcal{I}} = A_1^{\mathcal{I}} \cup \ldots \cup A_m^{\mathcal{I}} \cup \ldots \cup A_n^{\mathcal{I}} \Rightarrow$<br>$A^{\mathcal{I}} \; O \; X^{\mathcal{I}} \Leftrightarrow$<br>$A \; o \; X$ |
| $A_m$ *possible overlap* $X$ | $A_m \; \Theta \; X \sqcap A \equiv \bigcup_{i=1}^{n} A_i \rightarrow A \; \Theta \; X$ |
| | If $A_m$ has a possible overlap with X and A is an union of classes $A_1$ to $A_n$ ($A_m$ is one of these classes), A also has a possible overlap with X |
| | $A_m \; \Theta \; X \sqcap A \equiv \bigcup_{i=1}^{n} A_i \Leftrightarrow$<br>$A_m^{\mathcal{I}} \; \theta \; X^{\mathcal{I}} \wedge A^{\mathcal{I}} = A_1^{\mathcal{I}} \cup \ldots \cup A_m^{\mathcal{I}} \cup \ldots \cup A_n^{\mathcal{I}} \Rightarrow$<br>$A^{\mathcal{I}} \; \theta \; X^{\mathcal{I}} \Leftrightarrow$<br>$A \; \Theta \; X$ |

Table C.9: $A \equiv \bigcup_{i=1}^{n} A_i \wedge 1 \leqslant m \leqslant n$, part 2

| | |
|---|---|
| *X equals A* | $X \equiv A \sqcap A \equiv \bigcap_{i=1}^{n} A_i \rightarrow X \sqsubseteq A_1 \sqcap \ldots \sqcap X \sqsubseteq A_n$ |
| | If X equals A and A is an intersection of classes $A_1$ to $A_n$, X is contained in the classes $A_1$ to $A_n$. |
| | $X \equiv A \sqcap A \equiv \bigcap_{i=1}^{n} A_i \Leftrightarrow$<br>$X^{\mathcal{I}} = A^{\mathcal{I}} \wedge A^{\mathcal{I}} = A_1^{\mathcal{I}} \cap \ldots \cap A_n^{\mathcal{I}} \Rightarrow$<br>$X^{\mathcal{I}} \sqsubseteq A_1^{\mathcal{I}} \wedge \ldots \wedge X^{\mathcal{I}} \sqsubseteq A_n^{\mathcal{I}} \Leftrightarrow$<br>$X \sqsubseteq A_1 \sqcap \ldots \sqcap X \sqsubseteq A_n$ |
| *X contained in A* | $X \sqsubseteq A \sqcap A \equiv \bigcap_{i=1}^{n} A_i \rightarrow X \sqsubseteq A_1 \sqcap \ldots \sqcap X \sqsubseteq A_n$ |
| | If X is more specific than A and A is an intersection of classes $A_1$ to $A_n$, X is containd in the classes $A_1$ to $A_n$. |
| | $X \sqsubseteq A \sqcap A \equiv \bigcap_{i=1}^{n} A_i \Leftrightarrow$<br>$X^{\mathcal{I}} \subseteq A^{\mathcal{I}} \wedge A^{\mathcal{I}} = A_1^{\mathcal{I}} \cap \ldots \cap A_n^{\mathcal{I}} \Rightarrow$<br>$X^{\mathcal{I}} \subseteq A_1^{\mathcal{I}} \wedge \ldots \wedge X^{\mathcal{I}} \subseteq A_n^{\mathcal{I}} \Leftrightarrow$<br>$X \sqsubseteq A_1 \sqcap \ldots \sqcap X \sqsubseteq A_n$ |
| *X contains A* | $X \sqsupseteq A \sqcap A \equiv \bigcap_{i=1}^{n} A_i \rightarrow X\ o\ A_1 \sqcap \ldots \sqcap X\ o\ A_n$ |
| | If X is less specific than A and A is an intersection of classes $A_1$ to $A_n$, X overlaps with classes $A_1$ to $A_n$. |
| | $X \sqsupseteq A \sqcap A \equiv \bigcap_{i=1}^{n} A_i \Leftrightarrow$<br>$X^{\mathcal{I}} \supseteq A^{\mathcal{I}} \wedge A^{\mathcal{I}} = A_1^{\mathcal{I}} \cap \ldots \cap A_n^{\mathcal{I}} \Rightarrow$<br>$X^{\mathcal{I}}\ O\ A_1^{\mathcal{I}} \wedge \ldots \wedge X^{\mathcal{I}}\ O\ A_n^{\mathcal{I}} \Leftrightarrow$<br>$X\ o\ A_1 \sqcap \ldots \sqcap X\ o\ A_n$ |
| *X overlaps A* | $X\ o\ A \sqcap A \equiv \bigcap_{i=1}^{n} A_i \rightarrow X\ o\ A_1 \sqcap \ldots \sqcap X\ o\ A_n$ |
| | If X overlaps with A and A is an intersection of classes $A_1$ to $A_n$, X overlaps with classes $A_1$ to $A_n$. |
| | $X\ o\ A \sqcap A \equiv \bigcap_{i=1}^{n} A_i \Leftrightarrow$<br>$X^{\mathcal{I}}\ O\ A^{\mathcal{I}} \wedge A^{\mathcal{I}} = A_1^{\mathcal{I}} \cap \ldots \cap A_n^{\mathcal{I}} \Rightarrow$<br>$X^{\mathcal{I}}\ O\ A_1^{\mathcal{I}} \wedge \ldots \wedge X^{\mathcal{I}}\ O\ A_n^{\mathcal{I}} \Leftrightarrow$<br>$X\ o\ A_1 \sqcap \ldots \sqcap X\ o\ A_n$ |
| *X possible overlap A* | $X\ \Theta\ A \sqcap A \equiv \bigcap_{i=1}^{n} A_i \rightarrow X\ \Theta\ A_1 \sqcap \ldots \sqcap X\ \Theta\ A_n$ |
| | If X has a possible overlap with A and A is an intersection of classes $A_1$ to $A_n$, X has a possible overlap with classes $A_1$ to $A_n$. |
| | $X\ \Theta\ A \sqcap A \equiv \bigcap_{i=1}^{n} A_i \Leftrightarrow$<br>$X^{\mathcal{I}}\ \theta\ A^{\mathcal{I}} \wedge A^{\mathcal{I}} = A_1^{\mathcal{I}} \cap \ldots \cap A_n^{\mathcal{I}} \Leftrightarrow$<br>$X^{\mathcal{I}}\ \theta\ A_1^{\mathcal{I}} \wedge \ldots \wedge X^{\mathcal{I}}\ \theta\ A_n^{\mathcal{I}} \Leftrightarrow$<br>$X\ \Theta\ A_1 \sqcap \ldots \sqcap X\ \Theta\ A_n$ |

Table C.10: $A \equiv \bigcap_{i=1}^{n} A_i \wedge 1 \leqslant m \leqslant n$, part 1

| | |
|---|---|
| $A_m$ *equals* $X$ | $A_m \equiv X \sqcap A \equiv \bigcap_{i=1}^{n} A_i \rightarrow A \sqsubseteq X$ |
| | If $A_m$ equals X and A is an intersection of classes $A_1$ to $A_n$ ($A_m$ is one of these), A is contained in X |
| | $A_m \equiv X \sqcap A \equiv \bigcap_{i=1}^{n} A_i \Leftrightarrow$<br>$A_m^{\mathcal{I}} = X^{\mathcal{I}} \wedge A^{\mathcal{I}} = A_1^{\mathcal{I}} \cap \ldots \cap A_m^{\mathcal{I}} \cap \ldots \cap A_n^{\mathcal{I}} \Rightarrow$<br>$A^{\mathcal{I}} \subseteq X^{\mathcal{I}} \Leftrightarrow$<br>$A \sqsubseteq X$ |
| $A_m$ *is contained in* $X$ | $A_m \sqsubseteq X \sqcap A \equiv \bigcap_{i=1}^{n} A_i \rightarrow A \sqsubseteq X$ |
| | If $A_m$ is more specific than X and A is an intersection of classes $A_1$ to $A_n$ ($A_m$ is one of these), A is contained in X |
| | $A_m \sqsubseteq X \sqcap A \equiv \bigcap_{i=1}^{n} A_i \Leftrightarrow$<br>$A_m^{\mathcal{I}} \subseteq X^{\mathcal{I}} \wedge A^{\mathcal{I}} = A_1^{\mathcal{I}} \cap \ldots \cap A_m^{\mathcal{I}} \cap \ldots \cap A_n^{\mathcal{I}} \Rightarrow$<br>$A^{\mathcal{I}} \subseteq X^{\mathcal{I}} \Leftrightarrow$<br>$A \sqsubseteq X$ |
| $A_m$ *contains* $X$ | $A_m \sqsupseteq X \sqcap A \equiv \bigcap_{i=1}^{n} A_i \rightarrow A \,\Theta\, X$ |
| | If $A_m$ is less specific than X and A is an intersection of classes $A_1$ to $A_n$ ($A_m$ is one of these), A possibly overlaps with X |
| | $A_m \sqsupseteq X \sqcap A \equiv \bigcap_{i=1}^{n} A_i \Leftrightarrow$<br>$A_m^{\mathcal{I}} \supseteq X^{\mathcal{I}} \wedge A^{\mathcal{I}} = A_1^{\mathcal{I}} \cap A_m^{\mathcal{I}} \cap \ldots \cap A_n^{\mathcal{I}} \Rightarrow$<br>$A^{\mathcal{I}}\ \theta\ X^{\mathcal{I}} \Leftrightarrow$<br>$A \,\Theta\, X$ |
| $A_m$ *overlaps* $X$ | $A_m\ o\ X \sqcap A \equiv \bigcap_{i=1}^{n} A_i \rightarrow A \,\Theta\, X$ |
| | If $A_m$ overlaps with X and A is an intersection of classes $A_1$ to $A_n$ ($A_m$ is one of these), A has a possible overlap with X |
| | $A_m\ o\ X \sqcap A \equiv \bigcap_{i=1}^{n} A_i \Leftrightarrow$<br>$A_m^{\mathcal{I}}\ O\ X^{\mathcal{I}} \wedge A^{\mathcal{I}} = A_1^{\mathcal{I}} \cap \ldots \cap A_m^{\mathcal{I}} \cap A_n^{\mathcal{I}} \Rightarrow$<br>$A^{\mathcal{I}}\ \theta\ X^{\mathcal{I}} \Leftrightarrow$<br>$A \,\Theta\, X$ |
| $A_m$ *possible overlap* $X$ | $A_m \,\Theta\, X \sqcap A \equiv \bigcap_{i=1}^{n} A_i \rightarrow A \,\Theta\, X$ |
| | If $A_m$ has a possible overlap with X and A is an intersection of classes $A_1$ to $A_n$ ($A_m$ is one of these), A also has a possible overlap with X |
| | $A_m \,\Theta\, X \sqcap A \equiv \bigcap_{i=1}^{n} A_i \Leftrightarrow$<br>$A_m^{\mathcal{I}}\ \theta\ X^{\mathcal{I}} \wedge A^{\mathcal{I}} = A_1^{\mathcal{I}} \cap \ldots \cap A_m^{\mathcal{I}} \cap A_n^{\mathcal{I}} \Rightarrow$<br>$A^{\mathcal{I}}\ \theta\ X^{\mathcal{I}} \Leftrightarrow$<br>$A \,\Theta\, X$ |

Table C.11: $A \equiv \bigcap_{i=1}^{n} A_i \wedge 1 \leqslant m \leqslant n$, part 2

| | |
|---|---|
| *X equals A* | $X \equiv A \sqcap A \equiv \neg B \rightarrow X \equiv \neg B$ |
| | If X equals A and A is the complement of B, X is the complement of B. |
| | $X \equiv A \sqcap A \equiv \neg B \Leftrightarrow$<br>$X^{\mathcal{I}} = A^{\mathcal{I}} \wedge A^{\mathcal{I}} = \neg B^{\mathcal{I}} \Rightarrow$<br>$X^{\mathcal{I}} = \neg B^{\mathcal{I}} \Leftrightarrow$<br>$X \equiv \neg B$ |
| *X contained in A* | $X \sqsubseteq A \sqcap A \equiv \neg B \rightarrow X \sqsubseteq \neg B$ |
| | If X is more specific than A and A is the complement of B, X contains the complement of B |
| | $X \sqsubseteq A \sqcap A \equiv \neg B \Leftrightarrow$<br>$X^{\mathcal{I}} \subseteq A^{\mathcal{I}} \wedge A^{\mathcal{I}} = \neg B^{\mathcal{I}} \Rightarrow$<br>$X^{\mathcal{I}} \subseteq \neg B^{\mathcal{I}} \Leftrightarrow$<br>$X \sqsubseteq \neg B$ |
| *X contains A* | $X \sqsupseteq A \sqcap A \equiv \neg B \rightarrow X \,\Theta\, B$ |
| | If X is less specific than A and A is the complement of B, X has a possible overlap with B. |
| | $X \sqsupseteq A \sqcap A \equiv \neg B \Leftrightarrow$<br>$X^{\mathcal{I}} \supseteq A^{\mathcal{I}} \wedge A^{\mathcal{I}} = \neg B^{\mathcal{I}} \Rightarrow$<br>$X^{\mathcal{I}}\ \theta\ B^{\mathcal{I}} \Leftrightarrow$<br>$X \,\Theta\, B$ |
| *X overlaps A* | $X\ o\ A \sqcap A \equiv \neg B \rightarrow X \,\Theta\, B$ |
| | If X overlaps with A and A is the complement of B, X has a possible overlap with B. |
| | $X\ o\ A \sqcap A \equiv \neg B \Leftrightarrow$<br>$X^{\mathcal{I}}\ O\ A^{\mathcal{I}} \wedge A^{\mathcal{I}} = \neg B^{\mathcal{I}} \Rightarrow$<br>$X^{\mathcal{I}}\ \theta\ B^{\mathcal{I}} \Leftrightarrow$<br>$X \,\Theta\, B$ |
| *X possible overlap A* | $X \,\Theta\, A \sqcap A \equiv \neg B \rightarrow X \,\Theta\, B$ |
| | If X has a possible overlap with A and A is the complement of B, X has a possible overlap with B. |
| | $X \,\Theta\, A \sqcap A \equiv \neg B \Leftrightarrow$<br>$X^{\mathcal{I}}\ \theta\ A^{\mathcal{I}} \wedge A^{\mathcal{I}} = \neg B^{\mathcal{I}} \Rightarrow$<br>$X^{\mathcal{I}}\ \theta\ B^{\mathcal{I}} \Leftrightarrow$<br>$X \,\Theta\, B$ |

Table C.12: $A \equiv \neg B$, part 1

| *B equals X* | $B \equiv X \sqcap A \equiv \neg B \rightarrow A \equiv \neg X$ |
|---|---|
| | If B equals X and A is the complement of B, A equals the complement of X |
| | $B \equiv X \sqcap A \equiv \neg B \Leftrightarrow$<br>$B^{\mathcal{I}} = X^{\mathcal{I}} \wedge A^{\mathcal{I}} = \neg B^{\mathcal{I}} \Rightarrow$<br>$A^{\mathcal{I}} = \neg X^{\mathcal{I}} \Leftrightarrow$<br>$A \equiv \neg X$ |
| *B is contained in X* | $B \sqsubseteq X \sqcap A \equiv \neg B \rightarrow A \ \Theta \ X$ |
| | If B is more specific than X and A is the complement of B, A has a possible overlap with X |
| | $B \sqsubseteq X \sqcap A \equiv \neg B \Leftrightarrow$<br>$B^{\mathcal{I}} \subseteq X^{\mathcal{I}} \wedge A^{\mathcal{I}} = \neg B^{\mathcal{I}} \Rightarrow$<br>$A^{\mathcal{I}} \ \theta \ X^{\mathcal{I}}$<br>$A \ \Theta \ X$ |
| *B contains X* | $B \sqsupseteq X \sqcap A \equiv \neg B \rightarrow \neg X \sqsupseteq A$ |
| | If B is less specific than X and A is the complement of B, the complement of X contains A. |
| | $B \sqsupseteq X \sqcap A \equiv \neg B \Leftrightarrow$<br>$B^{\mathcal{I}} \supseteq X^{\mathcal{I}} \wedge A^{\mathcal{I}} = \neg B^{\mathcal{I}} \Rightarrow$<br>$\neg X^{\mathcal{I}} \supseteq A^{\mathcal{I}} \Leftrightarrow$<br>$\neg X \sqsupseteq A$ |
| *B overlaps X* | $B \ o \ X \sqcap A \equiv \neg B \rightarrow A \ \Theta \ X$ |
| | If B overlaps with X and A is the complement of B, A has a possible overlap with X |
| | $B \ o \ X \sqcap A \equiv \neg B \Leftrightarrow$<br>$B^{\mathcal{I}} \ O \ X^{\mathcal{I}} \wedge A^{\mathcal{I}} = \neg B^{\mathcal{I}} \Rightarrow$<br>$A^{\mathcal{I}} \ \theta \ X^{\mathcal{I}}$<br>$A \ \Theta \ X$ |
| *B possible overlap X* | $B \ \Theta \ X \sqcap A \equiv \neg B \rightarrow A \ \Theta \ X$ |
| | If B has a possible overlap with X and A is the complement of B, A also has a possible overlap with X |
| | $B \ \Theta \ X \sqcap A \equiv \neg B \Leftrightarrow$<br>$B^{\mathcal{I}} \ \theta \ X^{\mathcal{I}} \wedge A^{\mathcal{I}} = \neg B^{\mathcal{I}} \Rightarrow$<br>$A^{\mathcal{I}} \ \theta \ X^{\mathcal{I}}$<br>$A \ \Theta \ X$ |

Table C.13: $A \equiv \neg B$, part 2

## C.3 Case Study - Process Modelling

### C.3.1 Bootstrap Model Transformation

Listing C.2: The bootstrap model transformation in QVT Relations syntax

```
transformation ProcessToProcess(prc_1:Process; prc_2:Process) {

  key process::Process {name};
  key process::Step {name};
  key process::Task {name, namespace};
  key process::Decision {name, namespace};
  key process::Merge {name, namespace};
  key process::Flow {name, namespace};

  top relation ProcessToProcess {
    n: String;
    checkonly domain prc_1 p_1:Process {
      name=n
      };
    enforce domain prc_2 p_2:Process {
      name=n
      };
  }

  top relation TaskToTask {
    n: String;
    checkonly domain prc_1 t_1:Task {
      namespace=p_1:Process {},
      name=n
      };
    enforce domain prc_2 t_2:Task {
      namespace=p_2:Process {},
      name=n
      };
    when
    {
      ProcessToProcess(p_1,p_2);
    }
    where {
      StepToStep_out(t_1,t_2);
      StepToStep_in(t_1,t_2);
    }
  }

  top relation DecisionToDecision {
    n: String;
    checkonly domain prc_1 d_1:Decision {
      namespace=p_1:Process {},
      name=n
      };
    enforce domain prc_2 d_2:Decision {
      namespace=p_2:Process {},
      name=n
      };
    when
    {
      ProcessToProcess(p_1,p_2);
    }
```

```
    where
    {
      StepToStep_out(d_1,d_2);
      StepToStep_in(d_1,d_2);
    }
  }

  top relation MergeToMerge {
    n: String;
    checkonly domain prc_1 m_1:Merge {
      namespace=p_1:Process {},
      name=n
      };
    enforce domain prc_2 m_2:Merge {
      namespace=p_2:Process {},
      name=n
      };
    when
    {
      ProcessToProcess(p_1,p_2);
    }
    where
    {
      StepToStep_out(m_1,m_2);
      StepToStep_in(m_1,m_2);
    }
  }

  relation StepToStep_out {
    n: String;
    checkonly domain prc_1 s_1:Step {
      outFlow=out_1:Flow {
        name=n
        }
      };
    enforce domain prc_2 s_2:Step {
      outFlow=out_2:Flow {
        name=n
        }
      };
  }

  relation StepToStep_in {
    n: String;
    checkonly domain prc_1 s_1:Step {
      inFlow=in_1:Flow {
        name=n
        }
      };
    enforce domain prc_2 s_2:Step {
      inFlow=in_2:Flow {
        name=n
        }
      };
  }

  top relation FlowToFlow {
    n: String;
    checkonly domain prc_1 f_1:Flow {
```

```
      namespace=p_1:Process {},
      name=n
      };
    enforce domain prc_2 f_2:Flow {
      namespace=p_2:Process {},
      name=n
      };
    when
    {
      ProcessToProcess(p_1,p_2);
    }
    where
    {
      Flow2Flow_sink(f_1,f_2);
      Flow2Flow_source(f_1,f_2);
    }
  }

  relation Flow2Flow_sink {
    n: String;
    checkonly domain prc_1 f_1:Flow {
      sink=sink_1:Step {
        name=n
        }
      };
    enforce domain prc_2 f_2:Flow {
      sink=sink_2:Step {
        name=n
        }
      };
  }

  relation Flow2Flow_source {
    n: String;
    checkonly domain prc_1 f_1:Flow {
      source=source_1:Step {
        name=n
        }
      };
    enforce domain prc_2 f_2:Flow {
      source=source_2:Step {
        name=n
        }
      };
  }
}
```

### C.3.2 Reasoning Results

Listing C.3: Reasoning results for classes

```
<urn:ua:pvs/Process> <equal> <urn:ua:pvs/Epc>
<urn:ua:pvs/Process> <possibleoverlap> <urn:ua:pvs/Join>
<urn:ua:pvs/Process> <possibleoverlap> <urn:ua:pvs/ControlElement>
<urn:ua:pvs/Process> <possibleoverlap> <urn:ua:pvs/Split>
<urn:ua:pvs/Process> <possibleoverlap> <urn:ua:pvs/Epcelement>
<urn:ua:pvs/Process> <possibleoverlap> <urn:ua:pvs/Function>

<urn:ua:pvs/Step> <equal> <urn:ua:pvs/Epcelement>
<urn:ua:pvs/Step> <general> <urn:ua:pvs/Join>
<urn:ua:pvs/Step> <general> <urn:ua:pvs/Function>
<urn:ua:pvs/Step> <general> <urn:ua:pvs/Split>
<urn:ua:pvs/Step> <general> <urn:ua:pvs/ControlElement>
<urn:ua:pvs/Step> <possibleoverlap> <urn:ua:pvs/Epc>

<urn:ua:pvs/Task> <equal> <urn:ua:pvs/Function>
<urn:ua:pvs/Task> <specific> <urn:ua:pvs/Epcelement>
<urn:ua:pvs/Task> <possibleoverlap> <urn:ua:pvs/Split>
<urn:ua:pvs/Task> <possibleoverlap> <urn:ua:pvs/Join>
<urn:ua:pvs/Task> <possibleoverlap> <urn:ua:pvs/Epc>
<urn:ua:pvs/Task> <possibleoverlap> <urn:ua:pvs/ControlElement>

<urn:ua:pvs/Merge> <equal> <urn:ua:pvs/Join>
<urn:ua:pvs/Merge> <specific> <urn:ua:pvs/ControlElement>
<urn:ua:pvs/Merge> <specific> <urn:ua:pvs/Epcelement>
<urn:ua:pvs/Merge> <possibleoverlap> <urn:ua:pvs/Split>
<urn:ua:pvs/Merge> <possibleoverlap> <urn:ua:pvs/Function>
<urn:ua:pvs/Merge> <possibleoverlap> <urn:ua:pvs/Epc>

<urn:ua:pvs/Decision> <equal> <urn:ua:pvs/Split>
<urn:ua:pvs/Decision> <specific> <urn:ua:pvs/ControlElement>
<urn:ua:pvs/Decision> <specific> <urn:ua:pvs/Epcelement>
<urn:ua:pvs/Decision> <possibleoverlap> <urn:ua:pvs/Join>
<urn:ua:pvs/Decision> <possibleoverlap> <urn:ua:pvs/Function>
<urn:ua:pvs/Decision> <possibleoverlap> <urn:ua:pvs/Epc>

<urn:ua:pvs/Flow> <equal> <urn:ua:pvs/Connector>
```

Listing C.4: Reasoning results for associations

```
<urn:ua:pvs/Process:flows> <equal> <urn:ua:pvs/Epc:connectors>

<urn:ua:pvs/Process:steps> <general> <urn:ua:pvs/Epc:controlelements>
<urn:ua:pvs/Process:steps> <general> <urn:ua:pvs/Epc:functions>

<urn:ua:pvs/Step:namespace> <general> <urn:ua:pvs/Function:namespace>
<urn:ua:pvs/Step:namespace> <general> <urn:ua:pvs/Split:namespace>
<urn:ua:pvs/Step:namespace> <general> <urn:ua:pvs/Join:namespace>

<urn:ua:pvs/Step:outFlow> <general> <urn:ua:pvs/Function:outConnectorF>
<urn:ua:pvs/Step:outFlow> <general> <urn:ua:pvs/Join:outConnectorJ>
<urn:ua:pvs/Step:outFlow> <general> <urn:ua:pvs/Split:outConnectorS>

<urn:ua:pvs/Step:inFlow> <general> <urn:ua:pvs/Function:inConnectorF>
<urn:ua:pvs/Step:inFlow> <general> <urn:ua:pvs/Join:inConnectorJ>
<urn:ua:pvs/Step:inFlow> <general> <urn:ua:pvs/Split:inConnectorS>

<urn:ua:pvs/Flow:namespace> <general> <urn:ua:pvs/Connector:namespace>

<urn:ua:pvs/Flow:sink> <general> <urn:ua:pvs/Connector:targetJoin>
<urn:ua:pvs/Flow:sink> <general> <urn:ua:pvs/Connector:targetFunction>
<urn:ua:pvs/Flow:sink> <general> <urn:ua:pvs/Connector:targetSplit>

<urn:ua:pvs/Flow:source> <general> <urn:ua:pvs/Connector:sourceJoin>
```

```
<urn:ua:pvs/Flow:source> <general> <urn:ua:pvs/Connector:sourceFunction>
<urn:ua:pvs/Flow:source> <general> <urn:ua:pvs/Connector:sourceSplit>
```

Listing C.5: Reasoning results for properties

```
<urn:ua:pvs/Process:name> <equal> <urn:ua:pvs/Epc:name>
<urn:ua:pvs/Process:name> <possibleoverlap> <urn:ua:pvs/Split:name>
<urn:ua:pvs/Process:name> <possibleoverlap> <urn:ua:pvs/Epc:name>
<urn:ua:pvs/Process:name> <possibleoverlap> <urn:ua:pvs/Epcelement:name>
<urn:ua:pvs/Process:name> <possibleoverlap> <urn:ua:pvs/Join:name>
<urn:ua:pvs/Process:name> <possibleoverlap> <urn:ua:pvs/ControlElement:name>
<urn:ua:pvs/Process:name> <possibleoverlap> <urn:ua:pvs/Function:name>

<urn:ua:pvs/Task:name> <equal> <urn:ua:pvs/Function:name>
<urn:ua:pvs/Task:name> <specific> <urn:ua:pvs/Epcelement:name>
<urn:ua:pvs/Task:name> <possibleoverlap> <urn:ua:pvs/Epc:name>
<urn:ua:pvs/Task:name> <possibleoverlap> <urn:ua:pvs/Join:name>
<urn:ua:pvs/Task:name> <possibleoverlap> <urn:ua:pvs/Function:name>
<urn:ua:pvs/Task:name> <possibleoverlap> <urn:ua:pvs/Split:name>
<urn:ua:pvs/Task:name> <possibleoverlap> <urn:ua:pvs/ControlElement:name>

<urn:ua:pvs/Decision:name> <equal> <urn:ua:pvs/Split:name>
<urn:ua:pvs/Decision:name> <specific> <urn:ua:pvs/ControlElement:name>
<urn:ua:pvs/Decision:name> <specific> <urn:ua:pvs/Epcelement:name>
<urn:ua:pvs/Decision:name> <possibleoverlap> <urn:ua:pvs/Function:name>
<urn:ua:pvs/Decision:name> <possibleoverlap> <urn:ua:pvs/EPC:name>
<urn:ua:pvs/Decision:name> <possibleoverlap> <urn:ua:pvs/Split:name>
<urn:ua:pvs/Decision:name> <possibleoverlap> <urn:ua:pvs/Join:name>

<urn:ua:pvs/Merge:name> <equal> <urn:ua:pvs/Join:name>
<urn:ua:pvs/Merge:name> <specific> <urn:ua:pvs/ControlElement:name>
<urn:ua:pvs/Merge:name> <specific> <urn:ua:pvs/Epcelement:name>
<urn:ua:pvs/Merge:name> <possibleoverlap> <urn:ua:pvs/Epc:name>
<urn:ua:pvs/Merge:name> <possibleoverlap> <urn:ua:pvs/Split:name>
<urn:ua:pvs/Merge:name> <possibleoverlap> <urn:ua:pvs/Join:name>
<urn:ua:pvs/Merge:name> <possibleoverlap> <urn:ua:pvs/Function:name>

<urn:ua:pvs/Flow:name> <equal> <urn:ua:pvs/Connector:name>
```

### C.3.3 Generated Model Transformation

Listing C.6: The generated model transformation in QVT Relations syntax

```
transformation ProcessToProcess(prc_1:Process; prc_2:Epc) {

  key Epc::Epc {name};
  key Epc::Function {name, namespace};
  key Epc::Split {name, namespace};
  key Epc::Join {name, namespace};
  key Epc::Connector {name, namespace};

  top relation ProcessToProcess {
    n: String;
    checkonly domain prc_1 p_1:Process {
      name=n
      };
    enforce domain prc_2 p_2:Epc {
      name=n
      };
  }

  top relation TaskToTask {
    n: String;
    checkonly domain prc_1 t_1:Task {
      namespace=p_1:Process {},
      name=n
      };
    enforce domain prc_2 t_2:Function {
      namespace=p_2:Epc {},
      name=n
      };
    when
    {
      ProcessToProcess(p_1,p_2);
    }
    where {
      StepToStep_out_1(t_1,t_2);
      StepToStep_in_1(t_1,t_2);
    }
  }

  top relation DecisionToDecision {
    n: String;
    checkonly domain prc_1 d_1:Decision {
      namespace=p_1:Process {},
      name=n
      };
    enforce domain prc_2 d_2:Split {
      namespace=p_2:Epc {},
      name=n
      };
    when
    {
      ProcessToProcess(p_1,p_2);
    }
    where
    {
      StepToStep_out_2(d_1,d_2);
```

```
56        StepToStep_in_2(d_1,d_2);
      }
58    }

60    top relation MergeToMerge {
        n: String;
62      checkonly domain prc_1 m_1:Merge {
          namespace=p_1:Process {},
64        name=n
          };
66      enforce domain prc_2 m_2:Join {
          namespace=p_2:Epc {},
68        name=n
          };
70      when
        {
72        ProcessToProcess(p_1,p_2);
        }
74      where
        {
76        StepToStep_out_3(m_1,m_2);
          StepToStep_in_3(m_1,m_2);
78      }
      }
80
      relation StepToStep_out_1 {
82      n: String;
        checkonly domain prc_1 s_1:Step {
84        outFlow=out_1:Flow {
            name=n
86          }
          };
88      enforce domain prc_2 s_2:Function {
          outConnectorF=out_2:Connector {
90          name=n
            }
92        };
      }
94
      relation StepToStep_out_2 {
96      n: String;
        checkonly domain prc_1 s_1:Step {
98        outFlow=out_1:Flow {
            name=n
100         }
          };
102     enforce domain prc_2 s_2:Split {
          outConnectorS=out_2:Connector {
104         name=n
            }
106       };
      }
108
      relation StepToStep_out_3 {
110     n: String;
        checkonly domain prc_1 s_1:Step {
112       outFlow=out_1:Flow {
            name=n
114         }
```

```
      };
    enforce domain prc_2 s_2:Join {
      outConnectorJ=out_2:Connector {
        name=n
        }
      };
  }

  relation StepToStep_in_1 {
    n: String;
    checkonly domain prc_1 s_1:Step {
      inFlow=in_1:Flow {
        name=n
        }
      };
    enforce domain prc_2 s_2:Function {
      inConnectorF=in_2:Connector {
        name=n
        }
      };
  }

  relation StepToStep_in_2 {
    n: String;
    checkonly domain prc_1 s_1:Step {
      inFlow=in_1:Flow {
        name=n
        }
      };
    enforce domain prc_2 s_2:Split {
      inConnectorS=in_2:Connector {
        name=n
        }
      };
  }

  relation StepToStep_in_3 {
    n: String;
    checkonly domain prc_1 s_1:Step {
      inFlow=in_1:Flow {
        name=n
        }
      };
    enforce domain prc_2 s_2:Join {
      inConnectorJ=in_2:Connector {
        name=n
        }
      };
  }

  top relation FlowToFlow {
    n: String;
    checkonly domain prc_1 f_1:Flow {
      namespace=p_1:Process {},
      name=n
      };
    enforce domain prc_2 f_2:Connector {
      namespace=p_2:Epc {},
      name=n
```

```
174       };
      when
176     {
          ProcessToProcess(p_1,p_2);
178     }
      where
180     {
          Flow2Flow_sink(f_1,f_2)_1;
182       Flow2Flow_sink(f_1,f_2)_2;
          Flow2Flow_sink(f_1,f_2)_3;
184       Flow2Flow_source(f_1,f_2)_1;
          Flow2Flow_source(f_1,f_2)_2;
186       Flow2Flow_source(f_1,f_2)_3;
    }
188 }

190 relation Flow2Flow_sink_1 {
    n: String;
192   checkonly domain prc_1 f_1:Flow {
      sink=sink_1:Step {
194       name=n
          }
196     };
    enforce domain prc_2 f_2:FConnector {
198     targetFunction=sink_2:Function {
          name=n
200       }
      };
202 }

204 relation Flow2Flow_sink_2 {
    n: String;
206   checkonly domain prc_1 f_1:Flow {
      sink=sink_1:Step {
208       name=n
          }
210     };
    enforce domain prc_2 f_2:Connector {
212     targetSplit=sink_2:Split {
          name=n
214       }
      };
216 }

218 relation Flow2Flow_sink_3 {
    n: String;
220   checkonly domain prc_1 f_1:Flow {
      sink=sink_1:Step {
222       name=n
          }
224     };
    enforce domain prc_2 f_2:Connector {
226     targetJoin=sink_2:Join {
          name=n
228       }
      };
230 }

232 relation Flow2Flow_source_1 {
```

```
    n: String;
    checkonly domain prc_1 f_1:Flow {
      source=source_1:Step {
        name=n
        }
      };
    enforce domain prc_2 f_2:Connector {
      sourceFunction=source_2:Function {
        name=n
        }
      };
  }

  relation Flow2Flow_source_2 {
    n: String;
    checkonly domain prc_1 f_1:Flow {
      source=source_1:Step {
        name=n
        }
      };
    enforce domain prc_2 f_2:Connector {
      sourceSplit=source_2:Split {
        name=n
        }
      };
  }

  relation Flow2Flow_source_3 {
    n: String;
    checkonly domain prc_1 f_1:Flow {
      source=source_1:Step {
        name=n
        }
      };
    enforce domain prc_2 f_2:Connector {
      sourceJoin=source_2:Join {
        name=n
        }
      };
  }
}
```

## C.4 Case Study - Service Modelling

### C.4.1 Initial Model Transformation

Listing C.7: The input model transformation in QVT relational syntax

```
transformation SPL4AOXtoPIM4SOA(aox:SPL4AOX; soa:PIM4SOA) {

  key PIM4SOA::PIM4SOAmodel {name};
  key PIM4SOA::Collaboration {name, namespace};
  key PIM4SOA::CollaborationUse {name};
  key PIM4SOA::ServiceProvider {name, namespace};
  key PIM4SOA::Role {name, roleOwner};
  key PIM4SOA::RoleBinding {name, namespace};

  top relation SPL4AOXToPIM4SOAmodel {
    n: String;
    checkonly domain aox a_1:SPL4AOXmodel {
      name=n
    };
    enforce domain soa s_1:PIM4SOAmodel {
      name=n
    };
  }

  top relation CollaborationToCollaboration {
    n: String;
    checkonly domain aox a_1:ServiceCollaboration {
      namespace=am_1:SPL4AOXmodel {},
      name=n
    };
    enforce domain soa s_1:Collaboration {
      namespace=sm_1:PIM4SOAmodel {},
      name=n
    };
    when
    {
      SPL4AOXToPIM4SOAmodel(am_1,sm_1);
    }
    where
    {
      CollaborationRoleToCollaborationRole(a_1,s_1);
    }
  }

  relation CollaborationRoleToCollaborationRole {
    n: String;
    checkonly domain aox a_1:ServiceCollaboration {
      role=ar_1:Role {
        name=n
      }
    };
    enforce domain soa s_1:Collaboration {
      roles=sr_1:Role {
        name=n,
        roleOwner='C'
      }
    };
  }
```

```
54
  top relation CollaborationUseToCollaborationUse_Collaboration {
56  n: String;
    checkonly domain aox a_1:CollaborationUse {
58    namespace=ac_1:ServiceCollaboration {},
      service=ac_2:ServiceCollaboration {},
60    name=n
    };
62  enforce domain soa s_1:CollaborationUse {
      namespace2=sc_1:Collaboration {},
64    collaboration=sc_2:Collaboration {},
      name=n
66  };
    when
68  {
      CollaborationToCollaboration(ac_1,sc_1);
70    CollaborationToCollaboration(ac_2,sc_2);
    }
72  }

74  top relation RoleBindingToRoleBinding_Collaboration {
    n: String;
76  checkonly domain aox a_1:RoleBinding {
      namespace=arb_1:CollaborationUse {},
78    name=n
      };
80  enforce domain soa s_1:RoleBinding {
      namespace=srb_1:CollaborationUse {},
82    name=n
      };
84  when
    {
86    CollaborationUseToCollaborationUse_Collaboration(arb_1,srb_1);
    }
88  where
    {
90    RoleBindingToRoleBinding_role_Collaboration(a_1,s_1);
      RoleBindingToRoleBinding_boundRole_Collaboration(a_1,s_1);
92  }
  }
94

  relation RoleBindingToRoleBinding_role_Collaboration {
96  n,nr: String;
    checkonly domain aox a_1:RoleBinding {
98    role=ar_1:Role {
        name=nr
100   }
    };
102 enforce domain soa s_1:RoleBinding {
      role=sr_1:Role {
104     name=nr,
        roleOwner='C'
106   }
    };
108 }

110 relation RoleBindingToRoleBinding_boundRole_Collaboration {
    n,nr: String;
112 checkonly domain aox a_1:RoleBinding {
```

```
      boundRole=ar_1:Role {
        name=nr
      }
    };
    enforce domain soa s_1:RoleBinding {
      boundRole=sr_1:Role {
        name=nr,
        roleOwner='C'
      }
    };
  }

  top relation ServiceProviderToServiceProvider {
    n,nr: String;
    checkonly domain aox a_1:ServiceProvider {
      namespace=am_1:SPL4AOXmodel {},
      name=n,
      role=ar:Role{
        name=nr,
        usingService=ac:ServiceCollaboration {}
      }
    };
    enforce domain soa s_1:ServiceProvider {
      namespace=sm_1:PIM4SOAmodel {},
      name=n,
      participates=scu:CollaborationUse {
        name=n+'_SP',
        collaboration=sc:Collaboration {},
        bindings=srb:RoleBinding {
          name=n+'_RB'
        }
      }
    };
    when
    {
      SPL4AOXToPIM4SOAmodel(am_1,sm_1);
      CollaborationToCollaboration(ac,sc);
    }
    where
    {
      roleBindingSP_Provider(ar,srb);
      roleBindingSP_Requestor(ar,srb);
    }
  }

  relation roleBindingSP_Provider {
    nr: String;
    checkonly domain aox a_1:Role {
      name=nr,
      roleType='PROVIDER'
    };
    enforce domain soa s_1:RoleBinding {
      role=sRole:Role{
        name=nr,
        roleOwner='C'
      },
      boundRole=sBoundRole:Role {
        name=nr,
        roleOwner='SP',
```

```
172         roleType='PROVIDER'
        }
174     };
    }
176
      relation roleBindingSP_Requestor {
178     nr: String;
      checkonly domain aox a_1:Role {
180       name=nr,
        roleType='REQUESTOR'
182     };
      enforce domain soa s_1:RoleBinding {
184       role=sRole:Role{
          name=nr,
186         roleOwner='C'
        },
188       boundRole=sBoundRole:Role {
          name=nr,
190         roleOwner='SP',
          roleType='REQUESTOR'
192       }
      };
194   }

196   top relation RoleToRole_Provider {
      n: String;
198     checkonly domain aox a_1:Role {
        namespace=asp_1:ServiceProvider {},
200       name=n,
        roleType='PROVIDER'
202     };
      enforce domain soa s_1:Role {
204       namespace1=ssp_1:ServiceProvider {},
        name=n,
206       roleType='PROVIDER',
        roleOwner='SP'
208     };
      when
210     {
        ServiceProviderToServiceProvider(asp_1,ssp_1);
212     }
    }
214
    top relation RoleToRole_Requestor {
216     n: String;
      checkonly domain aox a_1:Role {
218       namespace=asp_1:ServiceProvider {},
        name=n,
220       roleType='REQUESTOR'
      };
222     enforce domain soa s_1:Role {
        namespace1=ssp_1:ServiceProvider {},
224       name=n,
        roleType='REQUESTOR',
226       roleOwner='SP'
      };
228     when
      {
230       ServiceProviderToServiceProvider(asp_1,ssp_1);
```

```
        }
232   }
}
```

### C.4.2 Output Model Transformation

Listing C.8: The output model transformation in QVT relational syntax

```
transformation SPL4AOXtoUPMS(aox:SPL4AOX; soa:UPMS) {

  key UPMS::UPMSmodel {name};
  key UPMS::Contract {name, namespace};
  key UPMS::Fulfillment {name};
  key UPMS::Component {name, namespace};
  key UPMS::Role {name};
  key UPMS::Service {name};
  key UPMS::Requisition {name};
  key UPMS::Dependency {name, namespace};

  top relation SPL4AOXToUPMSmodel {
    n: String;
    checkonly domain aox a_1:SPL4AOXmodel {
      name=n
    };
    enforce domain soa s_1:UPMSmodel {
      name=n
    };
  }

  top relation CollaborationToContract {
    n: String;
    checkonly domain aox a_1:ServiceCollaboration {
      namespace=am_1:SPL4AOXmodel {},
      name=n
    };
    enforce domain soa s_1:Contract {
      namespace=sm_1:UPMSmodel {},
      name=n
    };
    when
    {
      SPL4AOXToUPMSmodel(am_1, sm_1);
    }
    where
    {
      CollaborationRoleToContractRole(a_1, s_1);
    }
  }

  relation CollaborationRoleToContractRole {
    n: String;
    checkonly domain aox a_1:ServiceCollaboration {
      role=ar_1:Role {
        name=n
      }
    };
    enforce domain soa s_1:Contract {
      parts=sr_1:Role {
        name=n
      }
    };
  }
```

```
top relation CollaborationUseToFulfillment_Collaboration {
  n: String;
  checkonly domain aox a_1:CollaborationUse {
    namespace=ac_1:ServiceCollaboration {},
    service=ac_2:ServiceCollaboration {},
    name=n
  };
  enforce domain soa s_1:Fulfillment {
    namespace2=sc_1:Contract {},
    contractUse=sc_2:Contract {},
    name=n
  };
  when
  {
    CollaborationToContract(ac_1, sc_1);
    CollaborationToContract(ac_2, sc_2);
  }
}

top relation RoleBindingToDependency_Collaboration {
  n: String;
  checkonly domain aox a_1:RoleBinding {
    namespace=arb_1:CollaborationUse {},
    name=n
  };
  enforce domain soa s_1:Dependency {
    namespace=srb_1:Fulfillment {},
    name=n
  };
  when
  {
    CollaborationUseToFulfillment_Collaboration(arb_1, srb_1);
  }
  where
  {
    RoleBindingToDependency_role_Collaboration(a_1, s_1);
    RoleBindingToDependency_boundRole_Collaboration(a_1, s_1);
  }
}

relation RoleBindingToDependency_role_Collaboration {
  n,nr: String;
  checkonly domain aox a_1:RoleBinding {
    role=ar_1:Role {
      name=nr
    }
  };
  enforce domain soa s_1:Dependency {
    role=sr_1:Role {
      name=nr
    }
  };
}

relation RoleBindingToDependency_boundRole_Collaboration {
  n,nr: String;
  checkonly domain aox a_1:RoleBinding {
    boundRole=ar_1:Role {
      name=nr
```

```
      }
    };
    enforce domain soa s_1:Dependency {
      boundRole=sr_1:Role {
        name=nr
      }
    };
  }

  top relation ServiceProviderToComponent
  {
    n,nr: String;
    checkonly domain aox a_1:ServiceProvider {
      namespace=am_1:SPL4AOXmodel {},
      name=n,
      role=ar:Role {
        name=nr,
        usingService=ac:ServiceCollaboration{}
      }
    };
    enforce domain soa s_1:Component {
      namespace=sm_1:UPMSmodel {},
      name=n,
      contracts=scu:Fulfillment {
        name=n+'_SP',
        contractUse=sc:Contract {},
        roleBinding=srb:Dependency {
          name=n+'_RB'
        }
      }
    };
    when
    {
      SPL4AOXToUPMSmodel(am_1, sm_1);
      CollaborationToContract(ac, sc);
    }
    where
    {
      roleBindingSP_Service(ar, srb);
      roleBindingSP_Requisition(ar, srb);
    }
  }

  relation roleBindingSP_Service {
    nr: String;
    checkonly domain aox a_1:Role {
      name=nr,
      roleType='PROVIDER'
    };
    enforce domain soa s_1:Dependency {
      role=sRole:Role{
        name=nr
      },
      boundRoleService=sBoundRole:Service {
        name=nr
      }
    };
  }
```

```
  relation roleBindingSP_Requisition {
    nr: String;
    checkonly domain aox a_1:Role {
      name=nr,
      roleType='REQUESTOR'
    };
    enforce domain soa s_1:Dependency {
      role=sRole:Role{
        name=nr
      },
      boundRoleRequisition=sBoundRole:Requisition {
        name=nr
      }
    };
  }

  top relation RoleToService {
    n: String;
    checkonly domain aox a_1:Role {
      namespace=asp_1:ServiceProvider {},
      name=n,
      roleType='PROVIDER'
    };

    enforce domain soa s_1:Service {
      namespace=ssp_1:Component {},
      name=n
    };
    when
    {
      ServiceProviderToComponent(asp_1, ssp_1);
    }
  }

  top relation RoleToRequisition {
    n: String;
    checkonly domain aox a_1:Role {
      namespace=asp_1:ServiceProvider {},
      name=n,
      roleType='REQUESTOR'
    };
    enforce domain soa s_1:Requisition {
      namespace=ssp_1:Component {},
      name=n
    };
    when
    {
      ServiceProviderToComponent(asp_1, ssp_1);
    }
  }
}
```

# Curriculum Vitae

| | |
|---|---|
| **Name** | Stephan Roser |
| **Date of birth** | 25th August 1979 |
| **Place of birth** | Augsburg |
| **Nationality** | German |

| | |
|---|---|
| **since 10/2004** | Research assistant at the Department of Computer Science (University of Augsburg) |
| **10/2001 - 09/2004** | Studies in Applied Computer Science (University of Augsburg)<br>Degree: Diploma |
| **05/2000 - 09/2001** | Studies in Economathematics (University of Augsburg)<br>Degree: Intermediate diploma |
| **07/1999 - 04/2000** | Basic military service (Bad Reichenhall) |
| **09/1990 - 06/1999** | Secondary school (Neusäß)<br>Leaving certificate: Abitur |
| **09/1986 - 07/1990** | Primary school (Neusäß) |