“QoS-aware model-driven SOA using SoaML”

Niels Schot

A thesis submitted for the degree of
MSc Computer Science

University of Twente
EEMCS - TRESE: Software Engineering Group

Examination committee:
Luís Ferreira Pires
Marten van Sinderen

December 2012 *version 1.00*

Abstract

This project considers the application of Model-Driven Engineering (MDE) techniques, like metamodeling and model transformations, to Service-Oriented Architecture (SOA). SOA modeling languages should support the specification of quality aspects, so that conditions, constraints and requirements on these aspects can be specified in service models. Examples of quality aspects are reliability, performance, availability, security, cost and integrity. A concrete quality requirement could be that the service response time must not exceed two seconds.

The term quality of service (QoS) is often used to refer to the collection of quality requirements on a service execution. A policy is associated with the point of view of each individual participant in service interactions. For instance, the policy of a service provider defines the quality aspects of its provided service, and should be taken into consideration in the service contract between this service provider and a service consumer, in case a service consumer agrees with the policy.

During the development of a SOA application, several activities are performed by various stakeholders. For instance, a service provider needs to describe the offers of the provided service. An example of an activity for the service provider is to express the QoS offers with, for instance, a policy. Another example is the specification of service contracts which describe the agreement on QoS requirements. A lot of these activities can be optimized by the application of MDE in terms of productivity, understandability and consistency.

The main goal of this project is to investigate how to include, define and use QoS definitions in model-driven SOA, which requires the definition of quality requirements in both platform-independent (PIM) and platform-specific (PSM) models. The SOA Modeling Language (SoaML) is a standard to model SOA applications at PIM-level, which has been recently released and looks promising. SoaML is chosen in this project to model the PIM-level models. These SoaML models have to be transformed to PSM-level models in order to realize and benefit from MDE. For instance, the QoS-aware SoaML models are mapped to WS-Policy to realize a model transformation which can be used to automatically generate policy documents. A second transformation is made to generate pre-filled WS-Agreement templates based on service contract specifications in these SoaML models.

Preface

During the last months of my master, I have worked extensively on this thesis to finish my study at the University of Twente. Now I have completed my thesis, I would like to thank some people who helped during this period.

First, I would like to thank my supervisors Luís and Marten for their useful reviews and discussions about this project. The project offered a lot of directions and possibilities, and they helped me to choose the right approach to perform this research. The numerous meetings really helped me to distinguish the main points from the side issues of this project.

Last but not least, I would like my girlfriend Annerie, family and friends for their support and the patience they showed for me while I was working on this thesis behind my computer.

December 7, 2012
Niels Schot

Contents

Abstract i

Preface ii

List of Figures 4

List of Tables 5

Glossary 6

1 Introduction 7
- 1.1 Motivation . . . 7
- 1.2 Problem statement . . . 8
- 1.3 Objectives . . . 9
- 1.4 Research questions . . . 9
- 1.5 Approach . . . 10
- 1.6 Scope of the research . . . 11
- 1.7 Structure of report . . . 11

2 Service models 13
- 2.1 Service life-cycle . . . 13
- 2.2 Stakeholders . . . 15
- 2.3 Service developer . . . 15
- 2.4 Service provider . . . 17
- 2.5 Service consumer . . . 19
- 2.6 Application provider . . . 20
- 2.7 Service broker . . . 20
- 2.8 Composition designer . . . 21
- 2.9 Quality aspects . . . 23
- 2.10 Optimizable activities . . . 25

3 SOA modeling 28
- 3.1 Relevant MDE principles for SOA . . . 28
- 3.2 PIM-level language selection . . . 29
- 3.3 SoaML concepts . . . 35
- 3.4 SoaML tooling . . . 37
  - 3.4.1 Selection . . . 38

4 QoS specifications in SoaML 40
- 4.1 Case study . . . 40
- 4.2 The scope of SoaML . . . 41
- 4.3 Available QoS modeling approaches . . . 41

4.3.1 SoaML UML constraints . . . 42
4.3.2 QoS languages . . . 46
4.3.3 SoaML metamodel refinements . . . 50
4.4 Selection QoS modeling approach . . . 51
4.5 Implementing SoaML models . . . 52
4.5.1 Quality requirements . . . 53
4.5.2 Modeling policies . . . 53
4.5.3 Modeling QoS in service contracts . . . 55

5 PSM-level QoS modeling **57**
5.1 Search scope . . . 57
5.2 Available policy languages . . . 57
5.2.1 WS-Policy . . . 58
5.2.2 UDDI . . . 59
5.2.3 Web Services Policy Language (WSPL) . . . 60
5.2.4 Other techniques . . . 61
5.3 Selection of policy language . . . 61
5.4 Available service contract languages . . . 62
5.4.1 WS-Agreement . . . 62
5.4.2 Web Service Level Agreement (WSLA) . . . 64
5.4.3 SLA* . . . 65
5.4.4 SLAng . . . 66
5.4.5 Other techniques . . . 66
5.5 Selection of service contract language . . . 67
5.6 Implementing PSM models . . . 68

6 Model transformations **72**
6.1 Transformation environment . . . 72
6.1.1 Implementing the extended SoaML metamodel . . . 72
6.1.2 Rebuilding the SoaML profile models . . . 73
6.2 Mappings . . . 74
6.2.1 Policies . . . 74
6.2.2 Service contracts . . . 77
6.3 Transformation results . . . 80
6.3.1 Policies . . . 80
6.3.2 Service contracts . . . 81

7 Final remarks **83**
7.1 Related work . . . 83
7.2 Research results . . . 84
7.3 Future work . . . 84

References **92**

A ATL transformations **93**
A.1 SoaML2WSPolicy . . . 93
A.2 SoaML2WSAgreement . . . 97

List of Figures

1.1 Overview SOA application modeling process . . . 8
1.2 Research approach . . . 10

2.1 Terminology SOA application and service composition . . . 13
2.2 General view service of the life-cycle . . . 14
2.3 Service life-cycle service developer . . . 16
2.4 General service negotiation view . . . 17
2.5 Service life-cycle service provider . . . 18
2.6 Service life-cycle service consumer . . . 19
2.7 Service life-cycle activities application provider . . . 20
2.8 Service life-cycle service broker . . . 21
2.9 Service life-cycle service broker . . . 22
2.10 Service life-cycle service broker . . . 22
2.11 Service contract instantiation with policies . . . 24
2.12 QoS specification with policies . . . 26
2.13 Contract definition . . . 26

3.1 Example of a service interface defined in SoaML . . . 36

4.1 High-level view of the example . . . 40
4.2 Policy specification as UML constraint . . . 43
4.3 Policy specification as UML OCL constraint . . . 44
4.4 Part of the QoS&FT profile implementation in RSA . . . 46
4.5 Definition of availability characteristic . . . 47
4.6 Defining a QoS constraint with UML constraints . . . 47
4.7 Defining a QoS constraint with UML dependency and QoS value objects . . . 48
4.8 Evaluation QoS modeling techniques for SoaML . . . 51
4.9 The involved participants . . . 52
4.10 Availability and performance characteristics . . . 53
4.11 Linking QoS requirements to participant . . . 54
4.12 Nested QoS specifications . . . 55
4.13 Expressing QoS offers . . . 55
4.14 QoS specification in service contract . . . 56

5.1 Evaluation PSM-level policy languages . . . 61
5.2 Evaluation PSM-level contract languages . . . 67
5.3 Partial WS-Policy Ecore implementation . . . 68
5.4 Simple WS-Policy assertion grammar (l) and example model (r) . . . 69
5.5 Partial WS-Agreement Ecore implementation . . . 70
5.6 Structure WS-Agreement model [1] (l) and example model (r) . . . 70

6.1 Partial SoaML Ecore implementation . . . 72

6.2 Partial QoS&FT Ecore implementation 73
6.3 Profile-based to metamodel-based models 74
6.4 Properties of QoSDimension 74
6.5 Participant policy package to WS-Policy document 75
6.6 Dependency to alternatives 75
6.7 Separate policy subjects and nested policy 76
6.8 QoS values to quality aspects 76
6.9 QoS value data to quality attributes 77
6.10 Contract specification to WS-Agreement document 77
6.11 Service contract to context 78
6.12 Dependency to alternatives 78
6.13 Nested QoS specifications 79
6.14 QoS value to properties and guarantee terms 80
6.15 Transforming the SoaML models to WS-Policy 81
6.16 Transforming the SoaML models to WS-Agreement 82

List of Tables

3.1 SOA modeling approaches 30
3.2 SoaML tool support 37

4.1 Natural language UML constraints approach observations 43
4.2 OCL UML constraints approach observations 45
4.3 QoS&FT approach observations 48
4.4 UML-Q approach observations 49
4.5 Metamodel-based approach observations 50

5.1 WS-Policy observations 59
5.2 UDDI QoS extension observations 60
5.3 WSPL observations 61
5.4 WS-Agreement observations 64
5.5 WSLA observations 65
5.6 SLA* observations 66
5.7 SLAng observations 66

Glossary

ATL	ATLAS Transformation Language
BPEL	Business Process Execution Language
CIM	Computation-independent model
EMF	Eclipse Modeling Framework
MDA	Model-Driven Architecture
MDE	Model-Driven Engineering
OMG	Object Management Group
PIM	Platform-independent model
PSM	Platform-specific model
QoS	Quality of Service
QoS&FT	OMG's UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms Specification
RSA	IBM's Rational Software Architect
SOA	Service-oriented architecture
SoaML	OMG's Service-oriented architecture Modeling Language
UML	Unified Modeling Language
WS-CDL	Web Services Choreography Description Language
WSDL	Web Services Description Language
XMI	XML Metadata Interchange
XML	eXtensible Markup Language

Chapter 1

Introduction

This chapter gives an introduction to the purpose of this document. It explains the motivation, the objective of the research along with the research questions, the used approach and the structure of the report.

1.1 Motivation

The software engineering domain is gradually becoming more mature but is still evolving. For instance, the relatively recent software engineering paradigms model-driven engineering (MDE) and service-oriented architectures (SOAs) have proved to be beneficial in this domain [2, 3]. Both SOA and MDE help manage and improve complex software projects in several aspects [4].

Model-driven engineering (MDE) is a development approach based on the use of models in the software construction. These models are then leading for the development process, i.e. they can be used to understand, estimate, communicate and produce code [2]. MDE focuses on domain models, which can be used in various model transformations to produce other useful models such as code. Productivity can be improved with automated transformations.

The Model Driven Architecture (MDA) is a well-known initiative proposed by the Object Management Group (OMG) for implementing a model-driven approach by providing a set of tools that manage models [5]. It proposes a separation of the development process in three different abstraction levels at which computation-independent models (CIMs), platform-independent models (PIMs) and platform-specific models (PSMs) are defined. For instance, a PIM abstracts from platform-specific details, which are considered in PSMs.

MDE is more comprehensive than MDA because it considers multiple modeling dimensions in addition to the platform-independent/platform-specific dimensions, such as models for different purposes and stakeholders and organizational issues [2]. MDE is used to preserve analysis, conception and development efforts, improve productivity and facilitate the migration of applications from one platform to another [2].

Service-oriented architecture (SOA) is an architectural style which allows designers to design and develop software as interoperable services. It facilitates the development of services that are modular and can be easily integrated and reused. This is beneficial because in software development the focus changes from developing applications from scratch, to developing services as the reusable building blocks to build applications. Consumers can access SOA services in a standardized way and without needing to understand how the service is implemented [6].

Because SOAs can consist of several artefacts such as services, contracts, participants, relations and quality constraints, SOAs could become complex. A good way to understand them would be to somehow model these service-oriented architectures. SOA models help explain, formalize and understand these architectures. There are several initiatives to model SOAs [7, 8, 5].

Research work and papers shows that applying MDE to the development of SOAs is beneficial [9, 3, 10, 11]. A potential benefit is that the SOA application is modeled at different abstraction levels, making the SOA application more understandable for all SOA stakeholders. MDE can be used to fulfill the separation of technological from technology-independent models [12]. This can be used to increase the speed of development of SOAs, thus reducing software costs. The migration to other (possibly newer) technologies should cost less effort when following the MDE approach [2].

1.2 Problem statement

A complete service life-cycle covers all the activities SOA stakeholders have to deal with, and therefore covers all essential phases, starting from service design to service management. In order to improve the development of SOA applications with MDE, we have to apply MDE to activities of the service life-cycle.

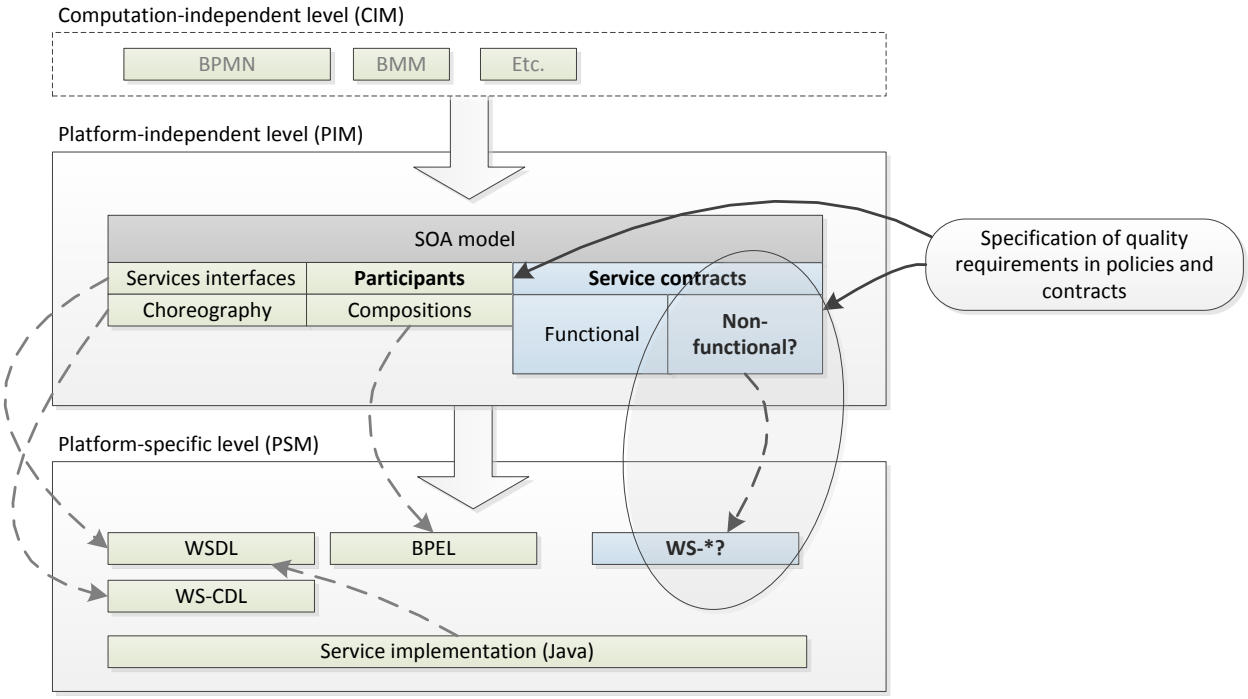


Figure 1.1: Overview SOA application modeling process

The specification of quality requirements in, for instance, policies and contracts, is one criterion for SOA modeling languages that we identified from the service-lifecycle analysis [13]. There are many quality requirements possible with respect to SOAs [14]. A SOA modeling lan-

guage should allow designers to model quality aspects (also known as quality attributes), so that requirements on these aspects can be specified in the models [15]. Examples of quality aspects are reliability, performance, availability, security, cost and integrity [14]. The term quality of service (QoS) is often used to refer to the collection of quality requirements for a service. A policy is associated with the point of view of an individual participant [16]. For instance, a service provider's policy that defines the quality aspects which are important for this provider, may be the basis for a service provider/consumer's contract when a service consumer agrees to the provider's policy.

Figure 1.1 illustrates the transformation process starting from CIM to PSM models for SOA. Section 1.6 explains that this research mainly considers the PIM and PSM-level. A SOA modeling language is needed to model the SOA applications and services at PIM-level. Specifying quality requirements at PIM-level (i.e., they can be incorporated in the corresponding SOA models) seems to improve the understandability and completeness of SOA models and could further improve development speed [17]. In this case these quality specifications can be used in the MDE process, and, for instance, (automatically) be transformed and included in PSM models. In order to realize this, we need to sort out how to properly represent these quality constraints in SOA models.

1.3 Objectives

The objective of this research is stated as follows:

> *"Investigate how quality requirements of service models should be specified at both PIM and PSM level and how these PIM models should be transformed to PSM models, so that productivity for the activities of stakeholders in the service life-cycle is improved."*

1.4 Research questions

To attain the research objective the following main research question is defined:

> *"How to apply MDE technologies to support the specification of QoS requirements in SOA at both PIM-level and PSM-level?"*

Below, several research sub-questions have been defined that help to answer the main research question.

RQ1 *Which activities of stakeholders in the service life-cycle involve the modeling of quality requirements and can be facilitated by applying MDE?*

- **RQ1.1** *How detailed must the quality requirements be modeled in this activity?*
- **RQ1.2** *How can the productivity of this activity be improved?*

RQ2 *Which SOA modeling languages are available and are most suitable to model the PIM-level service models?*

- **RQ2.1** *Which platform-independent SOA modeling language is most promising and suitable for further research?*

RQ2.2 *How does this language support the modeling of quality requirements?*

RQ3 *Which platform-specific SOA techniques are available and are able to represent the quality requirements of the service models?*

RQ4 *How can we achieve transformations from the quality requirements defined in the platform-independent service models to the platform-specific models we identified?*

RQ4.1 *Can existing metamodels be used for the PIM to PSM transformation, and how can we map the PIM metamodel onto the PSM metamodel?*

RQ5 *Is productivity of the stakeholder activities improved by using model-driven engineering for the transformation of quality attributes in PIM service models to PSM techniques?*

1.5 Approach

Figure 1.2 illustrates our approach. The upper left corner of each step shows a possible relation to the research questions of a step.

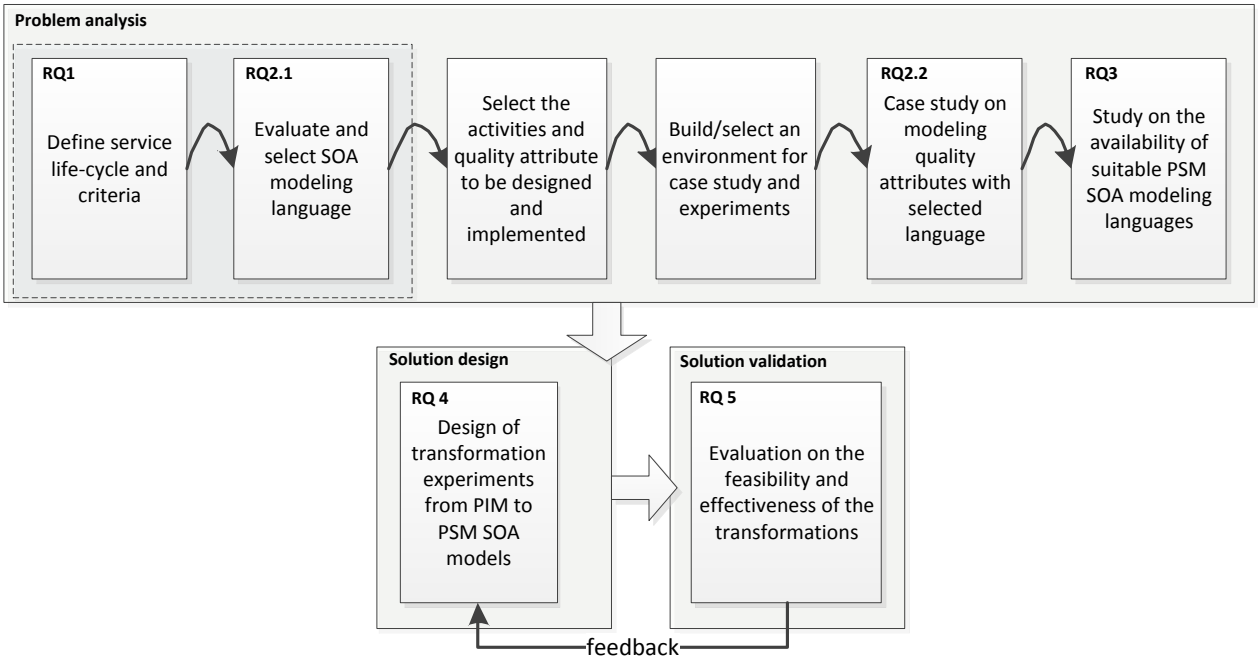


Figure 1.2: Research approach

To fulfill the objectives, the following steps have been performed:

1. A literature study, based on the service life-cycle, to explore and select PIM-level SOA modeling languages for this research. This step (shown in Figure 1.2 in the dashed area) has already been partly performed in prior research [13], and is summarized and further explained in this document.

2. A selection of actual activities and corresponding quality requirements of certain stakeholders that are used in the research to define the scope of the project.
3. The development and/or selection of tooling and environment in which the SOA modeling languages are evaluated and in which the transformation experiments are performed.
4. A case study in which the selected PIM-level language has been evaluated to test its capabilities to model quality requirements at platform-independent level.
5. A further study to explore suitable PSM modeling languages which are able to represent the quality requirements at the platform-specific level.
6. Design and execution of the PIM to PSM transformation of SOA models including the quality requirements.
7. A test of hypothesis to evaluate the success of the transformation experiments. In this step we check whether the transformations have improved the speed of development and correctness of SOA applications.

1.6 Scope of the research

A design always should have a purpose, for instance, generating an implementation, refining another or analysis of system properties. In this research, models are considered in which quality attributes should be specified by stakeholders. The relation and transformation of these models, from PIM-level to PSM-level, is also considered in this research.

This means that this document focuses on (technical) service models rather than business models which can be considered at the CIM-level. This work is about technical (ICT) services and the corresponding SOA applications that are composed with these services.

1.7 Structure of report

The structure of the thesis resembles the research questions we have answered during our research. The remainder of this thesis is structured as follows:

Chapter 2 introduces the service life-cycle, and identifies the activities of stakeholders which involve modeling of quality attributes. Eventually, the chapter lists some typical optimizable activities which are used as guiding line in this research.

Chapter 3 introduces some relevant MDE concepts and explains why SoaML has been selected as the most suitable PIM-level SOA modeling language for further research. The chapter also gives a brief introduction to the language and shows which environment and tooling is used in this research.

Chapter 4 evaluates SoaML on its capabilities to model quality of service aspects. The chapter shows which techniques can be used to include quality attributes in SoaML models, and explains why we have selected a QoS modeling language as our choice.

Chapter 5 studies the available PSM-level models often used to support the quality aspects as defined in the PIM-level SoaML models. The chapter considers policy and service contract techniques and explains the final selection so that these languages can be used as languages for the

target models of the transformations.

Chapter 6 explains the transformation environment and how QoS-aware SoaML models can be transformed to PSM models including the quality attributes. The chapter also shows results of the transformation.

Chapter 7 gives an overview of related work, summarizes our findings, and suggests topics for future research.

Chapter 2

Service models

This chapter defines and explains the service life-cycle we considered in this research. It also explains which activities of the different stakeholders involve the modeling of quality requirements.

2.1 Service life-cycle

Figure 2.1 introduces the terminology used in this report. We use the term service models to denote all models of a service that describe the functionality, interface, terms, conditions and behavior of the service, i.e., the complete specification of a service. The term service description is used to denote the models needed for publishing, finding and using a service, i.e., the models needed in the service registry.

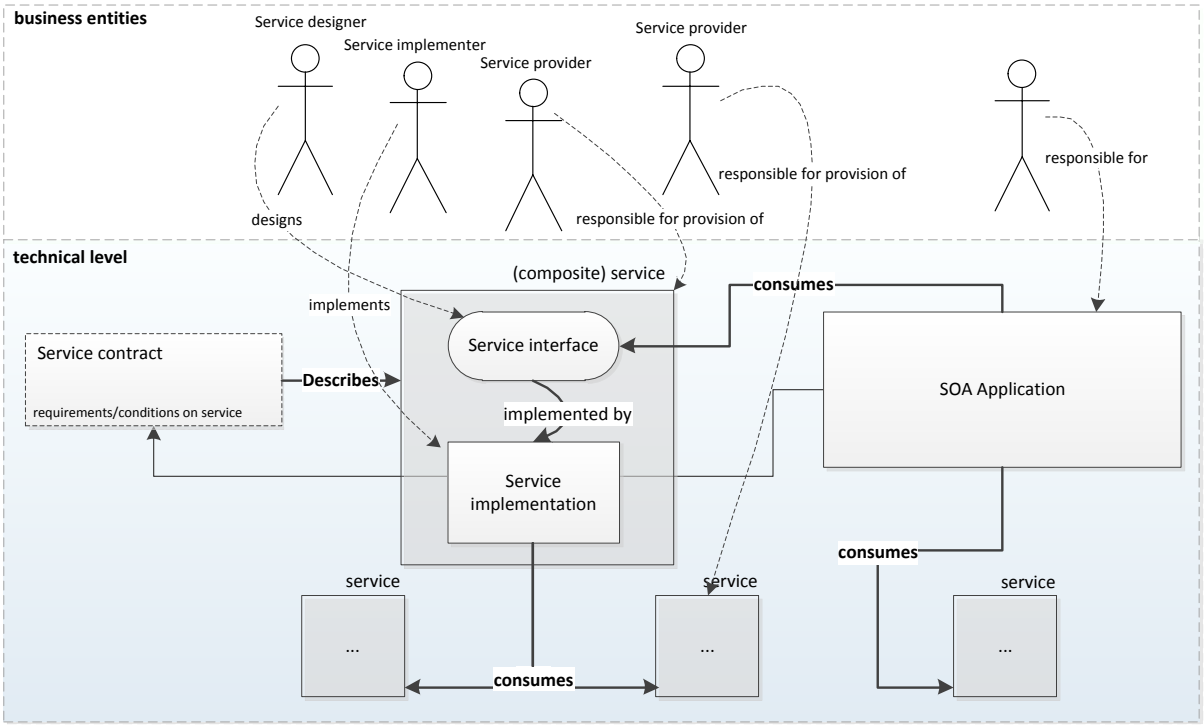


Figure 2.1: Terminology SOA application and service composition

A SOA application uses services and is developed for the end-user (the potential customer). The end-user may be a company if an employee of a company uses SOA to improve the company's business. A SOA application could involve multiple services and stakeholders. Each service has a service interface, which is publicly available for others, and an implementation.

A service that is formed from other services (component services) is called a composite service. The concept of building services from other services is called service composition. The term service contract is used to describe an agreement on the requirements, criteria and/or specification of the required interaction of a service. Initially we call this specification a proposed service contract or contract template. When involved parties agree to terms of the contract, we call it an instantiated (or agreed) service contract.

It is important to acknowledge the distinction between the business level and the technical level as shown in Figure 2.1. When we talk about service consumers and providers we mean the stakeholders at business level, which are responsible for the entities in the technical level. For instance, the employee responsible for the SOA application in Figure 2.1 manages a SOA application that consumes services. This employee plays the role of a service consumer, but the actual service consumption is in the application at the technical level. When we talk about the consumed/provided service or SOA application we mean the entities at the technical level.

A service life-cycle describes through which phases a service goes from design to operation. Some commonly identified phases of a service life-cycle [18, 19, 20] are listed below, where the stakeholders' points of view are not considered yet. Figure 2.2 shows the phases we have identified from the literature.

The black arrows show the sequence of phases during the development and usage of a service. The grey dashed arrows show that certain phases are related to each other, for instance, the SOA application uses services in operation.

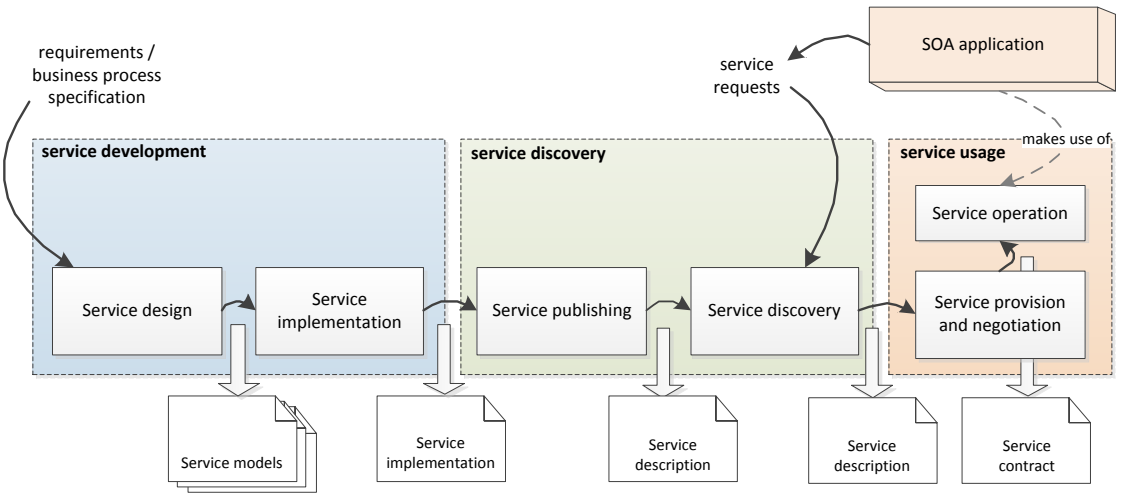


Figure 2.2: General view service of the life-cycle

Figure 2.2 further shows the corresponding results of each phase. For instance, when the service design phase ends, service models should have been produced, and a service description is needed to publish and find a service.

2.2 Stakeholders

In this section we identify the stakeholders according to the service life-cycle depicted in Figure 2.2. In principle, each stakeholder is responsible for part of the service life-cycle. Stakeholders are either involved or participate in the life-cycle activities. The following list shows the different types of stakeholders which are explained in the next sections. The composition designer subroles are explained later because separate scenarios are used as explained in the approach (see Section 1.5.

- *Providing services*
  - Service developer
    - Service designer
    - Service implementer
    - Composition designer
  - Service provider
- *Consuming services*
  - Service consumer
- *Designing and providing SOA application*
  - Application provider
    - Composition designer
- *Providing and maintaining service registry*
  - Service broker

For stakeholders it is important to know how to properly apply MDE to their activities in building, maintaining and using services and SOA applications. Different stakeholders with different points of view need different models for the activities of their part of the service life-cycle. Because the development and maintenance of a SOA application involves many parties, models of a certain stakeholder must be understood by other participants, stakeholders and maybe even other companies.

To achieve this, MDE should be applied to those models that can be transformed to other models, and which can then be understood by other stakeholders. It is also important to know which phases of the development can be facilitated by model transformations, such as service design to service implementation for the service developer. In principle, modeling and model transformations should be used in the life-cycle of services and in the SOA application when this is beneficial for stakeholders. Examples are the transformation of policies to service contracts and instantiating the service contract for provider and consumer.

2.3 Service developer

Figure 2.3 shows the phases that are relevant for the service developer. We distinguish the (sub)roles service designer, composition designer and service implementer since these activities can be performed by different people.

In the service design phase, a service designer needs to specify the service [18] according to the requirements. A service designer must be able to specify how a service works and how it can be used by others. In order to do so the designer needs a SOA modeling language that allows the specification of the service's functionality, interfaces, interaction protocols, messages and other

aspects. The designer also needs corresponding tools which facilitate the modeling of services. When the design phase ends the service models must have been defined.

When a service requires bi-directional interaction with the consumer, i.e. the interaction is more advanced than a request-response scenario, this should be modeled by the service designer. When such bi-directional interaction is defined, and possibly also other criteria, a service contract is used [11] which refers to the required interaction. In that case all parties should agree on the contract before the service can be used. The actual form of the service contract is not an activity for the service developer but for the service provider.

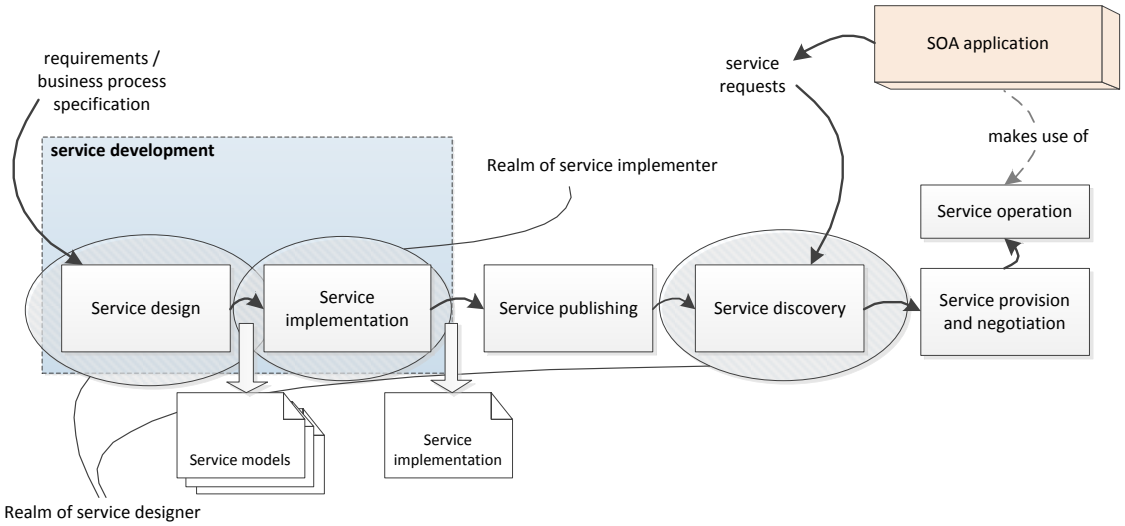


Figure 2.3: Service life-cycle service developer

During the design of the service, the designer may need service discovery to seek for existing services. If certain services can be reused, service composition is used to build the new service instead of building the new service from scratch (see Section 2.9). When no services are reused, the service implementer implements the service according to the service designs by, for instance, programming code. In this life-cycle we assume that testing is part of the design and implementing activities. At the end of the testing and implementation phase the service is implemented.

The service implementer may apply MDE in the development process. Especially the transition between design and implementation (and tests) can profit from MDE. In this case, the models from the design phase are transformed into more concrete models in the implementation phase. This could lead to, for instance, code generation of the implementation and tests [12]. Languages and tools to implement the service in the service implementation phase are necessary [18]. To make use of MDE, the tool could support transformations from CIM and PIM to PSM models, for instance, facilitating the transition from design to implementation with code generation.

The service designer needs to be able to:

- [mandatory task] Discover services.

- [mandatory task] Specify the service interface(s) and other service models.
    - [optional task] Specify the required interaction with other participants (included in service contract template).

The service implementer needs to be able to:

- [mandatory task] Implement services according to service design models.

2.4 Service provider

When services are published, potential consumers can discover services via the service registry by looking through the service descriptions. When a suitable service is found, a contract has to define how well services are delivered in terms of costs, availability, performance, etc., by the service provider [18]. When all involved parties agree on the contract, the service can be provided according to the contract and used in a SOA application.

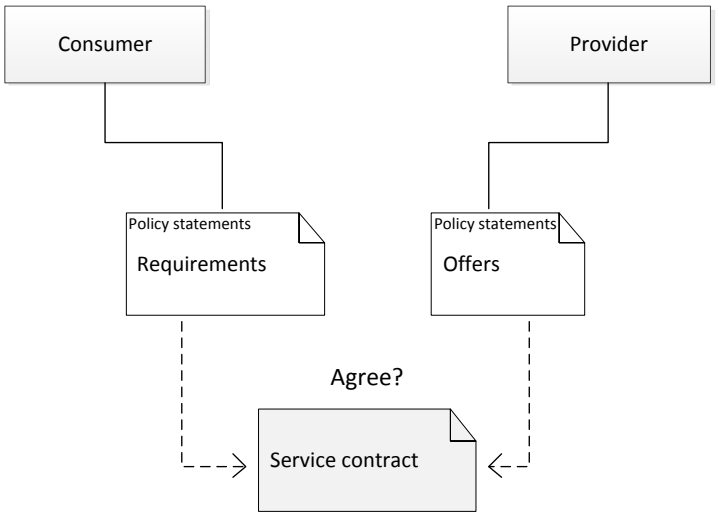


Figure 2.4: General service negotiation view

In some cases, before the service is used the consumer and provider need to negotiate in order to agree about the terms of service provisioning. Usually services are only accessible by authorized users. Figure 2.4 shows a general view of a negotiation. A consumer has certain requirements on the service to be consumed, and a provider has certain offers with respect to the provided service. These offers and requirements can relate to many aspects, such as security, privacy, availability and functionality. We denote these requirements/offers as the policy of that provider.

A service provider needs to able to define a policy (which could become a contract) describing its offers and capabilities. This contract might refer to a specification of the required interaction (choreography) defined by a service developer. It should be possible to model these concepts in such a way that it can be realized and instantiated in the negotiation phase. MDE

can help to transform a policy of a provider to a contract template which can be accepted by a service consumer. Another example is the realization of an agreed service contract (enforcement) on a certain platform. In all these cases modeling and model transformations can help optimize the process. Preferably these policies can already be modeled at PIM-level and then be included in the transformation to PSM-level models to use the advantages of MDE.

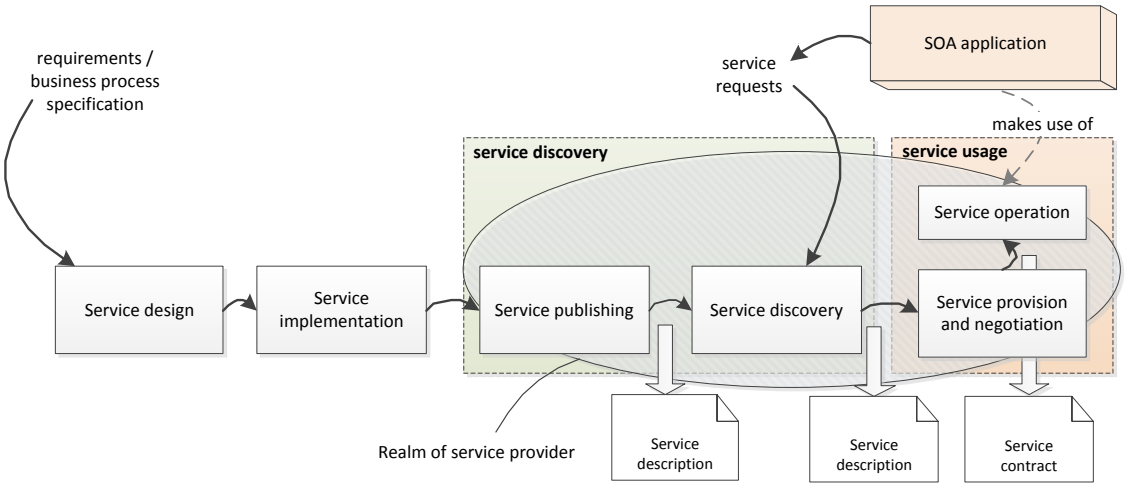


Figure 2.5: Service life-cycle service provider

Figure 2.5 shows the phases in which the service provider plays a role. The service provider publishes the service to the service registry in order to allow potential consumers to find the service. A provider should be able to define constraints/conditions on the use of the provided service [16], which should be included in a service template. In case some party wants to use the published service and agrees with the contract template, a contract is established. The service provider is then expected to provide the service in the service provision phase according to the agreed contract.

When the service is provided, monitoring is needed to check if the agreements in the contract are met. This is captured in the service operation phase. For instance, if the contract defines that the service should have an availability of 99%, then service monitoring is used to measure whether the availability of the service is sufficient.

Furthermore, changes to the service should have as less as possible impact to clients. This means that any runtime changes that occur should be handled [18], e.g., in an agreement that could be included in the service contract.

The service provider needs to be able to:

- [mandatory task] Publish services.
- [mandatory task] Provide service description for service discovery (e.g. for a register).
    - [optional task] Specify a service contract template with his offers, and/or express offers with a policy.
    - [optional task] Participate in the contract negotiation.

- [optional task] Monitor services.
- [mandatory task] Maintain services.

2.5 Service consumer

Entities that use services are service consumers, possibly after finding the services in a service registry. A service consumer often uses a service for the realization of a SOA application (as depicted in Figure 2.1). This application also has its life-cycle phases from design to maintenance. These phases do not directly involve the service life-cycle, but modeling an SOA application can be helpful for the common understanding among all stakeholders. An IT department of a company which is responsible for building the SOA application can play the role a service consumer [11, 15, 21]. However, it is also possible that there are different service consumers involved in the SOA application, all realizing a part of the SOA application [11]. A distinction is made between the role of a service consumer and application provider (where the latter is the role of designing and maintaining the SOA application, explained in the next section).

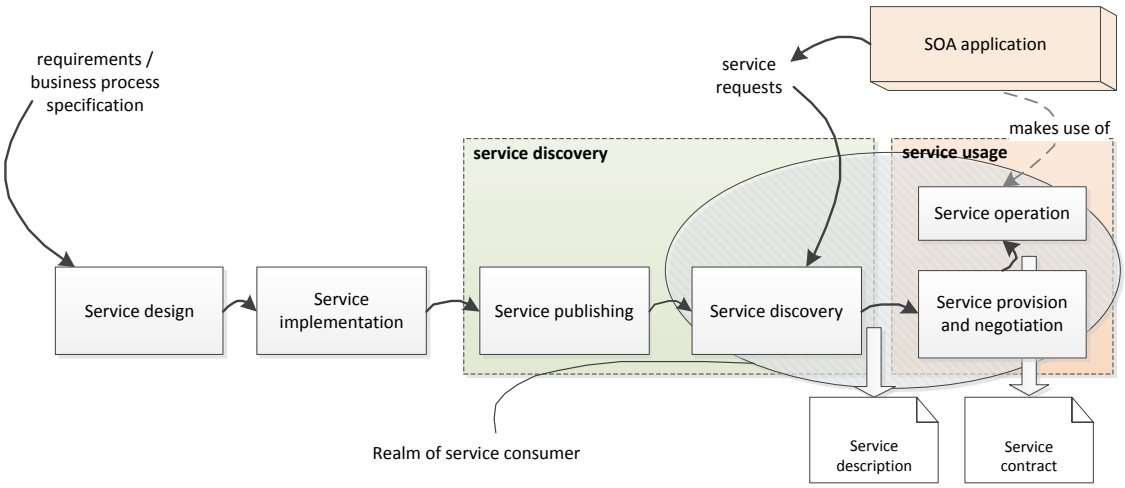


Figure 2.6: Service life-cycle service consumer

In the service consumer role, service discovery, usage, and service composition are important. The quality of the consumed services is also important for consumers. A consumer should check the service description of a provided service to determine if its offers are sufficient. If the offers are not sufficient a consumer might start a negotiation with the provider. A consumer can also express his requirements with a policy, so that providers or possibly a broker can see them.

Figure 2.6 shows that the consumer is involved in service discovery, composition, the negotiation and monitoring in the operation. A consumer needs service discovery to find suitable services to fulfil its requirements. When a potential service is found in the registry, a negotiation between provider and consumer takes place to agree on the usage of the service. In case the service is selected for usage, the result of this negotiation is an instantiated service contract.

Furthermore the consumer is important in the service design as potential user of the new service.

The service consumer needs to be able to:

- [mandatory task] Discover services.
- [mandatory task] Participate in the contract negotiation, e.g., by expressing requirements with a policy.
- [optional task] Agree on the service contract and use service.
- [optional task] Monitor the consumed service.

2.6 Application provider

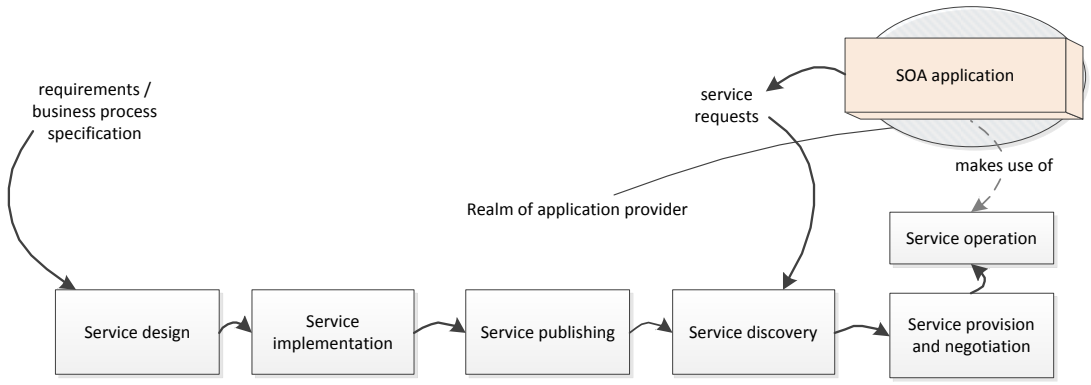


Figure 2.7: Service life-cycle activities application provider

We distinguish the role of a application provider from the role of a service consumer because the activities can be performed by different people. The application provider is responsible for the SOA application, i.e., the application for the end-user that consists of several cooperating services. In many cases the application provider will also play the role of a service consumer for each of the services in the SOA application.

Modeling and maintaining an SOA application are the activities of the application provider. This stakeholder needs a language to model the SOA application. For instance, the application provider might need a language to describe the application architecture (high-level architecture) if this helps improve the business process and understanding for all involved stakeholders. A model of a SOA application can include participants, service contracts and the used services.

The application provider needs to be able to:

- [mandatory task] Model an SOA application.
- [mandatory task] Maintain SOA application.

2.7 Service broker

A service broker is a role often identified in the literature which acts as an intermediary between a provider and a consumer [22]. The main role of a service broker is to maintain service

information which is contained in a service registry to support service discovery. We assume that services should be visible for consumers somehow [16, 11] because the visibility is necessary for service discovery. If services are not visible, consumers can not find and access services. Providers use the service registry to publish their services, while consumers use it to look up services [22, 15].

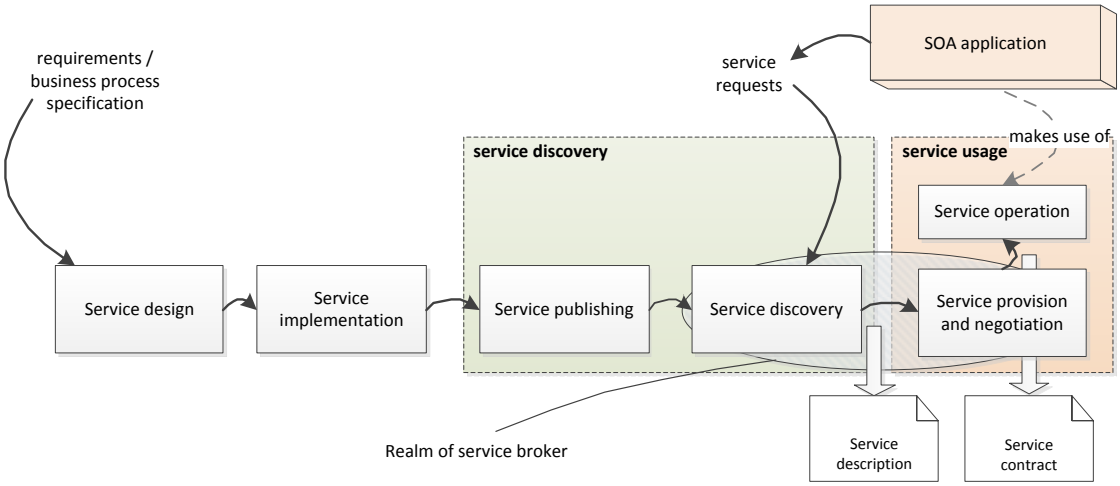


Figure 2.8: Service life-cycle service broker

The service broker is involved in the service discovery, negotiation and management phase. This stakeholder is responsible for the functioning and maintenance of a service registry. When the registry needs maintenance this should preferably effect the other stakeholders as little as possible. The service broker facilitates the service negotiation and provision phase by making service information available for consumers.
The service broker needs to be able to:

- [mandatory task] Support service discovery (with a service registry).
- [mandatory task] Maintain the service registry.

2.8 Composition designer

Service composition can also be considered in the stakeholder activities. Two stakeholders are able to use service composition in their activities.

A service designer might reuse other services to built a new service. In this case service composition is part of the design phase of the service, e.g., the new service is obtained with composition and the developer needs to learn how other services can be used in the new service. The service designer then plays the role of a composition designer. The composition designer "implements" the service as a composition. In this case the designer does not know the actual service consumer.
The service designer as composition designer needs to be able to:

- [mandatory task] Design a service as a composition at design time.

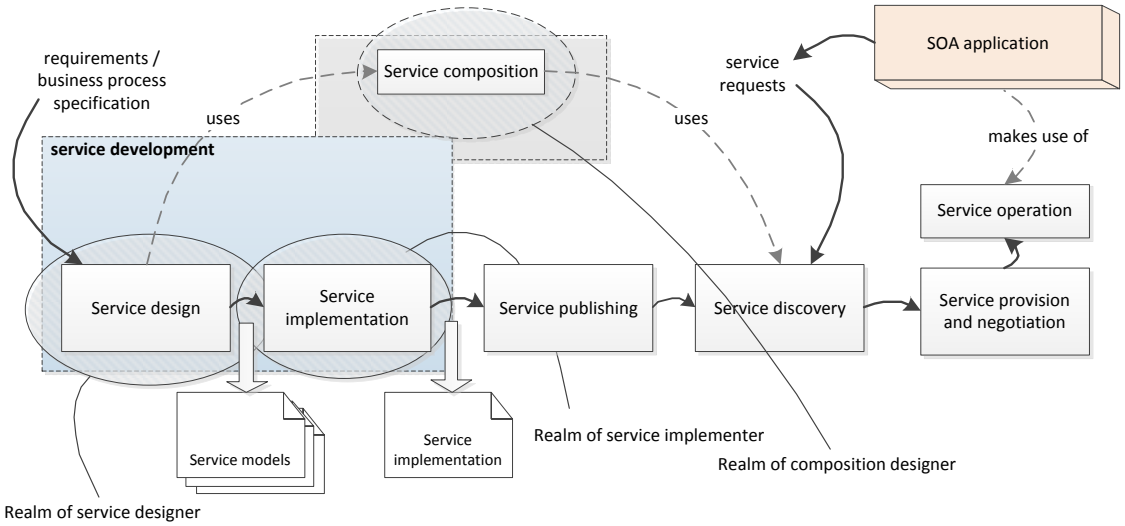


Figure 2.9: Service life-cycle service broker

The second stakeholder who might need service composition is the application provider. In this case the application provider might use other services, and thus use service composition to compose a service for its needs [20]. An application provider applies service composition to use several services in combination to fulfill a certain business goal. Since the consumer does not want to develop a new service at design time (as explained in the role of a service implementer), the consumer must be able to model this composition in the SOA application. In this case, the consumer plays the role of a service composition designer which composes new services at runtime.

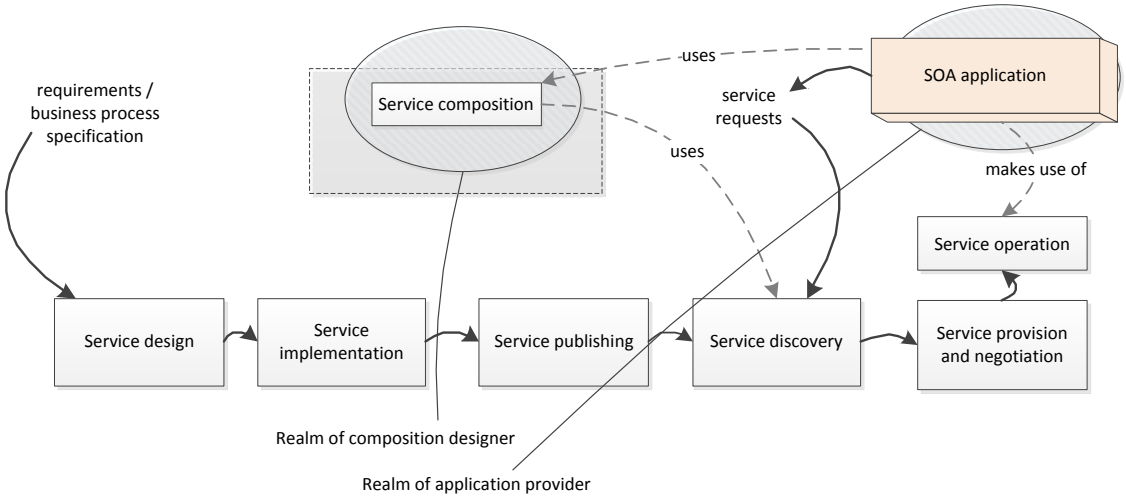


Figure 2.10: Service life-cycle service broker

The application provider as composition designer needs to be able to:

- [optional task] Design a SOA application as a composition at runtime.

2.9 Quality aspects

When modeling SOA applications and services, often the term quality of service (QoS) is used to refer to the collection of quality requirements for a service. QoS should be modeled [14] because these quality requirements can then be acknowledged by stakeholders. Different services can provide the same functionality for consumers, so QoS specifications of these services are essential in determining the most suitable services for consumers [23]. With respect to SOA QoS often covers security, performance, availability, and privacy [24]. The importance of a certain quality aspect depends on the context of the project in which the SOA is designed. A SOA modeling language should be able to model quality aspects, so that requirements on these aspects can be specified in the models [15]. In order to properly model quality requirements it should be determined at which abstraction levels these requirement are relevant, and how we can describe these requirements. Some concepts related to QoS in SOA are briefly discussed below.

Policies

Service developers, providers and consumers might need to specify the QoS of services being supported. A concept often used to specify constraints on models is a policy [11, 16]. A policy represents some constraint or condition on the use, deployment or description of an owned entity as defined by a participant. Policies allow the definition of constraints on required or supported QoS. Service policies potentially apply to many aspects of SOA, such as, for example, security, privacy and manageability [16]. For example, the service consumer may require secure interactions. A service provider might specify that his service has a minimum availability of 95%. Therefore we must investigate how to describe constraints and policies in and on the models with SOA modeling languages.

Policies relate to, but are different from, service contracts. A service contract is an agreement between two or more participants, while a policy is associated with the point of view of an individual participant [16]. For instance, a service provider's policy may be fulfilled by a service contract between them when a service consumer agrees to the provider's policy. Below we discuss some concepts related to policies.

Service negotiation

During the negotiation phase, the consumer and provider negotiate on the usage of the service. When both participants agree, the agreement is captured in the service contract. In order to enforce the contract, service monitoring is needed to check whether the agreement is violated or respected.

Figure 2.11 visualizes how policies and contracts relate to each other. The proposed service contract of the service provider is shown on the left side. This includes a description of the service with information about offers, service interface and possibly also the choreography. The policy of the provider contains a QoS offer and other possible terms and conditions. The service

consumer also has its constraints, which has to be fulfilled by the final service contract. The service contract reflects the agreements on the usage of the service.

Service level agreement (SLA) is the term often used in literature to denote a template for service contracts [15]. An instantiated SLA can represent the agreement on the policies as depicted in Figure 2.11.

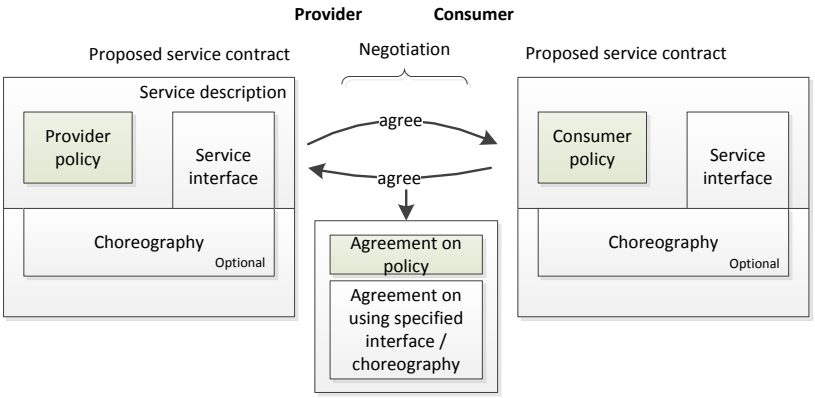


Figure 2.11: Service contract instantiation with policies

An SLA describes the intended boundaries for a service, particularly with regard to non-functional properties. SLA's are used when a certain level of verifiable quality is required. The key to defining SLAs is to provide enough information or verifiable metrics for a service consumer to preselect services based on the desired level of quality [25]. Service providers instrument their services in such a way that measurements are collected and then compared to the metrics specified in the SLA. A service consumer needs to agree with the SLA before the service can be consumed. An SLA is part of the service description, and could be instantiated in the service contract [15].

Aggregated QoS

When taking into account composite services we also have to consider how the QoS provided by composite services is defined, also known as the aggregated QoS. These are based on and influenced by the QoS of the component services [14]. Low quality of a component service may cause all services based on this service to degrade in quality [26]. Calculating the aggregated QoS is helpful for the provider of a composite service to define policies.

Common quality aspects in SOA applications

There are several papers providing overviews of common quality aspects in SOA applications. Some quality aspects which are often mentioned are availability, reliability, performance, usability [27, 23, 26]. The work of [23] is interesting because it considers multiple stakeholders as it is done in this work. We are interested in common quality requirements for stakeholders which should preferably be modeled during design-time. The exact form of the QoS requirements depends on the SOA application being developed. Some examples are listed below:

Availability A common quality aspect is availability. Some example metric is the availability of the service itself: availability = uptime / (uptime - downtime) [23]. A certain QoS requirement for a consumer could be that a service to be consumed should have an availability of at least 99%, because otherwise the SOA application does not function properly.

Performance Another aspect for the consumer could be performance. One example metric is the service response time (SRT) which can be calculated as: time when service consumer finishes sending request to the service - time when service consumer starts receiving response from the service.

Reliability A well-known example of a reliability requirement for the consumer could be that the mean time between failure (MTBF) should be lower than a certain limit. The MTBF can be calculated as: summations of time between failures / total number of failures [23].

Other example aspects Other examples of quality requirements for a provider could be the completeness of the interface description, the maximum number of the web service requests served in a given period of time (throughput) and security.

2.10 Optimizable activities

The previous sections have shown which activities and models are possible for service stakeholders, and why it is necessary to model quality attributes. This section lists some typical activities that require the modeling of quality attributes and which could be optimized with MDE. The following activities are used in this research to show that modeling quality attributes at PIM-level and corresponding transformations to more detailed models are beneficial for stakeholders. This section defines the hypothesis for our transformations. We take a closer look at these activities to determine at which level and how quality attributes must be specified in the corresponding models.

Policy specifications A consumer has certain quality requirements on the service to be consumed. For instance, one obvious requirement would be that the service has a certain level of availability which is sufficient for the consumer. The provider also has to specify a policy on quality aspects such as availability, throughput, security and interoperability [23]. It would be beneficial to already model these policy specifications at PIM-level because these requirements, or at least the type and bounds of these requirements, are probably already known at design time of the SOA application. Otherwise these requirements should be added later on (i.e., during the transformation to PSM-level models), which costs extra time. It would also be better to have as much as possible already specified in the PIM-level model because a PIM-level model should gather all the information needed to describe the behavior of the system in a platform independent way [28].

Figure 2.12 gives an overview of the situation. QoS requirements expressed in the policies of the consumer and provider can already be (partly) modeled at PIM-level. When generating PSM-level models (and ultimately runtime models, which we consider as detailed PSM-level models), the QoS requirements are already available.

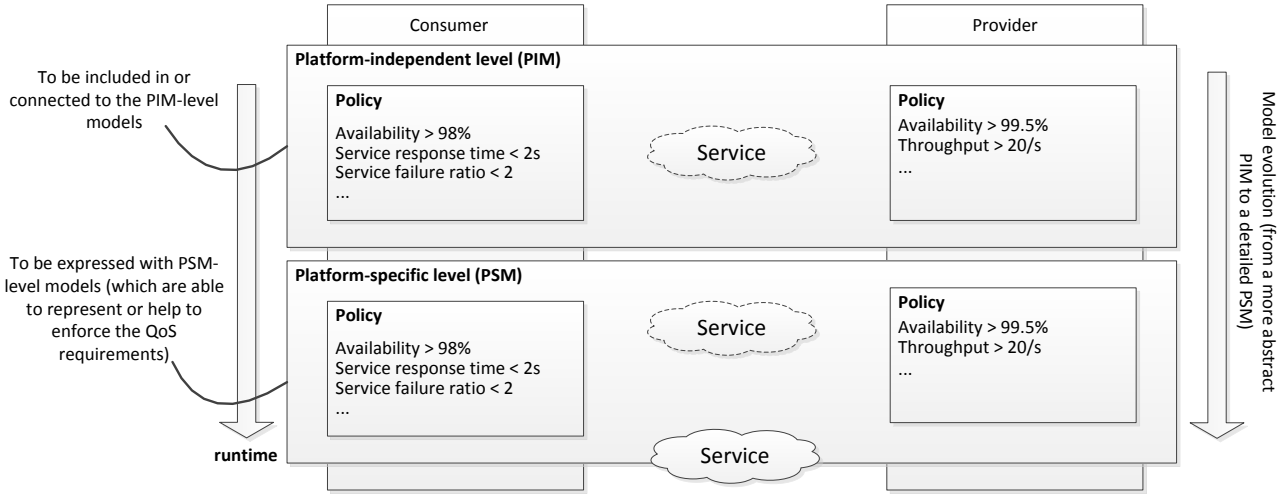


Figure 2.12: QoS specification with policies

Service contracts When modeling an SOA application, service contracts can already be modeled at design time to define the agreements on the usage of services [11]. The service contracts then form (a part of) the SOA application, and the modeling of these contracts is therefore part of the activities of the application provider. In order to fully model a service contract, agreements on quality attributes must be included in the service contract (as explained in Section 2.9). Both the provider and consumer are involved in the instantiation of this contract, because the service contract is based on the policies of these stakeholders. When these contracts are defined at PIM-level it would be effective to (automatically) translate these contract models to PSM-level models which are able to express or even instantiate the contracts on that specific platform.

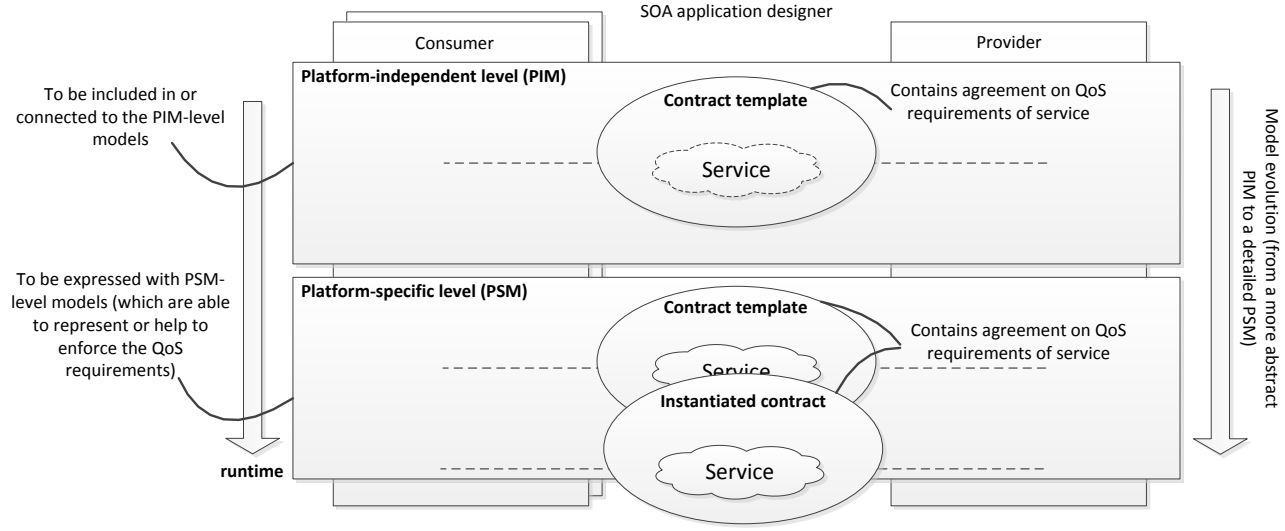


Figure 2.13: Contract definition

Figure 2.13 shows how an instantiated contract can be achieved based on a PIM-level contract, which preferably, already includes as much as possible QoS information.

The previous two activities can both be optimized when it is possible to already model quality attributes at PIM-level. Some advantages are that the transition from PIM-level to PSM-level models can be done faster and that the QoS requirements are already specified at PIM-level. This makes it possible to specify the design of the SOA application and services more complete and rigorous [17].

Chapter 3

SOA modeling

The chapter introduces some relevant MDE concepts and explains why the SOA Modeling Language (SoaML) is selected as PIM-level modeling language for further evaluation. It further gives a brief introduction the SoaML.

3.1 Relevant MDE principles for SOA

MDA is by far the most well-known MDE initiative and a lot literature is devoted to it. In this report the MDA concepts are also used regularly and therefore this section briefly introduces MDA's most important concepts. MDA defines the following models:

1. Computation-independent models (CIM) are a view of a system from the computation-independent viewpoint. The computation-independent viewpoint focuses on the environment and the requirements of the system; the details of the structure and processing of the system are hidden or are yet undetermined. A CIM does not show details of the structure of systems. A CIM is sometimes called a domain model, and a vocabulary that is familiar to the practitioners of the domain in question is used in its specification [29].
2. Platform-independent models (PIM) are a view of a system from the platform independent viewpoint. The platform independent viewpoint focuses on the operation of a system while hiding the details necessary for a particular platform. A platform independent view shows that part of the complete specification that does not change from one platform to another. A PIM exhibits a specified degree of platform-independence so as to be suitable for use with a number of different platforms of similar type [29].
3. Platform-specific models (PSM) are a view of a system from the platform-specific viewpoint. A PSM combines the specifications in the PIM with the details that specify how that system uses a particular type of platform [29].

In principle an MDA process starts with defining CIM or domain models, then these models can be transformed to PIM and PSM models. These transformation are called vertical transformations. A direct vertical transformation is not always possible because the gaps between the models that are too big to bridge in a single transformation. In these cases, additional transformations are used to, for instance, transform a certain abstract PIM model to a more detailed PIM. A PIM-model has a certain level of platform-independence which has to be identified. This could, for instance, be done using the concept of abstract platforms [30] supporting the platform-specific realization of a model.

When applied to the development of SOA applications, the CIM-level could describe business models which include goals, business rules, business processes and business services. This could be done with for instance the Business Motivation Model (BMM) and Business Process Modeling Notation (BPMN).

The PIM-level describes SOA models which are independent of the execution platform or technology being used. It includes models of service interfaces, service contracts, service enactments, participants, etc [11]. Models at this abstraction level are considered as software specification models.

The PSM-level describes platform models as executable artefacts. These models can be considered as software realization models. If the SOA application is implemented with, for instance, web services, service interfaces can be represented using the Web Service Definition Language (WSDL) [31]. Another example is the Business Process Execution Language (BPEL) for specifying business process behavior based on web services [32].

Metamodels

For MDE, metamodels are an important concept. Every model in a certain modeling language conforms to that language's metamodel. A metamodel can be considered as an explicit description (constructs and rules) of how a model can be built.

Model transformations can be used for many MDE and SOA purposes, such as, e.g., transforming PIM to PSM models, generating specific models for a certain stakeholder's viewpoint, etc. The most well-known metamodel-based model transformation languages are ATLAS Transformation Language (ATL) [33] and Query/View/Transformation (QVT) [34].

In order to apply MDE (and particularly MDA) to the development of SOAs, it is important to have a concrete metamodel for the SOA application to be modeled. If such a metamodel is not available it must be defined. This metamodel mechanism is needed to unambiguously define modeling languages, so that a transformation tool can then read, write, and understand the models. Transformations are defined as relations between metamodel elements the source and target models. [35].

3.2 PIM-level language selection

Several SOA reference models, reference architectures, maturity models, ontologies, modeling languages, and governance specifications have been released. Most of them come from open standards work fostered by OASIS (Organization for the Advancement of Structured Information Standards), The Open Group and OMG (Object Management Group). Because an abundance of specifications and standards have emerged, Kreger et al. [36] wrote an overview document that explains and positions these SOA architectural standards. Next to the open standards work, several SOA modeling initiatives come from the research community.

Role of UML

Software systems are often modeled with the general-purpose Unified Modeling Language (UML). OMG has fostered the UML standardization since in 1997. Over the past few years it has gone through several revisions up to UML 2.4, including notation techniques to model software-

intensive systems. It offers various diagrams to model the structure and behavior of software systems.

Although UML originally was developed for object-oriented system modeling, it can easily be extended to support modeling of, for instance, user interface flows, business activities, and data schemas [3]. Recent versions allow designers to describe many aspects of SOA applications [11]. However, [3] and [37] also emphasize the need to able to model a service component, which cannot be properly modeled with UML. Directly applying UML concepts for modeling SOA, although it can be regarded as a good starting point, is not an entirely convenient approach [37]. This means that with standard UML it is not possible to properly model the SOA concepts we have mentioned in our criteria [13].

Metamodel and UML profiles

UML has a built-in lightweight extension mechanism called UML profiles. With this mechanism it is possible to extend UML to match the needs of a certain domain. A set of stereotypes and constraints can be created and grouped into a profile. These stereotypes can be applied to mark up a model for a specific platform or domain.

These UML profiles are a way to implement metamodels represented in these SOA modeling specifications. Several authors defined a UML profile for their approach, such as [38], [10] and [11].

The UML profile is appropriate when the objective is to model services and SOA applications using already existing UML editors. The use of stereotypes and tagged values in these profiles preserve the UML semantics and do not create new languages.

For the transformations the metamodeling approach seems to be more suitable. In the first place, a metamodel is more convenient than UML profiles in transformation languages as explained in [39] and [33]. Furthermore a metamodel can be used to extend the modeling language so that it can be used in different domains. Users can reuse metamodel implementations and extend them to create their own domain specific languages [12]. This is beneficial for possible future work.

Overview SOA modeling approaches

Table 3.1 shows a summary of several SOA modeling approaches that have been evaluated in [13]. It gives an overview of several prominent and relevant SOA modeling approaches, documents and specifications. A short description and purpose of each approach is given.

Table 3.1: SOA modeling approaches

Name	Description	Purpose
OASIS Reference Model (SOA RM) (2006) [16]	The SOA RM is intended to capture the essence of SOA, as well as provide a vocabulary and common understanding of SOA [36]. It is written at a high abstraction level and it supports much of the SOA modeling criteria as defined in [13].	Explaining SOA core concepts *Used to understand SOA concepts, no modeling language.*

Open Group SOA Ontology (2010) [40]	This ontology extends, refines, and formalizes some of the core concepts of the OASIS Reference Model. It is used for understanding the core SOA concepts and facilitates a model-driven approach to SOA development [36] [40].	Explaining SOA core concepts *Used to understand SOA concepts, no modeling language.*
OASIS Reference Architecture (OASIS RA) (2011) [15]	OASIS RA is an abstract, foundation reference architecture addressing the ecosystem viewpoint for building and interacting within the SOA paradigm. It specifies three viewpoints; specifically, the Service Ecosystem viewpoint, the Realizing SOAs viewpoint, and the Owning SOAs viewpoint. Since it is an abstract and foundational reference architecture, it does not contain the level of specificity required to directly implement SOA-based systems. It does provide metamodels and architectural implications for each of the views useful in guiding other architecture work, including other reference architectures [36].	Understanding SOA from different viewpoints *Used to understand SOA concepts, less abstract than the reference model, no concrete modeling language.*
Open Group SOA Reference Architecture (2011) [21]	The Open Group SOA Reference Architecture is a layered architecture from the consumer and provider perspective with cross-cutting concerns describing these architectural building blocks and principles that support the realizations of SOA. It is used for understanding the different elements of SOA, deployment of SOA in the enterprise, basis for an industry or organizational reference architecture, implication of architectural decisions, and positioning of vendor products in SOA context [36]	Understanding SOA from different viewpoints *Used to understand SOA concepts, much aimed at business integration.*

Soa Modeling Language (SoaML) (2012) [11]	The SOA Modeling Language (SoaML) is a promising dedicated language for the modeling of SOAs. The Object Management Group SoaML Specification supports services modeling UML extensions. SoaML is used to represent SOA artifacts in UML. It supports a lot of the SOA concepts mentioned in The Open Group SOA Reference Architecture [36]. SoaML is not a methodology for developing SOAs but purely a modeling language. It is a standard proposed by OMG. It offers intuitive and complete support for modeling services in UML [11]. It keeps the main SOA modeling principles of PIM4SOA [41]. SoaML is meant to provide a rigorous definition of service-related terms and thereby form a foundation for dialog and common understanding in the service domain. It is focused on general service design and provisioning, it is methodology agnostic and is not able to cover the full service lifecycle [42].	SOA modeling language (metamodel + UML profile) *At first sight most complete language for our purposes. Satisfies most of the criteria, including all the functional aspects. They mention that service discovery, applicability, methodologies, deployment and runtime of services are out of scope.*
A Platform Independent Model for SOA (PIM4SOA) (2007) [43]	The PIM4SOA project aims to develop a metamodel for SOA. This metamodel consists of a set of essential aspects for SOA. PIM4SOA addresses four system aspects (views): processes (logical order in terms of actions, control flows and interactions between services), information (related to the messages or structures exchanged by services), services (description of services: access, operations and types) and quality of service (extra-functional qualities that can be applied to services, information and processes). The project also provides a set of transformations that link the metamodel with specific platforms (agents, web services, etc.) following the MDA approach.	SOA modeling language (metamodel) *Satisfies part of the criteria. Mainly the functional aspects but QoS is also supported. Service contracts, choreography and service discovery are not covered.*

CBDI-SAE Meta Model (2011)	The CBDI-SAE Meta Model for SOA is a class model of the concepts contained in the CBDI Service Architecture & Engineering (CBDI-SAETM) Reference Framework. The authors worked on the design of SoaML [44] and mention that the SAE Meta Model complements SoaML by providing full lifecycle support (for their SAE methodology). They say that CBDI-SAE Meta Model supports metadata and policies where SoaML does not support this. They say that SoaML is enough if additional metadata is not needed. However, to support the full service life-cycle, they claim SoaML is not enough. CBDI-SAE Meta Model defines a mapping to SoaML.	SOA modeling language (metamodel + UML profile) *Commercial modeling language part of their SAE methodology. Looks complete according to our criteria, but has no service discovery and security support.*
Service-Oriented Modeling and Architecture (SOMA) (2004) [7]	SOMA is a modeling and design technique for defining and developing a service-based IT solutions proposed by IBM in 2004. It was one of the first modeling approaches for SOAs. They describe that the main first-class constructs in an SOA are services, service components, and process flows. These are at a higher level of abstraction than that of objects, classes, and components. Hence, there needs to be a higher level of modeling and design principles that deal with the first-class constructs of an SOA [7]. The SOMA approach is built on top of object-oriented analysis and design (OOAD) while it adds modeling and design techniques specific to SOA. Deliverables of SOMA can be created with, for instance, the SOMA-Modeling Environment (SOMA-ME). This is a framework based on SOMA that uses UML profiles (which extend UML 2.0) to model SOAs in a model-driven way [45]. However it says that IBM's newest version of the SOMA methodology (version 2.9) heavily uses SoaML and is tightly aligned with the SoaML tooling in the modeling products [5]. It is hard to find further information about the SOMA methodology because the development is closed. Documentation and their commercial tools are not freely available.	SOA-based lifecycle methodology serving as a service-modeling platform *Commercial modeling language part of their SOMA methodology. Moved to SoaML as main modeling language.*

Service-Oriented Modeling Framework (SOMF) (2008) [46] [8]	The SOMF framework is an agile model-driven engineering methodology that offers a modeling language and best practices that can be used during various stages of the software development life cycle [46] proposed by Micheal Bell. The SOMF framework is very broad, it aims as a holistic language to design any application, business and technological environment, either local or distributed. SOMF specifies technology-neutral services. It does not assume a web service SOA-based implementation. The technology neutral representation of services is expressed at multiple levels of abstraction: conceptual, analysis, design and architecture.	SOA-based methodology including modeling language *Commercial modeling language included in SOMF framework. Remarkably few research papers about SOMF. Uses its own different syntax to represent SOA concepts. Includes support for standard notations such as SoaML.*
Modelling of Service-Oriented Architectures with UML (2008) [10]	Lopez-Sanz et al. proposed a UML profile for modeling PIM level architecture. With this profile it is possible to model several types of services and contracts in UML using stereotypes at the PIM level. Lopez-Sanz et al. were working on a specification of PSM-level SOA architectural models for different service execution platforms (web service, CORBA, etc.).	SOA modeling language (metamodel) *Modeling language which satisfies all our functional criteria and some constraints at a basic level. Next to the paper, there is less information available about their metamodel. Defined before SoaML was released.*
UML-S (2008) [47]	Dumez et. al proposed the UML-S (UML for Services) extension. It extends the UML 2.0 class and activity diagram to support developing composite web services. In order to realize the model-driven vision they provide high-level UML-S models which can be transformed to platform-specific code [47].	SOA UML extension (specifically aimed at service composition, includes UML-profile) *This UML extension does cover some of the functional criteria and composition, but misses a lot of other aspects such as participants, discovery, definition of constraints etc.*

Some SOA modeling approaches were not further evaluated because they are outdated, not complete or irrelevant. Examples are IBM's "UML 2.0 Profile for Software Services" [48] which is deprecated (and replaced by SoaML), UML-RT, governance frameworks, maturity models and various other papers such as Zhang et. al [49] which seem to be overtaken by the SoaML standard. Also the "Web Services Architecture" from W3C [50] was not evaluated further in detail because it just focuses on characteristics of web services, and does not propose a SOA modeling approach.

When we specifically look at SOA modeling languages, the CBDI-SAE Meta Model is another interesting model. It looks very complete, but unfortunately it is not a standard and has only commercial support. However, the metamodel is freely available in various formats. They also mentioned that their metamodel will be continued (as part of their methodology) so it might be interesting to consider this work when necessary. The research work and metamodel proposed by Lopez-Sanz et al. [10] may also be worth investigating, because they support PIM-level SOA modeling that is not based on any previous modeling language or methodology.

Most of the other approaches are unfortunately commercial, limited or only focused on explaining SOA concepts.

The most promising and complete SOA modeling approach is SoaML. In the recent future it is expected that SoaML becomes more and more adopted as a standard modeling language for SOA [37] [42]. Almost every well-known SOA modeling approach/company (such as OMG, OASIS, the Open Group and Everware-CBDI) have contributed to this language, apart from the fact that they already have a modeling language themselves. The language is available since 2008 and IBM's SOMA methodology already moved to SoaML. Other methodologies such as CBDI's SAE and SOMF support mapping and/or integration with SoaML.

3.3 SoaML concepts

SoaML is a recently proposed standard which is becoming increasingly popular for the modeling of SOAs [37, 51]. The SoaML specification supports services modeling with UML extensions and is used to represent SOA artifacts in UML. SoaML is not a methodology for developing SOAs but purely a modeling language. It offers intuitive and complete support for modeling services in UML [11]. SoaML is meant to provide a rigorous definition of service-related terms and thereby form a foundation for dialog and common understanding in the service domain. It focuses on general service design and provisioning, it is methodology agnostic and is not able to cover the full service lifecycle [42, 13] because certain aspects are out of SoaML's scope.

SoaML is suitable to specify both services and SOA applications, taking into account the needs of different stakeholders. It was designed to support MDA and as such provides a baseline modeling language for the specification of services within a SOA application. The extensions to UML provide the key language constructs for specifying the structure of services. SoaML does not specify which kind of behavioral notation to use. The goal of SoaML was not to be a broad modeling language to support all aspects of SOAs, but rather to be a small core that can be extended and integrated with other modeling languages such as, e.g., BPMN for behavioral modeling [31].

SoaML supports the modeling of SOA concepts by introducing specialized diagrams (e.g., the service contract diagram) and extends other UML concepts (e.g., ports) with additional semantics. These diagrams are extensions to the UML standard diagrams. Figure 3.1 shows an

example of a SoaML diagram describing a service interface. The extensions provide the required syntax to model the SOA concepts and can be used to express the semantics of these concepts.

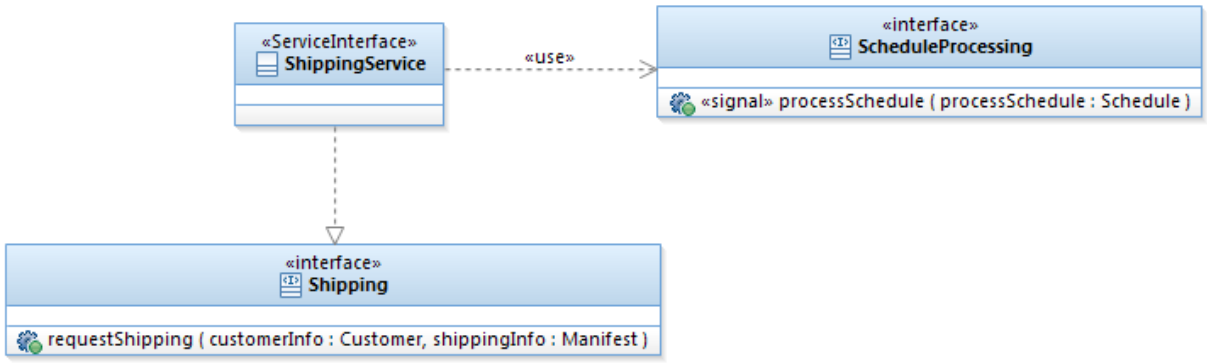


Figure 3.1: Example of a service interface defined in SoaML

SoaML is also based on loose coupling. Loosely coupled systems imply that services should be designed with little or no knowledge about particular consumers of these services. Different consumers may have a different view of what to do with a service based on what they are trying to accomplish [11]. For example, service contracts promote loose coupling in SoaML, since it is not necessary to define who, how or why parties will fulfill their obligations in a service contract. This may be modeled using SoaML in, e.g., a Participant diagram.

Furthermore SoaML supports modeling at different abstraction levels, by separating the inside and outside of several SOA concepts (such as participants). In this way, SOA concepts can be modeled more easily by different stakeholders. For instance, a SOA application stakeholder can model the overall architecture of an SOA, while a service implementer can model some service operations. Furthermore SoaML allows nesting of concepts, which further promotes modeling at different abstraction levels.

SoaML does not support service discovery. Service discovery has to be realized separately with a service registry and suitable service descriptions. The specification of policies and non-functional constraints of services are not covered either. Furthermore, the SoaML specification does not mention how we can refine SoaML models to PSM (e.g., runtime) models. Examples of stakeholder activities that need refinements of PIM models implementation towards concrete services (possibly including composition) and the enforcement of choreographies and other agreements in the contract.

SoaML can be used to model most of the mentioned criteria in [13], but some aspects are out of the scope. For instance, the SoaML specification does not address the specification of quality constraints. The SoaML specification describes that policies and quality attributes can be modeled as constraints that can be rules owned by model elements, including service ports and service participant components. The actual form of these policies and further details are out of scope for the SoaML specification [11].

In order to be able to use SoaML, stakeholders should know in which ways and how well quality constraints can be modeled in SoaML models. The SoaML specification explains it is possible to model quality aspects in SoaML, but does not prescribe any technique or methodology.

3.4 SoaML tooling

Since SoaML is a standard published in March 2012, its tool support is still limited. OMG's SoaML wiki [52] lists some of the available tool support for SoaML.

Table 3.2: SoaML tool support

Name	**Description**	**Licensing**
ModelPro	ModelPro is a general purpose MDA provisioning engine able to produce a wide variety of artifacts from models, based on the Eclipse tooling framework. It provides a SoaML cartridge which is able to produce executable web service implementations for services architectures defined in SoaML. Apparently SoaML is implemented as an UML profile [52].	Open source
Cameo SOA+ suite (NoMagic MagicDraw)	This suite provides a plugin for MagicDraw from No Magic and the ModelPro MDA tooling from ModelDriven.org. With this suite it is possible to visually model SoaML application in both MagicDraw and Eclipse (also code generation is supported in combination with ModelPro) [52, 53]	Commercial
Modelio	Commercial modeling tool with an open-source SoaML designer extension. Code generation is supported.	Commercial (partly open-source)
IBM Rational Software Architect	Commercial software architect tool supporting SoaML modeling. To be used in combination with other IBM Rational products [54]. Very complete implementation of SoaML based on its profile. It also support transformations to PSM-level models.	Commercial
SparxSystems Enterprise Architect	Commercial software architect tool supporting SoaML modeling. SoaML is supported in the Corporate, Systems Engineering, Business and Software Engineering and Ultimate editions of Enterprise Architect [52].	Commercial

SoaML Eclipse plug-in by Delgado et al. [55]	Eclipse plug-in (based on Eclipse EMF and GEF) which implements the SoaML profile, support visual modeling with a Papyrus (UML design tool) extension. SoaML models can be imported and exported as XMI files, but the tool seems to lack full SoaML support, because, for instance, service behavior cannot be modeled. [55]	Open-source (source-code not available yet)
SoaML Eclipse plug-in by Ali et al. [12]	A tool for modeling SOA using SoaML and generating OSGi Declarative Services Models from SoaML models. SoaML metamodel has been implemented as an Ecore model using the Eclipse Modeling Framework (EMF). An Eclipse plug-in that allows architects to graphically design SoaML models has been developed using the Graphical Modeling Framework (GMF) [12].	Source-code not available

Where possible, we prefer open-source tooling for our work because in this case we are not limited by licenses and possibly also get access to the source code. Furthermore we also prefer tools that use the SoaML metamodel (as explained in section 3.2) instead of the UML profile because this is more convenient for the model transformations. Most of the tools in Table 3.2 are commercial and some tools are results from research projects.

Most tools in Table 3.2 do not strictly cover SoaML as specified in the SoaML specification [11]. For instance, the supported syntax is slightly different, some functionality is missing or some constraints on the models are not implemented.

The open-source ModelPro project looks promising because [53] discusses how they used SoaML models with the MDA. A drawback is that SoaML is implemented as profile, and the documentation targets ModelPro in combination with the Cameo SOA+ suite, which is commercial.

The work of Delgado et al. [55] also seemed promising. The SoaML plugins are freely available, but it also uses UML profiles, needs further improvements, and the source code is not published yet.

3.4.1 Selection

The most appropriate choice for the modeling of the SoaML models seems to be IBM Rational Software Architect. This tool seems to be the most complete and mature SoaML tool. A drawback is its commercial support and licenses, and the use of the SoaML's UML profile.

With respect to the transformations, a suitable environment would be using EMF as done in [12]. EMF is open-source, we can use the SoaML metamodel, and generate editors to model the SOA applications. Furthermore we already have some experience with Eclipse EMF. A drawback is that we have to convert the SoaML metamodel to the Ecore format to make it suitable for EMF, and initially visual modeling will not be possible unless we implement a graphical editor, which takes a lot of time.

We have made the following decision for the environment: IBM Rational Software Architect is used to model the SOA applications and services in SoaML. With this tool we test SoaML capabilities to include QoS specification at PIM-level. We can fall back to EMF for the model transformations. In that case the SoaML models created using Rational Software Architect must be converted (or rebuilt) in EMF.

Chapter 4

QoS specifications in SoaML

This chapter identifies ways to represent QoS in SoaML. This is done using a case study and some modeling criteria. Several techniques and languages are evaluated to test how they can be used to model the QoS requirements as defined in SoaML models of the case study.

4.1 Case study

Throughout this chapter we apply the specification of quality attributes (as defined in Section 2.9) in SoaML to a simplified version of the use case explained in the tutorial of [56]. The case study consists of a company that wants to order products from a manufacturer. The ordering process involves scheduling, invoicing and shipping of the ordered products.

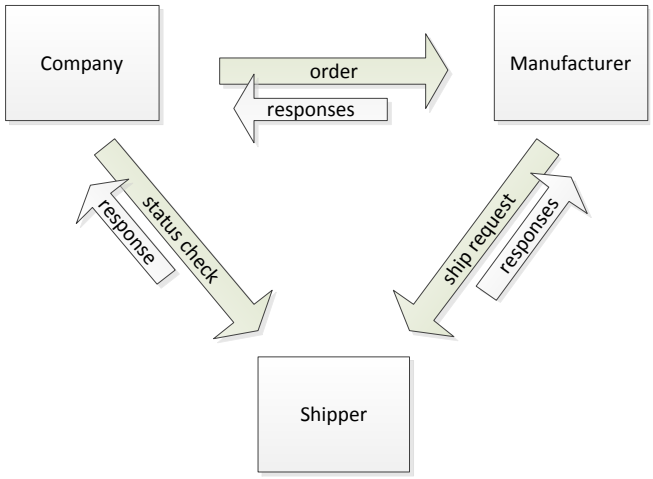


Figure 4.1: High-level view of the example

Figure 4.1 illustrates the case study. Since the business process specifications are out of the scope of this research, we assume the following services should be modeled: invoicing, scheduling and shipping.

We modeled the case study using the tool IBM Rational Software Architect (RSA). The models may look different from the SoaML models in the SoaML specification because IBM RSA uses

a slightly different syntax than the syntax shown in [11].

4.2 The scope of SoaML

The SoaML specification does not prescribe how non-functional constraints should be described. The specification states that policies should be modeled as constraints that can be owned rules of any model element, including service ports and service participant components. The actual representation of these policies are out of scope for the SoaML specification [11]. These UML model elements can then contain constraints that express quality requirements. UML constraints are an extension mechanism that enables one to refine the semantics of an UML model element. A constraint refines a model element by expressing a condition or a restriction in a textual statement to which the model element must conform [57]. UML constraints can be used in the service interface and contract-based approaches to specify policies. These UML constraints can be specified in, for instance, the Object Constraint Language (OCL) [58] or in natural language. In addition, the OMG QoS specification [59] may be used to model QoS constraints for services.

To facilitate the organization of models and constraints on models, SoaML introduced categorization. A certain model may be used for many different purposes and viewed from the perspective of many different stakeholders. As a result, the information in a model may need to be organized in various ways across many orthogonal dimensions. Categorization may also be useful for describing constraints, policies, or qualities of service that are applicable to the categorized element. For example, a model element in the service layer of an SOA might have additional policies for distribution, security, and transaction scope [11].

In summary, SoaML gives some hints on the specification of constraints on service models, but it does not prescribe how constraints should be defined nor give examples of constraints. The description of a policy in a service contract is possible with UML constraints, but the SoaML specification gives no further guidelines on how this can be done. The enforcement of policies and non-functional constraints of a service contract are out of the scope of SoaML, since enforcement of policies belongs in the platform-specific domain.

4.3 Available QoS modeling approaches

In this section some options to model quality attributes are investigated. We investigated how we can formally, unambiguously and correctly include the quality requirements in the SoaML models. In fact, all QoS modeling approaches need modeling guidelines which define where and which QoS requirements can be expressed and in which place in the SoaML models. These guidelines should be defined in order to optimize the selected lifecycle activities from Section 2.10. For each approach we list the observations we made and in the end we compare the approaches and make a selection. We use the following main criteria for the evaluation of the modeling approaches:

1. **QoS definition level**: Can be used to systematically define QoS aspects and metrics, i.e., the quality aspects and metrics listed in Section 2.9 for requirements of the selected activities in Section 2.10.
2. **QoS usage level**: Can be used to systematically use, organize and link QoS requirements to SoaML model elements, i.e., use and connect the defined QoS metrics to SoaML model

elements of the case study. This includes instantiating a defined QoS metric so that we can set a value to it, which then expresses a QoS requirement.

3. **Transformations**: Can be used in automatic model transformations. Some techniques are more suitable than others in model transformations, see also Section 3.2. We also take into account how the technique can be used with our tool preferences as explained in Section 3.4.1.
4. **Tooling**: Can be used with the SoaML tools as selected in Section 3.4.1, or has its own tooling that can be used to specify the QoS.
5. **Usability**: Is easy to use from the stakeholder's point of view.

We identified several ways to model QoS for SoaML model elements. Each way is briefly discussed below, explaining some potential benefits and drawbacks. In the remaining part of this section several concrete techniques are discussed in more detail.

- **Textual descriptions**. A first option to specify QoS requirements are textual descriptions. UML constraints are an approach to textually describe constraints in models [57], and can be used to add quality attribute specifications to SoaML models. This approach requires no further extensions to SoaML. In most cases, users prefer to express QoS requirements using natural language, given that this does not require additional specific expertise because the specification can be done informally [60]. However, it is advisable to use some systematic notation in order to avoid ambiguity, avoid misinterpretation and to ensure proper understanding [60]. A more systematic specification of QoS requirements would be obtained by using OCL instead of natural language to specify the UML constraints.
- **QoS languages**. Another option is to model QoS attributes in SoaML models by using a QoS modeling language, e.g., a UML profile which can be linked to SoaML. A possible choice would be OMG's profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms (QoS&FT) [59] which is given as an example in the SoaML specification [11]. This profile can be used in combination with SoaML, i.e., the SoaML models contain references to quality attributes specified using the QoS&FT profile. Guidelines should describe which and where these references could be placed in the SoaML models. A potential benefit of this approach is that we can define and use QoS aspects in a systematic way. A potential drawback would be increasing complexity of the transformations to PSM-level models, because additional languages (in addition to SoaML) are involved.
- **Metamodel extension**. A third option is a heavyweight extension in which the SoaML's metamodel is extended to model quality attributes. A similar approach has been reported in [61] where SoaML is extended to model security aspects. In this approach, SoaML can be extended, for instance, with the model elements of an existing QoS language as explained in the previous approach. A benefit would be that no new languages are used, and thus the model transformations will possibly be less complex. A potential drawback is that heavy-weight extensions require more effort to built and maintain, and existing UML editors are not usable.

4.3.1 SoaML UML constraints

UML allows one to add constraints to SoaML's model elements, such as to SoaML's participant, contract and interface model elements. A constraint refines a model element by expressing a

condition or a restriction in a textual statement to which the model element must conform. Constraints represent conditions or restrictions that cannot be represented in any other way in UML. An example of a constraint is a condition of an attribute that must have a specific value. There are several ways to represent these constraints [57]. Languages in which the constraints can be specified are natural languages such as English, programming languages (Java), mathematical notations and OCL [57].

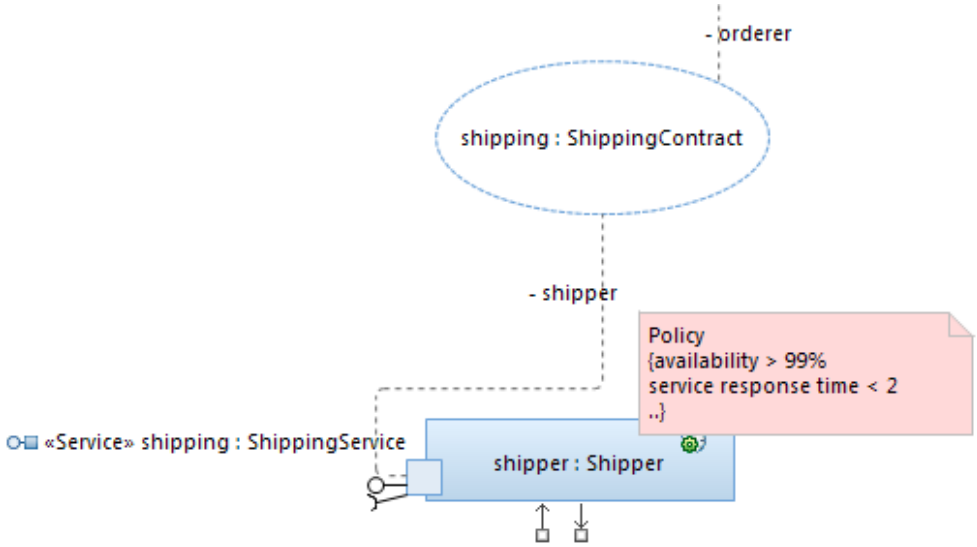


Figure 4.2: Policy specification as UML constraint

A first option is to express the policy in natural language (or some pseudo language) as shown in Figure 4.2. In this case we have to define guidelines on where and how to specify QoS requirements in the SoaML models so that stakeholders responsible for the transformation to PSM-level models can transform these values. We recognize that some commonly understood language rules are essential for automatic model transformations. Without an agreement about a specific format to express the QoS requirement (in that case you actually define a QoS modeling language), the natural language approach is only suitable for manual transformations because natural language can then only be understood by people. To illustrate this approach, we choose to add the policy to the shipper that is modeled in a SoaML services architecture diagram as shown in Figure 4.2. In practice, the policy specification could be added to any modeling element in SoaML. When using this approach, we should define extensive guidelines that help clarify the positions and semantics of the QoS specifications being modeled in the SoaML models.

Table 4.1: Natural language UML constraints approach observations

Simple to use.
Potentially easy to understand with modeling guidelines.
Can be added to any model element in SoaML.

Can express any QoS requirements.
Directly applicable in UML editors that support UML constraints.
Does not enable systematic definition of QoS aspects and metrics at design-time.
Does not enable systematic usage and expression of QoS requirements.
Requires manual QoS-aware transformations to PSM-level models when used language is not regular.
Requires extensive modeling guidelines to represent and interpret QoS requirements (to define semantics of QoS aspects, metrics and values).

The UML element constraints also allow us to use the Object Constraint Language (OCL) to define constraints. OCL is a standard language used to describe constraints on UML models [58]. The benefit of OCL is that it describes the constraints in a standardized way, and thus can be used to express QoS requirements in a more systematic way, as indiciated in our second criterion. However, a policy specification as in Figure 4.2 cannot be directly represented in OCL, since the SoaML models should have elements (such as attributes) where OCL expressions can be written about. The SoaML participant has no availability, performance or any other QoS attributes by default so we would have to add these attributes. QoS requirements can be defined by adding attributes to a participant, and then adding OCL constraint on these attributes. This means we can actually define QoS aspects and metrics, but not in a systematic way as prescribed in the first criterion.

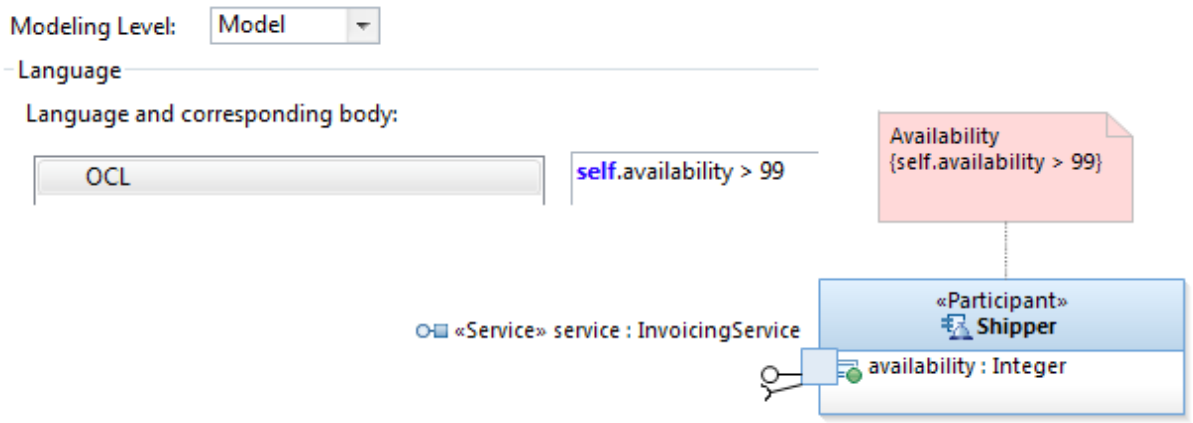


Figure 4.3: Policy specification as UML OCL constraint

To illustrate this approach in Figure 4.3 we defined an availability attribute of the type Integer in the Shipper participant, and added an OCL constraint that defines an availability of at

least 99%.

A drawback of this method is that the QoS specification requires the modeling of additional attributes in diagrams, while these attributes do not describe any functionality of the SOA application. As with the natural language approach, in order to organize and use the QoS specifications in transformations, we have to develop extensive guidelines on how and where to specify certain QoS aspects in SoaML models. These guidelines also should describe the semantics of the QoS specifications. Another drawback is that the transformations to PSM-level models will possibly become complex.

For instance, during the transformation, the attribute named availability in a participant diagram should be transformed to a PSM-level model that defines an availability described by the corresponding OCL constraint. In order to realize a transformation like this, this example requires guidelines on the positioning, naming and type of QoS attributes in SoaML models, and also requires the OCL metamodel to navigate and understand the OCL constraints that express the QoS requirement. This approach does not allow to systematically defined QoS aspects and metrics at definition level, e.g., defining an availability metric that can be reused. But this must be achieved indirectly via properties.

Table 4.2: OCL UML constraints approach observations

Able to systematically use and express QoS requirements, with additional guidelines to define the semantics.
Potentially easy to understand with modeling guidelines.
Relatively simple to use, only requires OCL knowledge.
Directly applicable in UML editors that support OCL constraints.
Does not enable systematic definition of QoS aspects and metrics at design-time.
Requires additional QoS attributes in SoaML models so that non-functional attributes are mixed with functional attributes. This might lead to misunderstanding and makes it more difficult to organize the requirements.
Cannot be used with any model element in SoaML, since some elements do not allow the definition of attributes.
QoS-aware transformations to PSM-level models will become complex because this approach lacks definition of QoS aspects and metrics, and the transformation requires the OCL metamodel to interpret the QoS requirements.
Requires extensive modeling guidelines to represent and interpret QoS requirements (to define semantics of QoS aspects, metrics and values).

4.3.2 QoS languages

In the last decade, there have been many proposals that apply QoS specifications to SOA models [24, 62, 60, 63]. We looked for recent extensions that can be used in combination with UML to represent QoS requirements at the PIM-level, because we want to include these requirements in SoaML models.

A few examples are UML-Q (UML for the QoS information specification) and UML-M [64], QoS Modeling Language (QML) [63] and OMG's UML Profile for Modeling Quality of Service and Fault Tolerance (QoS&FT) [59]. Each of these languages are briefly discussed below.

OMG QoS&FT

The QoS&FT profile was adopted in 2004 [59], and deals with non-functional aspects, more specifically QoS and Fault Tolerance (FT) issues. The general QoS framework provides support to ensure consistency in modeling different QoS aspects. The FT part assesses the capability of a system to deliver continuous and failure-free service. Since our research concentrates on QoS requirement specification approaches, the FT part of the profile is considered out of scope and ignored. In order to evaluate the profile we had to (manually) integrate the profile in the test environment.

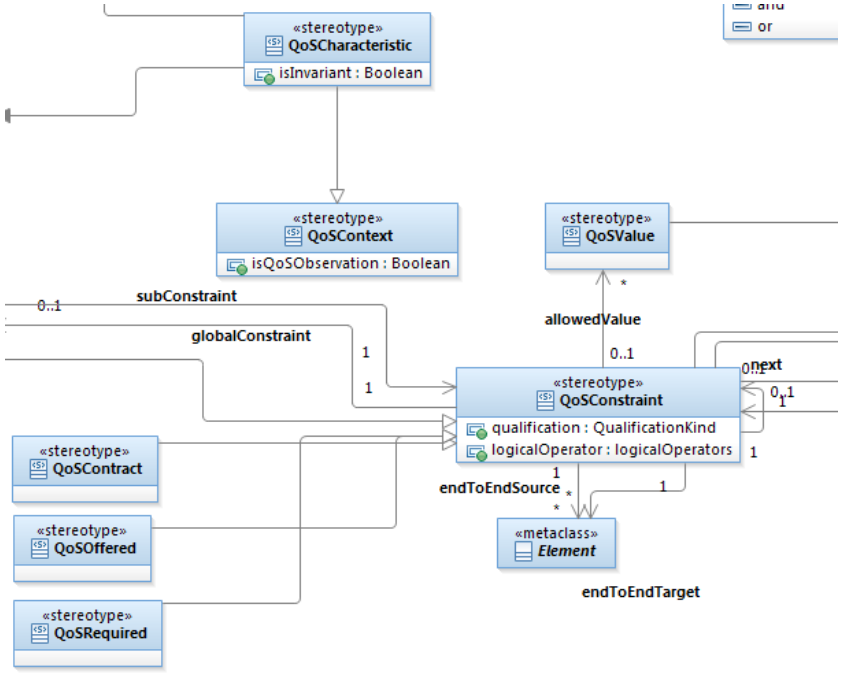
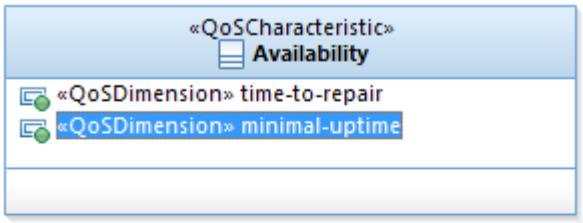


Figure 4.4: Part of the QoS&FT profile implementation in RSA

The QoS&FT profile consists of the following subprofiles: QoS characteristics, QoS constraints and QoS levels. Each subprofile has associated UML concepts (metamodels and stereotypes) and well-formedness rules and semantics expressed in OCL [59]. Quality requirements in QoS&FT are defined in three steps:

1. The definition of QoS characteristics.
2. The definition of a quality model in which QoS characteristics parameters are assigned to actual values).
3. The annotation of UML models with QoS constraints and QoS values according to the QoS characteristics defined in the quality model [60].

We have implemented part of the QoS&FT profile (Figure 4.4) to illustrate how we could use this profile. Figure 4.5 shows the defined availability QoS characteristic and some QoS dimensions which we apply to the case study.



Property	Value
QoSDimension	
direction	2 - undefined
statisticalQualifier	1 - minimum
unit	percent

Figure 4.5: Definition of availability characteristic

QoS constraints can be defined in several way with QoS&FT as explained in [59]. One way is to define them as UML constraints with stereotype «QoSRequired», «QoSOffered», or «QoSContract». The constraint then defines an OCL expression that limits the allowed values of the specific availability QoS characteristic, see Figure 4.6. This approach also uses UML constraints and OCL, however, in this approach we do not have to define additional attributes in the participant class itself, but we can use the QoS modeling elements provided by the QoS&FT profile.

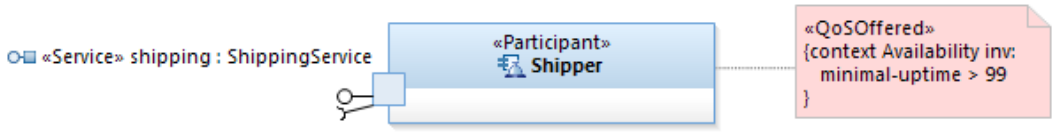


Figure 4.6: Defining a QoS constraint with UML constraints

Another way to express QoS constraints is to use an UML Dependency relationship with stereotype «QoSRequired», «QoSOffered», or «QoSContract». This relationship represents the dependency of a model element on a «QoSValue». This approach uses QoS value objects instead of OCL to express the allowed values of a QoS requirement, as shown in Figure 4.7.

The QoS&FT profile allows us to systematically define QoS aspects and metrics, and also instantiate and use them. OCL is therefore not required to express the requirements. This approach seems to be more suitable for automatic model transformations because there are dedicated model elements available that can be used for mappings, there are elements available to connect the requirements to SoaML model elements, and the transformation does not have to manipulate OCL expressions.

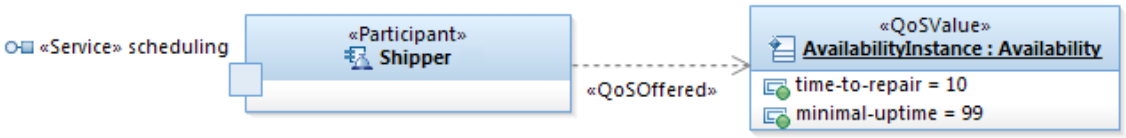


Figure 4.7: Defining a QoS constraint with UML dependency and QoS value objects

The work of [60] shows that the practical use of the profile is still limited. This is probably because the profile lacks good examples and is relatively complex for users. Another drawback is the tool support which is still limited, i.e., as far is we know, the QoS&FT UML profile is directly include in any UML tool.

Table 4.3: QoS&FT approach observations

Enables systematic definition of QoS aspects and metrics at design-time.
Enables systematic use and expression of QoS requirements.
QoS requirements can be organized and categorized.
Enable a clear approach to link QoS requirements to SoaML model elements.
Usable in any existing UML editor.
Automatic QoS-aware model transformations are facilitated.
From a stakeholders' perspective harder to use because it contains many modeling elements and lacks good examples.
Poor direct tool support.
Profile-based QoS-aware transformations are harder to implement than metamodel-based transformations, and next to SoaML, an additional profile (QoS&FT) must be considered in transformation.

UML-Q

The work of [64] also proposes two UML profiles to extend UML (2001). The first one (called UML-Q) has, for instance, been developed to allow specifying (at design time) how long it should

take to obtain the results of the invocation of a method of a particular object of an application. The second one (called UML-M) has been developed to allow specifying how to check whether that requirement is satisfied or not during runtime, by introducing new code into the application, for monitoring external events, etc [64]. Only UML-Q matches the scope of our research because we are looking for a profile that allows us to model PIM-level (design-time) models.

When compared with the QoS&FT profile, UML-Q seems to be outdated and not used in practice. UML-Q also defines a QoS constraint element which is similar to the one in QoS&FT. Other similarities are the QoS characteristics and QoS characteristic feature, which is named QoS dimension in the QoS&FT profile. QoS requirements are expressed differently in this profile. For instance, QoS requirements are organized in UML packages and only OCL can be used to express the requirements, by specifying the allowed values of a QoS requirement.

Table 4.4: UML-Q approach observations

Enables systematic definition of QoS aspects and metrics at design-time.
Enables systematic use and expression of QoS requirements, but actual representation is limited to OCL.
QoS requirements can be organized and categorized.
Usable in any existing UML editor.
Automatic QoS-aware model transformations are facilitated.
From a stakeholders' perspective harder to use because of additional modeling elements.
No direct tool support, i.e., the profile is not included nor suitable to use in existing tools.
Automatic model transformations require the OCL metamodel to interpret the QoS requirements expression.
Profile-based QoS-aware transformations are harder to implement than metamodel-based transformations, and next to SoaML, an additional profile (UML-Q) must be considered in transformation.

Other languages

Another language to represent QoS is the QoS Modeling Language (QML) by HP (1998) [63]. QML is different in that it proposes a separate language to express the QoS requirements, and uses stereotypes to link the QoS specifications to UML models. QML seems to be easy to use and expressive. Some disadvantages are that no metamodel is available, the method of linking QML specifications to UML models limits the specification of QoS in SoaML models and there is no tool support [60].

We also looked at the CQML QoS profile definition of [65] which has been submitted as a reaction to the request for proposal of the now available QoS&FT. We have not evaluated this

profile further because we observed that the QoS&FT profile provides similar functionality but has more capabilities.

The same applies to other work, such as [62, 66] which we consider less suitable for the specification of QoS in SoaML because they are less expressive or less suitable for model transformation than languages such as QoS&FT.

4.3.3 SoaML metamodel refinements

A heavy-weight metamodel extension is also an option to add QoS modeling capabilities to SoaML. This approach seems to be most suitable for the transformations to PSM-level models because in this case we only have to consider the extended SoaML metamodel and no additional profiles. There are several QoS metamodels available. An example is the metamodel presented in [67], which is a straightforward model to express QoS requirements but does not prescribe how these requirements can be linked to UML models.

A better choice would probably be a metamodel extension obtained by using the metamodel derived from one of approaches discussed in the previous section. One example is the PIM4SOA metamodel [43] that uses part of the QoS&FT profile to enable QoS specifications. SoaML4Security also uses part of the QoS&FT profile for QoS specification [61]. In this project we would then extend SoaML's metamodel with, for instance, QoS&FT modeling elements. In this evaluation we assume SoaML's metamodel is being extended with such QoS modeling elements sufficient to model QoS at both definition and usage level.

With this approach some drawbacks of the profile-approach disappear, such as the potentially complex profile transformations, and we can choose to keep the QoS part simple by only adding the necessary new QoS modeling elements. Furthermore, graphical notations to facilitate QoS specifications are easier to define with a metamodel extension.

The metamodel approach also has drawbacks in comparison with profile-based approaches, such as the definition of a new language, which cannot be used directly with other UML tools, and thus lacks visual modeling. A potential benefit is that the metamodel implementation can easily be reused by others to create their own domain-specific language with elements that are not supported by UML.

Table 4.5: Metamodel-based approach observations

Enables systematic definition of QoS aspects and metrics at design-time.
Enables systematic use and expression of QoS requirements.
QoS requirements can be organized and categorized.
Automatic QoS-aware model transformations are facilitated.
Flexible to extend in future work.
No direct tool support.
Defines a new language.

4.4 Selection QoS modeling approach

To conclude the evaluation of several QoS modeling approaches in SoaML we made an overview and final selection. Figure 4.8 shows a comparison table of the evaluation based on the criteria of Section 4.3.

	QoS definition level	QoS usage level	Transfor-mations	Tooling	Usability
Natural language UML constraints	-	-	-	+	+
OCL UML constraints	-	o	o	+	-
QoS&FT profile	+	+	o	o	o
UML-Q	+	+	o	-	o
Metamodel-based	+	+	+	-	+

Figure 4.8: Evaluation QoS modeling techniques for SoaML

First of all, the approaches in which we use UML constraints (natural language and OCL) and no further extensions only seem to be suitable for simple and straightforward QoS specifications. Complex QoS specifications with this approach are hard to organize, and the specifications can lead to cluttered models and misinterpretations by the stakeholders. This approach might be suitable in relatively simple projects where manual QoS transformation are no problem, but in our research, we are investigating techniques that allows us to specify QoS aspects and metrics in a systematic way (first two criteria) and we want to use them in automatic model transformations, making this approach unsuitable.

The use of additional modeling elements in the form of UML profiles (QoS&FT and UML-Q) is in this case a better way to specify QoS requirements. These approaches can be used directly with SoaML; we can model QoS requirements in a systematic way, and the more extended profiles are able also to represent and organize the QoS requirement in a suitable way. In this approach we do not have to define additional attributes in the participant class itself, but we can use the QoS modeling elements provided by the profile to define QoS aspects and metrics at definition level. One disadvantage is the more complex model transformation to PSM-level models as we pointed out in Section 3.2, and explained in the profile evaluations. Also the tool support is poor for these profiles since the profiles are not include or directly suitable to use in existing UML tools. The QoS&FT profile seems to be the most complete standard QoS specification language of the evaluation [60], and therefore we chose this profile as the language to model the QoS. One of the drawbacks of this profile is its lack of practical examples and poor tool support.

The metamodel-based approach is in our case more suitable for the transformations, but lacks graphical modeling support at this moment. This approach requires us to do some additional preparations in order to make use of the metamodel extension. For example, building the metamodel and creating a (visual) editor.

We decided to use a combination of the last two approaches. As explained in Section 3.4.1, we use IBM Rational Software Architect (RSA) to model the SoaML models using the SoaML

profile. In this tool we are able to express QoS requirement in SoaML using the QoS&FT profile in the intended visual way. In order to transform the SoaML models to PSM-level models we have used EMF. We extended SoaML with QoS&FT profile elements as metaclasses, and rebuilt the profile-based SoaML models created in RSA to metamodel-based models (see Section 6.1.2).

The best solution would be a SoaML modeling tool that uses SoaML's metamodel and allows us to extend it. However, to our knowledge, such a tool does not exist and it is not feasible to build one in this project. Our choice has been convenient from the point of view of tool support because we think it is inconvenient to evaluate SoaML with textual or tree models in EMF. Therefore we use RSA and QoS&FT in the remaining part of this chapter.

4.5 Implementing SoaML models

In this section the case study is further elaborated with the selected SoaML QoS specification technique based on the activities listed in Section 2.10.

We modeled the case (Section 4.1) with the following elements. We defined three participants: a company, a manufacturer and a shipper (Figure 4.9). The company is the customer that orders products, i.e., this participant needs some services for his needs. The manufacturer provides a purchase service with which customers can order products. This service consists of a request-response function that requires some order information and returns an invoice (we assume this invoice is generated automatically). The shipper provides two services:

1. A shipping service which can be used by the manufacturer to ship ordered products. This service is a bit more complicated because it uses asynchronous communication, i.e., the shipper responds when the shipping schedule is ready. A service contract describes how this interaction exactly works.
2. A shipping status service which can be used by the company to check the current shipment status.

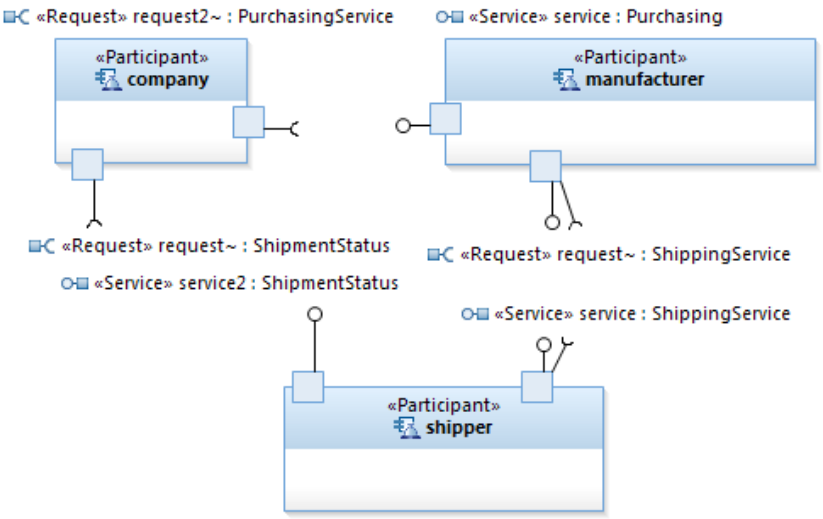


Figure 4.9: The involved participants

4.5.1 Quality requirements

Several common QoS attributes are listed in Section 2.9. In this section we apply these QoS attributes to our case. We assume that all participants have some quality expectations from the service they provide and/or consume, and we model these expectation in terms of policies and contracts.

We assume that the company requires an availability of 99% of the purchase service and expects the invoice within 30 seconds, i.e., the service response time should be smaller than 30 seconds. Because the manufacturer cannot handle more than 10 order requests per minute, this limitation in terms of maximum throughput must be described as QoS offer in the manufacturer's policy. Furthermore a service contract between the manufacturer and the shipper describes that these parties agree on the maximum number of allowed failures, which must not exceed 3 failures per day on average.

First of all we organized all SoaML models in packages. For instance, the SoaML participant models are in the "participants" package, the service interface models are in the "interfaces" package etc. We defined a new "qos" package in which we defined the required QoS&FT QoS characteristics. These "qos" package thus contains the QoS aspects and metrics which can be instantiated and referred to from other packages.

4.5.2 Modeling policies

The company's policy should express some QoS availability and performance requirements. Therefore we defined some QoS characteristics for these requirements as shown in Figure 4.10. The minimal-uptime dimension for availability has the following properties: increasing direction, minimum qualifier and percentage as unit. This means; an uptime of 99% is better than 98% (direction), the values should be interpreted as a minimum value (qualifier) and the value is expressed in percentage. These properties are also defined for the service response time (SRT) dimension.

The next step is to specify the QoS requirement in the SoaML models. Because we are modeling the policy for the company, these requirements should be defined somewhere in the company participant.

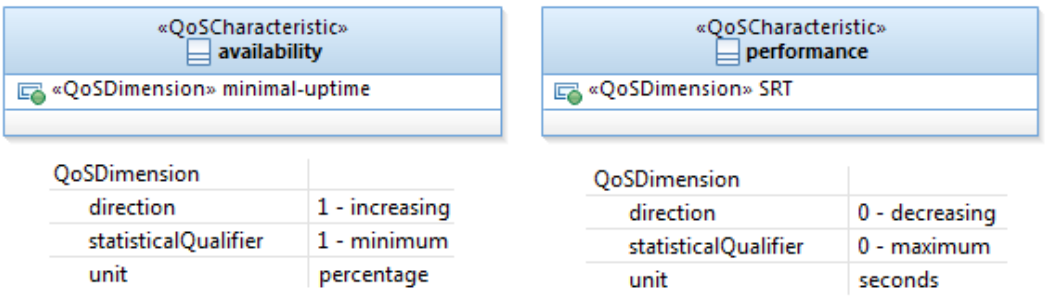


Figure 4.10: Availability and performance characteristics

We have decided to model the QoS values in a policy package inside the participant. These QoS values are instances of QoS metrics defined in the "qos" package. The policy of a consumer

has been modeled on SoaML request ports, and the policy of the provider has been modeled on SoaML service ports. With SoaML dependencies we can express which QoS requirements belong to which ports. Figure 4.11 shows that the required QoS for the purchasing service request is defined as an uptime of at least 99% and a maximum SRT of 30 seconds. Figure 4.11 shows the organization of the QoS specifications on the left side.

A policy should have one subject. For example, in our case, the policy relates to the PurchasingService. If we want to specify QoS requirements for the ShipmentStatus, these specifications should be included in a separate policy package.

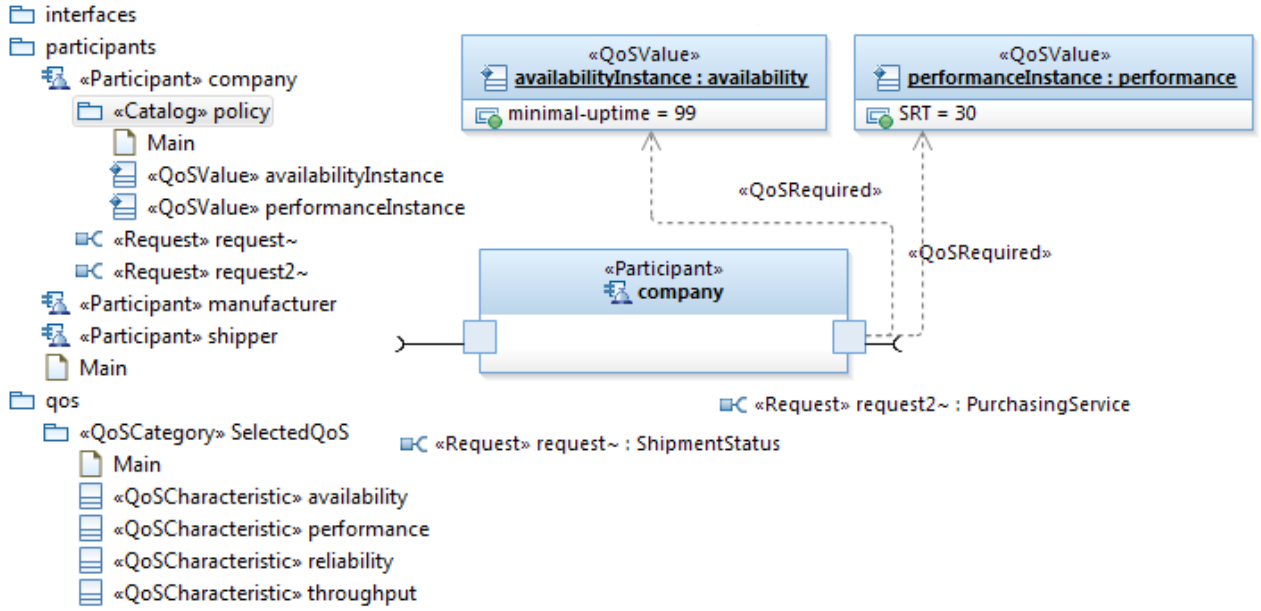


Figure 4.11: Linking QoS requirements to participant

We also considered that QoS requirements may be optional. For instance, when at least one of the two requirements should be satisfied, this can be modeled by setting the logical operator of these dependencies to "or". When all requirements must be satisfied, all operators must be set to "and". When the logical operator is not set, we assume that the requirement must be satisfied. Combinations are possible, e.g., two requirements must always be satisfied, and three requirements are optional. We consider all requirements in a policy package as one group.

More complex situations, such as grouping requirements, can be modeled by composing the policy from nested sub-policies, which is possible with our approach of defining policies in packages. Nested policies can be added as packages named "policy", and can be structured in the same way as the parent policy. Figure 4.12 shows an example of a nested policy. Assuming all logical operators are set to "or", we can interpret this specification as either the Requirement 1 or 2 have to be satisfied and either Requirement 3 and 4 have to be satisfied. Without a nested policy, i.e., when all requirements were specified at the same level, the interpretation would be Requirement 1, 2, 3 or 4 have to be satisfied.

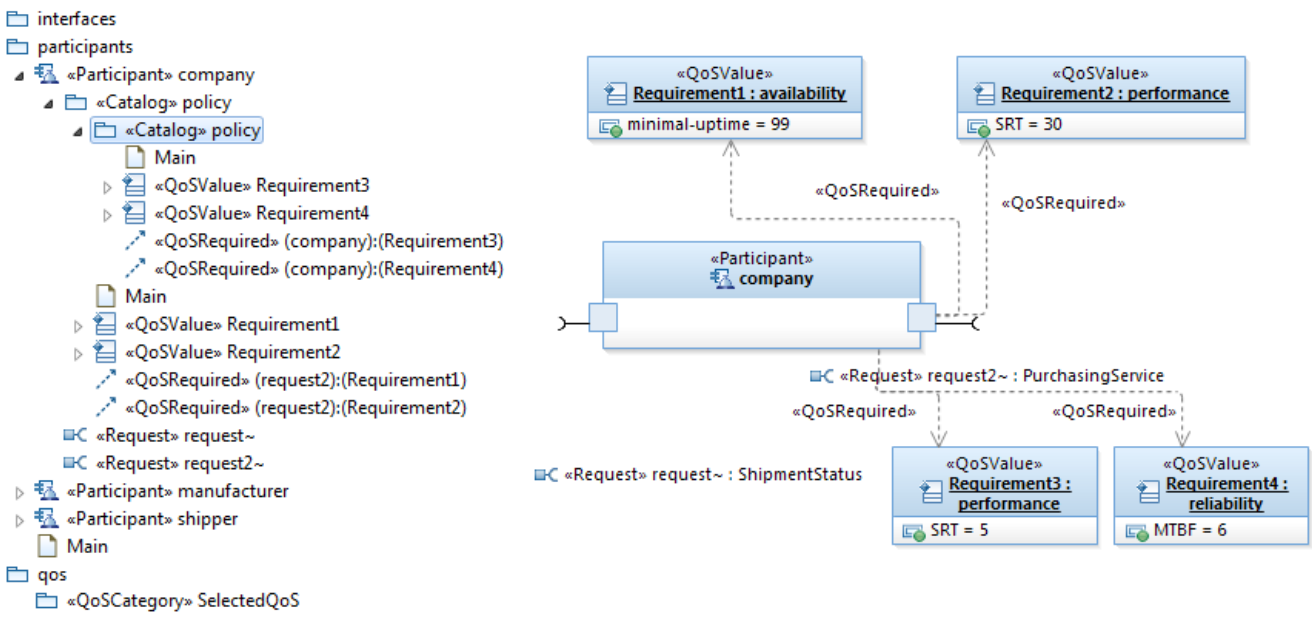


Figure 4.12: Nested QoS specifications

The policy of the manufacturer is also modeled with this approach. In our case study we have modeled a policy of a provider. The differences shown in Figure 4.13 are the QoS offered dependency instead of a QoS required dependency, and it is linked to a service port instead of a request port.

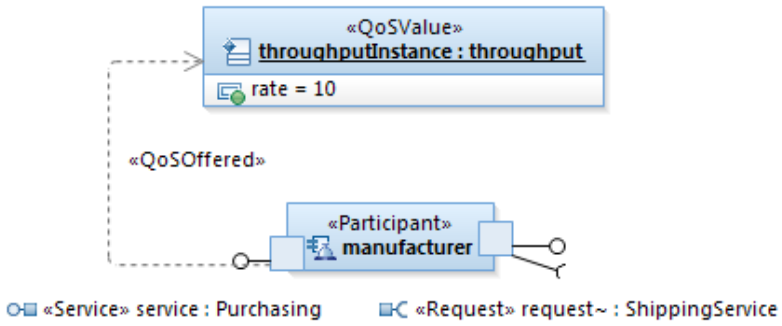


Figure 4.13: Expressing QoS offers

4.5.3 Modeling QoS in service contracts

In some cases QoS requirements can already be modeled in the service contract at PIM-level as explained in Section 2.10, for instance, in case a service contract describes certain QoS requirements that must be agreed in order to realize a well-functioning SOA application. In our case, we took the mean-time-between-failure (MTBF) QoS reliability requirement as example.

This QoS agreement should express that the MTBF must not exceed 3 failures per day on average. The MTBF QoS characteristic has the following properties: direction decreasing, max-

imum qualifier and failures/week as unit. The QoS value that must be linked to the contract should define a maximum of (3 * 7 days) 21 failures per week, assuming every day has an equal chance of having failures.

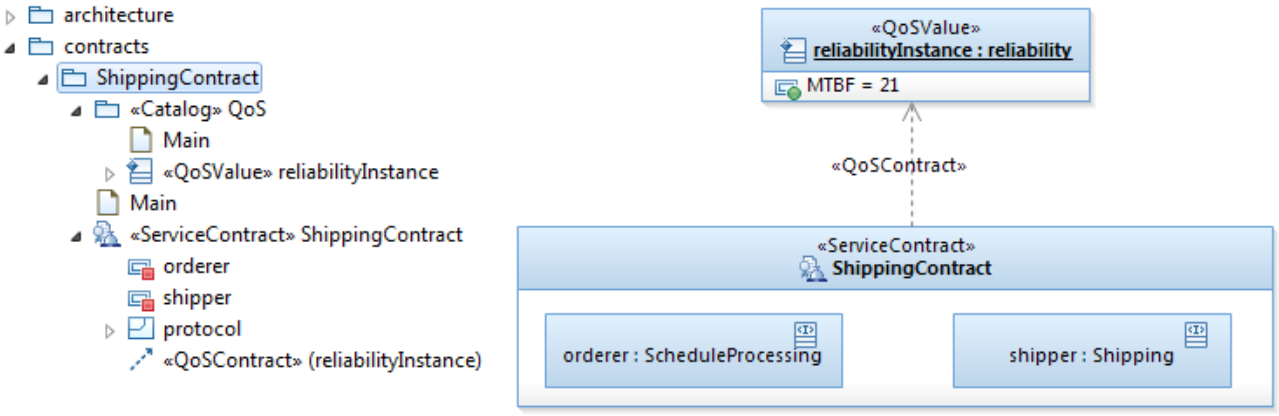


Figure 4.14: QoS specification in service contract

We think the most convenient place to specify the QoS requirements is in the service contract model element itself. However, packages cannot be specified inside a SoaML service contract because of limitations of the UML metamodel, i.e., a UML collaboration element cannot contain a package. A possible solution is to define a root package for all contract specifications, where we can define a QoS package which is used in a similar way as the policy in the previous examples, so that we can use the QoS values in the collaboration diagram. With this approach the QoS specifications are at the same level as the service contract collaboration diagram. Figure 4.14 shows how the reliability requirement is linked to the SoaML service contract. The left side of Figure 4.14 shows how we organized the QoS packages.

Similarly as with the policy specifications, optional requirements can be set with the logical operator property, and QoS packages can be nested to the define more complex alternatives.

Chapter 5

PSM-level QoS modeling

This chapter gives an overview of the available PSM-level modeling techniques to which SoaML models can be transformed. With these languages we build PSM models that allow the representation of the defined QoS aspects, metrics and requirements at a lower abstraction level than SoaML. These models can be used by stakeholders to realize the SOA application on a certain platform.

5.1 Search scope

There are many techniques available which can be used to realize web services on a specific platform. We are specifically searching for (preferably standard) techniques which we can use to describe QoS, and combine them with other commonly used techniques, such as, the Web Services Description Language (WSDL). We do not search for techniques that monitor, realize or enforce the required or offered QoS, but techniques that allow the description of the QoS as done in Section 4.5 but then at the platform level. We preferably search for generic techniques which can be used to express many QoS requirements, so that this technique can be applied to optimize many of the stakeholders activities listed in Chapter 2.

Functionality of a web service is often defined in WSDL, an XML standard which allows the specification of the operations that the web service can perform. The information provided in the WSDL covers all the data needed to invoke the web service itself, but does not allow the specification of the offered QoS [14].

There have been proposals that extend WSDL to allow QoS specifications, such as [24] and [68]. However, preferably QoS attributes should be described in a systematic way, and loosely coupled with the web service. This is not the case with these WSDL extensions. A better way would be a WSDL description that refers to a policy that specifies constraints and capabilities of the service implementation as explained by [69].

5.2 Available policy languages

In our PIM-level models, we have specified policies for providers and consumers. We are interested in how these policies can be represented at PSM-level, i.e., we search for techniques that can be used to represent QoS requirements of providers and consumers at the platform level.

For each language we list the observation we have made and in the end we compare all languages and make a selection. We use the following main criteria for the evaluation of the languages:

1. **Consumer**: Can be used to represent the consumer policies.
2. **Provider**: Can be used to represent the provider policies.
3. **Policy structure**: Supports modeling of more complex policies by supporting alternatives and priorization etc.
4. **Transformations**: Provides a metamodel that can be used in the transformation, or provides enough information to create a metamodel.
5. **QoS expression**: Provides grammar to express the QoS requirements.
6. **Integration**: Can be used in combination with other commonly used (XML-based) SOA techniques such as WSDL.
7. **Usability**: Is easy to use from the stakeholder's point of view. Considering syntax, documentation and provided examples.

5.2.1 WS-Policy

WS-Policy is a W3C recommendation published in 2007. This standard is used to express web services policies based on their non-functional properties [70]. WS-Policy is a specification that allows web services to advertise their capabilities, requirements, and general characteristics in a flexible and extensible grammar using XML format [71, 26].

WS-Policy also provides a mechanism to link the QoS requirements to the service models. The WS-PolicyAttachment recommendation [72] explains how to to associate policies with their subjects, reference policies from WSDL definitions, or associate policies with deployed web service endpoints.

A policy is defined as a collection of alternatives which is, itself, defined as a collection of assertions. An assertion is used to represent a requirement, capability or a behavior of a service [73]. Listing 5.1 shows a policy that requires a minimal response time of 5 seconds. Line 5 of this listing shows a pseudo code assertion as plain text. In practice we have to use a grammar to express this requirement.

Listing 5.1: Example policy with pseudo code assertion

```
1 <wsp:Policy xmlns:wsp=".../policy">
2   <wsp:ExactlyOne>
3     <wsp:All>
4       MRT = 5 seconds
5     </wsp:All>
6   </wsp:ExactlyOne>
7 </wsp:Policy>
```

WS-Policy introduces a grammar for defining policies, but it does not come with (predefined) assertions. The WS-Policy working group has developed the WS-PolicyAssertions specification to allow the description of some generic assertions (e.g. used character set) which are not related to QoS and are thus not directly interesting for this work [74]. External organizations or communities are in charge of defining domain-specific assertion grammar. For instance, WS-SecurityPolicy [75] and WS-ReliableMessaging [76] define assertions such as IncludeTimestamp, EncryptSignature, and Uses-SequenceSSL to formalize security and robustness requirements [69]. To the best of our knowledge, there is no standard assertion grammar for QoS attributes, such as, availability and service response time.

The exact type of assertions that can and should be used depends on the type of QoS to be expressed, and the environment where the policies will be used. A monitoring or QoS enforcement framework might require a specific kind of assertion that can be interpreted by the framework. One example is the WS-MeditationPolicy by IBM [77], which provides assertions that can be used in combination with their policy enforcement and policy administration products, such as e.g. WebSphere. Another example for a provider are assertions that refer to a service registry entry describing the actual QoS value as done in [78].

For our service models, we could develop some custom assertion grammar to express our QoS requirements. With these custom assertions we could demonstrate how to use the WS-Policy framework in combination with SoaML models.

Table 5.1: WS-Policy observations

Standard technique.
Allows policies to be connected to other techniques such as WSDL and UDDI.
Accepted and adopted language in industry [74].
Suitable to express policies.
Different QoS assertion grammars may be needed for a complete QoS specification, depending on the project.
QoS assertion grammar has to be defined.

5.2.2 UDDI

UDDI (Universal Description, Discovery and Integration) is an XML-based registry which defines a set of data structures to describe web services for the purpose of advertisement and discovery. Service providers can publish their web services to an UDDI registry so that the potential consumers can discover these services through different approaches [14].

A way to represent the quality offered by providers is to include QoS specifications in the service registry. In this way, potential consumers check the offered QoS of a service via the registry. Expressing the quality requirements of consumers is not possible with this approach, since we can only represent the QoS offerings defined in the policy of a provider.
UDDI contains three components which help service consumers find the right service [26]:

1. White pages serve as a telephone book. They give information about the service providers such as contact information and business area.
2. Yellow pages serve as industrial categorization of web services based on standard taxonomies. In UDDI we use the businessEntity element to include white and yellow page information.
3. Green pages offer technical information about the services offered by business identities including reference to specifications and pointers to access points [26]. In UDDI the businessService element is used to describe this information of the webservice, and bindingTemplate elements are used to describe technical information to access the webservice.

The actual green page information is saved in tModels, a generic container for the specification of information. Each tModel should contain an overviewURL, which references a document that describes the tModel and its use in more detail [14]. The green page is also the place to include QoS descriptions, by storing information in such tModels.

There are many different ways to express QoS with tModels [79]. In this approach we can express the provider's policy in a certain QoS tModel, which can then be attached to a web service entity.

Table 5.2: UDDI QoS extension observations

UDDI is a succesful concept [26].
UDDI is a standard.
No dedicated policy language.
No standard way to express QoS with tModels.
Only enables the description of provider's policies. Not suitable for expressing consumer's policies.

5.2.3 Web Services Policy Language (WSPL)

The Web Services Policy Language, WSPL [80] was developed to satisfy a set of use cases and requirements that were collected, reviewed, and published in an open, public forum. WSPL was defined as a strict subset of the OASIS eXtensible Access Control Markup Language (XACML). WSPL is a generic framework just as WS-Policy, but then based on XACML [74].

WSPL uses XACML attributes, which are always name-value pairs. The definition of the QoS attributes used by a particular service is outside the scope of WSPL. The semantic description of the attribute is irrelevant to the WSPL policy and its evaluation. WSPL does not specify either how a policy user obtains values for attributes [80]. Listing 5.2 illustrates a fictitious policy expressing the MRT of a service.

Listing 5.2: Example policy with WSPL

```
1 Policy (Aspect = ''Performance'') {
2  Rule {
3    Attribute = ''MRT'',
4    Value < 5 }
5  }
```

The central problem in WS-Policy, from XACML authors perspective, is the lack of formalization. Therefore, issues such as merging consumer and service provider policies as means of negotiation a commonly accepted policy need domain models. XACML does not have this problem. However, possibly due to popularity of WS-Policy (at least in terms of how well it is known), Oracle (formerly known as Sun) has expressed an interest to reconcile the differences with Web Services Policy Constraints Language (WS-PolicyConstraints), which is basically a stripped down version of WSPL [74]. In fact, WS-PolicyConstraints can be used as an additional layer between the WS-Policy framework and the assertion grammar to add additional semantics.

However, in practice, this language seems to be barely used.

Table 5.3: WSPL observations

Suitable to express policies.
Supports merging of policies.
Less documentation and practical examples.
Seems to be overtaken by WS-Policy (in terms of popularity).

5.2.4 Other techniques

Other policy specification languages include WSDL compositors [74] and WSDL 2.0 "Features and properties" element [81]. We have not further investigated these languages because they are outdated or less suitable for transforming the SoaML models.

Semantic web services initiatives have also looked at the specification of policies at PSM-level. One example is a mapping between WS-Policy constructs and Web Ontology Language (OWL) for Semantic Web Services (OWL-S) constructs [82]. However, these semantic web techniques have been excluded because they take a completely different direction. In this research, we focus on relatively known non-semantic web alternatives.

5.3 Selection of policy language

	Consumer	Provider	Policy structure	Transformations	QoS expression	Integration	Usability
WS-Policy	+	+	+	+	-	+	+
UDDI	-	+	-	o	+	o	+
WSPL	+	+	o	o	o	o	o

Figure 5.1: Evaluation PSM-level policy languages

Figure 5.1 shows a comparison table of the evaluation based on the criteria of Section 5.2. From the techniques we have considered, WS-Policy and UDDI seem to be the most suitable and practical techniques (in terms of usability) to which the models can be transformed. Both techniques are widely used and have good documentation.

UDDI seems to be a practical choice, since it is widely used, and the generated models (e.g. tModels with QoS information) could potentially be directly usable in practice. However, the technique is not perfectly suitable to express policies. We can create our own UDDI models that include some QoS requirements, but these models only consider the provider's point of view and are not directly able to express complex policy specifications (e.g., prioritizing requirements,

and giving alternative options). Possible QoS requirements of a consumer cannot be expressed. Therefore we think in this situation, the other techniques are more suitable.

The main benefit of WSPL over WS-Policy is the support for negotiation. While we consider negotiation as is an important aspect, because it is part of activities as we pointed out in Chapter 2, we also see that there are initiatives that support negotiation with WS-Policy, such as [70]. Furthermore, the WSPL language provides a standard grammar that can be used for assertions, where WS-Policy has not.

Considering all benefits and drawbacks for this project, WS-Policy seems to be the most suitable choice to express policies as defined in a SOA application. The technique can be used in practice to express policies for both consumer and provider, and is also able to express optional QoS requirements. The specification clearly shows how the language works we can therefore built a metamodel for the transformation. WS-Policy does not prescribe an assertion grammar so we are free to choose our own. Because no suitable grammars were found to express the QoS aspects used in this project, we developed an assertion grammar to demonstrate the technique. The development of our own grammar, and the lack of negotiation support are some drawbacks for WS-Policy, but in comparison with WSLA, WS-Policy seems to be a more practical choice in terms of documentation, integration examples and user community.

5.4 Available service contract languages

There are several techniques to describe service contracts at PSM-level. QoS is usually defined in static manually managed SLAs. However, systematic and dynamic languages to represent these contracts are becoming increasingly necessary because they make it possible to interpret these contracts so that, for instance, service providers can be chosen on the fly [83]. In this section we give an overview of several well-known service contract techniques that allow QoS specifications as part of the agreement.

In the same way as we have evaluated policy languages, we evaluate service contract language using the following main criteria:

1. **Contract structure**: Supports modeling of complete contract including aspects such as template, roles, expiration dates, penalties, alternatives and priorities.
2. **Transformations**: Provides a metamodel that can be used in the transformation, or provides enough information to create a metamodel.
3. **QoS expression**: Provides grammar to express the QoS requirements.
4. **Integration**: Can be used in combination with other commonly used (XML-based) SOA techniques such as WSDL.
5. **Usability**: Is easy to use from the stakeholder's point of view. Considering syntax, documentation and provided examples.

5.4.1 WS-Agreement

WS-Agreement [84] is a specification published in 2007 that defines a language and a protocol to create Service Level Agreements (SLA) between a service provider and a service consumer. It is a general framework for XML specification of agreements and agreement templates. An agreement is contract negotiated between a client and a provider, for instance, on QoS attributes

such as latency or availability. An agreement is just a empty canvas, and WS-Agreement allows the use of any language for the actual contained specifications. The structure of an agreement template is the same as that of an agreement, but an agreement template may also contain creation constraint section, i.e. a section with constraints on possible values of terms for creating an agreement. The constraints make it possible to specify the valid ranges or distinct values that the terms may take [85].

An agreement consist of the agreement name, its context and the agreement terms. The context contains information about the involved parties and metadata such as the duration of the agreement [85]. Agreement terms define the content of an agreement. There are several types of agreement terms:

1. Service description terms (SDTs) define the functionality that is delivered under an agreement. A SDT includes a description of the offered or required functionality (the service itself) [85].
2. Guarantee terms define this assurance on service quality, associated with the service described by the SDTs [84].
3. Service level objectives (SLOs) describe the actual bound of the QoS requirement of the service that have to be fulfilled by the provider.

Listing 5.3 shows a stripped example version of an agreement about the ShipmentStatus service. In this example the SLO (line 16) is expressed in pseudo code.

Listing 5.3: Stripped example contract with WS-Agreement

```
1  <wsag:Agreement Name="exampleContract" xmlns:wsag="...">
2    <wsag:AgreementContext>... context of the contract ...</wsag:AgreementContext>
3    <wsag:Terms>
4      <wsag:All>
5        <wsag:ServiceDescriptionTerm wsag:Name="General" wsag:ServiceName="ShipmentStatus">
6              ... describe service and offered functionality ...
7         </wsag:ServiceDescriptionTerm>
8         <wsag:ServiceProperties
9            wsag:Name="AvailabilityProperties" wsag:ServiceName="ShipmentStatus">
10           <wsag:VariableSet>
11             <wsag:Variable wsag:Name="SRT"></wsag:Variable>
12           </wsag:VariableSet>
13        </wsag:ServiceProperties>
14        <wsag:GuaranteeTerm name="FastReaction" obligated="ServiceProvider">
15           <wsag:ServiceScope ServiceName="ShipmentStatus">... refers to service description ...</wsag:
                 ServiceScope>
16           <wsag:ServiceLevelObjective>SRT = 5 seconds</wsag:ServiceLevelObjective>
17        </wsag:GuaranteeTerm>
18      </wsag:All>
19   </wsag:Terms>
20 </wsag:Agreement>
```

The WS-Agreement specification describes that a typical agreement creation process consists of the following three steps [85]:

1. The initiator retrieves a template from the responder, which advertises the types of offers the responder is willing to accept.
2. The initiator makes an offer.
3. The offer is either accepted or rejected by the responder.

Table 5.4: WS-Agreement observations

Standard technique.
Highly extensible; contains several sections where intended users are expected to define domain-specific elements and properties [25].
Accepted and adopted language in industry [86].
A suitable domain-specific language must be defined (or found) to express service descriptions and guarantee terms.

5.4.2 Web Service Level Agreement (WSLA)

A WSLA document (referred to as a WSLA) defines assertions of a service provider to perform a service according to agreed guarantees for quality attributes, such as response time and throughput, and measures to be taken in case of deviation and failure to meet the asserted service guarantee, such as, for example, a notification of the service customer [87].

WSLA consists of a framework for defining and monitoring SLAs for web services. The WSLA language is designed to capture service level agreements in a formal way to enable automatic configuration of both the service implementation of a provider organization as well as the system that is used to supervise the agreed quality of service. To facilitate automatic configuration,an WSLA comprises [87]:

1. A description of the parties, their roles (provider, customer, third parties) and the action interfaces they expose to the other parties of the contract.
2. A detailed specification of the service level parameters (SLA parameters) to which guarantees are applied to. SLA parameters are specified by metrics, which define how to measure an item (in the case of resource metrics) or how to aggregate metrics into composite metrics. A metrics description also includes which party is in charge of measuring and aggregating the metrics and how the metrics can be retrieved.
3. A representation of the parties' obligations. Service level objectives contain a formal expression of the guaranteed condition of a service in a given period. Action guarantees represent promises of parties to take some actions, for example, to send a notification in case the guarantees are not met.

Listing 5.4 shows the structure of an WSLA top-level document.

Listing 5.4: Stripped example top-level contract with WSLA

```
1  <wsla:SLA xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:wsla="http://www.ibm.com/wsla" name="
       ExampleContract" >
2      <Parties>
3      ... participant info ...
4      </Parties>
5      <ServiceDefinition>
6      ... describe service and offered functionality, and metrics ...
7      </ServiceDefinition>
8      <Obligations>
9          <ServiceLevelObjective name="g1" serviceObject=ShippingService">
10             <Expression>
```

```
                <Predicate xsi:type="wsla:Less"><SLAParameter>SRT</SLAParameter><Value>5</Value></Predicate>
            </Expression>
        </ServiceLevelObjective>
    </Obligations>
</wsla:SLA>
```

Table 5.5: WSLA observations

Mature language with practical examples.
Includes elements to define and express QoS requirements.
Lacks flexibility with respect to contract structure [88].
Bound to IBM tools and examples because these are the only resources available.

5.4.3 SLA*

Another promising language that we found is SLA* [88]. SLA* was developed as a generalization and refinement of the web service-specific XML standards: WS-Agreement, WSLA, and WSDL. SLA* uses much of the WS-agreement constructs. The SLA* model was developed as part of the SLA@SOI project [88].

The most significant difference between WS-Agreement and SLA* is the dependency on a certain metalanguage or rendering. WS-Agreement is inherently dependent on XML (and XML Schema) as a description language. References within an SLA document in WS-Agreement, for example, are therefore most conveniently realized using XPath [89].

Another important difference between both languages is that WS-Agreement does not offer any domain-specific expressions that could be used to define ranges or constraints. These are left entirely to the user of the specification. SLA*, on the other hand, can be used to directly describe a large number of resources and constraints on those resources [89].

Listing 5.5 shows the structure of an SLA* contract.

Listing 5.5: Stripped example contract with SLA*

```
sla_template{
    // "qos" = "http://www.slaatsoi.org/commonTerms#"
    uuid = example_contract // universally unique identifier for this SLAT
    sla_model_version = sla_at_soi_sla_model_v1.0
    parties{ ... participant info ... }
    interface_declr{ ... describe service and offered functionality ... }
    agreement_term{
        id = term_1
        guaranteed_state{
            id = guaranteed_state_1
            qos:completion_time(shipping_service) < 5s
        }
    }
}
```

Table 5.6: SLA* observations

Provides good practical examples and clear syntax [88].
Does not depend on XML.
Less popular than WS-Agreement.

5.4.4 SLAng

The primary aim of SLAng is to provide a language that enables the specification of contractual relationships between consumers and providers, and by that allows for a clear definition of obligations on all involved partners with respect to the provided QoS [90]. The SLAng white paper defines an SLA as an arrangement between a customer and a provider, describing technical and non-technical characteristics of a service, including QoS requirements and the related set of metrics with which the provision of these requirements should be measured [91].

The SLAng syntax is defined using an XML Schema, which favours the integration with existing service description languages. For example, SLAng can be easily combined with WSDL [91].

The content of an SLA varies depending on the service offered, and incorporates the elements and attributes required for the particular negotiation. In general, an SLA includes:

1. An end-point description of the contractors (e.g., information on customer/provider location and facilities).
2. Contractual statements (e.g., start date, duration of the agreement, charging clauses).
3. Service Level Specifications (SLSs), i.e. the technical QoS description and the associated metrics [91].

SLAng places emphasis on semantics, providing formal notions of SLA compatibility, monitorability and constrained service behavior. It is, however, targeted at electronic services and provides only a limited set of domain-specific QoS constraints [88].

Even though the language seems to be very well designed, activities on this topic seem to have slowed down or probably stopped, as the last activities on the repository have occurred in 2009 [90].

Table 5.7: SLAng observations

Complete language (with an emphasis on semantics)
Not an adopted language.
No signs for further development.

5.4.5 Other techniques

Another technique that can be used to include QoS contracts is the Rule-Based Service Level Agreement (RBSLA) language [92]. RBSLA is based on the XML-based Rule Markup language RuleML, which is a standardization initiative with the goal of creating an open XML/RDF based

web language for rules [92].

CC-Pi offers a theoretical framework for mapping SLAs to service constraints. The CCPi model is, however, tightly-coupled to the mechanics of negotiation, and does not address common constructs such as agreement party details or service interfaces [88].

We have not further investigated these techniques because they are barely used in practice, or are not as complete and suitable as the techniques discussed before.

5.5 Selection of service contract language

	Contract structure	Transformations	QoS expression	Integration	Usability
WS-Agreement	+	+	-	+	+
WSLA	o	+	+	o	o
SLA*	+	+	+	o	o
SLAng	o	-	o	+	-

Figure 5.2: Evaluation PSM-level contract languages

The most referenced and complete specifications that relate to service contracts for SOA environments, and in particular web services, are WSLA and WS-Agreement [25]. Another suitable and relatively new language is SLA*.

Although the WSLA specification is available for download, few cases studies have been reported outside of IBM. It seems that the specification language has not been widely adopted but rather superseded by WS-Agreement [25]. Furthermore, WSLA has a closed source nature and lacks further development since 2003 [90]. In fact, core concepts of the WSLA were brought into the WS-Agreement [25].

WS-Agreement's main benefit is its versatility. The specification leaves room for domain-specific implementations and imposes minimal restrictions on potential usage scenarios. The basic protocol is another advantage, since it allows an easier adoption to new partners that already use WS-Agreement. The protocol has the further advantages of being symmetric, so that both producers and consumers can be either initiators or responders [89]. WS-Agreement's minimalist nature can be regarded as a disadvantage as well, because a domain-specific language must be provided. In some mid-sized scenarios this overhead might be considerable and thus stakeholders may decide to use other specifications [89].

WS-Agreement currently provides the only standardized way to negotiate and create SLAs. There is a strong user community and their working group, responsible for the development and maintenance of WS-Agreement, is active and responsive [90]. SLA* is also a complete and suitable language with the potential to become popular. In fact, both WS-Agreement and SLA* are suitable for our project. Furthermore, we would like to see that languages support as much as possible of the information from SoaML models. Examples are the role types in a service contract etc. SLA* and WS-Agreement both satisfy this requirement regarding contract structure.

When we compare the two languages we have the impression that WS-Agreement is at this moment the most widely used language, and it is also a standard that goes well with the policy technique we have selected. We chose WS-Agreement as the technique to represent QoS with service contracts at the PSM-level. WS-Policy and WS-Agreement are both standards in the WS-* group, and we recognized similar language constructs when studying these specifications. This means we may be able to create language elements that are usable in both the policy and contract transformations, such as a domain-specific grammar.

5.6 Implementing PSM models

In this section we briefly introduce WS-Policy and WS-Agreement by explaining their metamodels. We also show some example models implemented in these languages. Ultimately, the models shown here are target models which should be automatically generated by models transformations based on the SoaML input models.

Policies

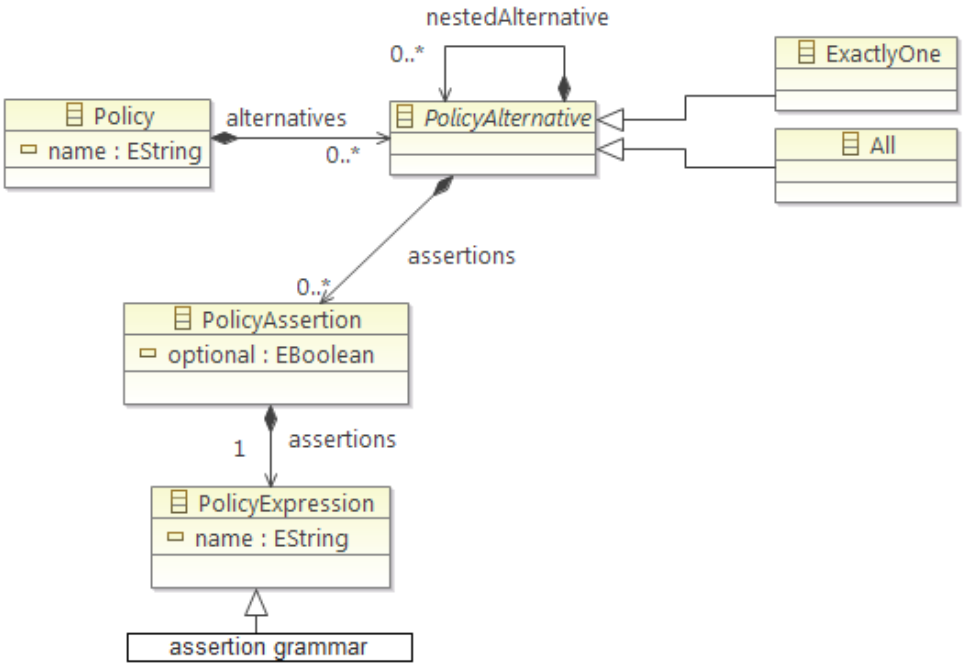


Figure 5.3: Partial WS-Policy Ecore implementation

In order to describe policies in the EMF, we have to built a WS-Policy metamodel in Ecore. Figure 5.3 shows our representation of the metamodel described by the WS-Policy specification [71]. The building blocks of policies are assertions which are composed using alternatives (All, ExactlyOne). For instance, all assertions in an "All" alternative must be satisfied to conform to that policy. In an "ExactlyOne" alternative, one of the assertions must be satisfied. The current

WS-Policy specification lacks the "OneOrMore" operator so the interpretation of "ExactlyOne" may sometimes be "at least one" [93]. We leave the interpretation of this ExactlyOne operand to the user. Furthermore, alternatives can be nested to create more complex policies.

As mentioned earlier, an assertion grammar is needed to express the QoS requirements. We have defined a simple example grammar which is sufficient to model the QoS requirements specified in the SoaML models. Figure 5.4 shows the metamodel of the grammar. A QualityAspect can be used to create a quanti aspect such as performance. A QualityAttribute can be used to represent the actual value and metric of a QoS requirement for that aspect. For example, the service response time of a certain which should be lower than 30 seconds.

With these metamodels we are able to build PSM-level WS-Policy models. The right side of Figure 5.4 shows a WS-Policy that corresponds to the policy specified in SoaML in Figure 4.11.

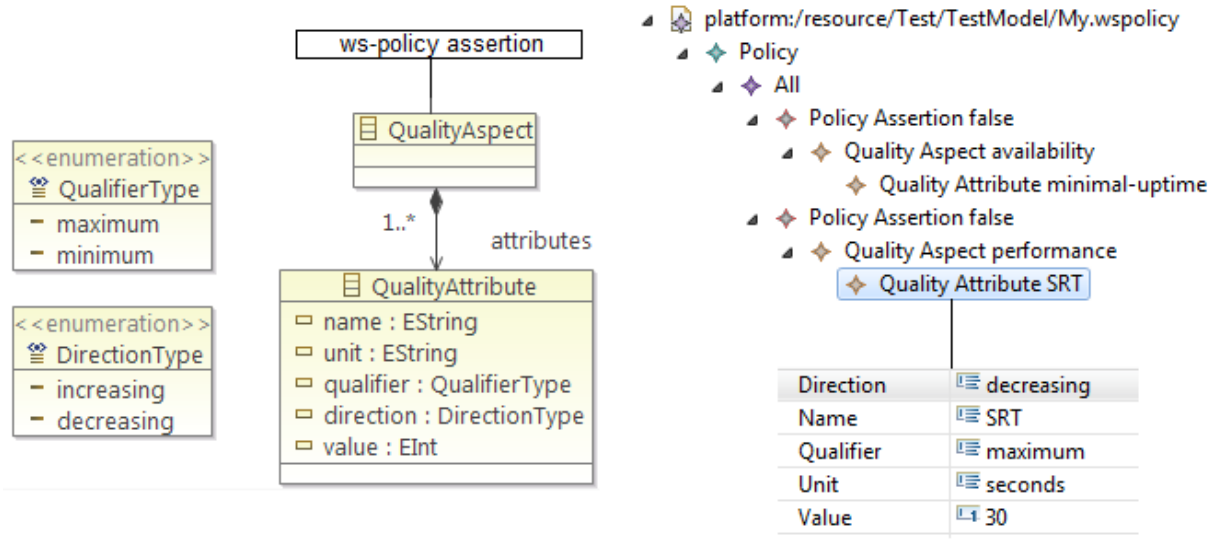


Figure 5.4: Simple WS-Policy assertion grammar (l) and example model (r)

Service contracts

Similarly to the WS-Policy metamodel, we have also defined the WS-Agreement metamodel in Ecore for the service contract transformations. The metamodel in Figure 5.5 shows our representation of the WS-Agreement specification [84], and the WS-Agreement metamodel presented in [94].

This metamodel is more complex than WS-Policy's metamodel. The left side of Figure 5.6 shows the basic structure of an agreement. The agreement has a name and context (AgreementContext in metamodel) explaining the purpose and some metadata of the agreement. The agreement is structured by the term compositors: "All", "ExactlyOne" and "OneOrMore". The first two have the same meaning as in WS-Policy, and furthermore an "OneOrMore" operator is available here. These compositors can be used to structure the actual terms, e.g., description terms, references, properties and guarantee terms (as explained in Section 5.4.1).

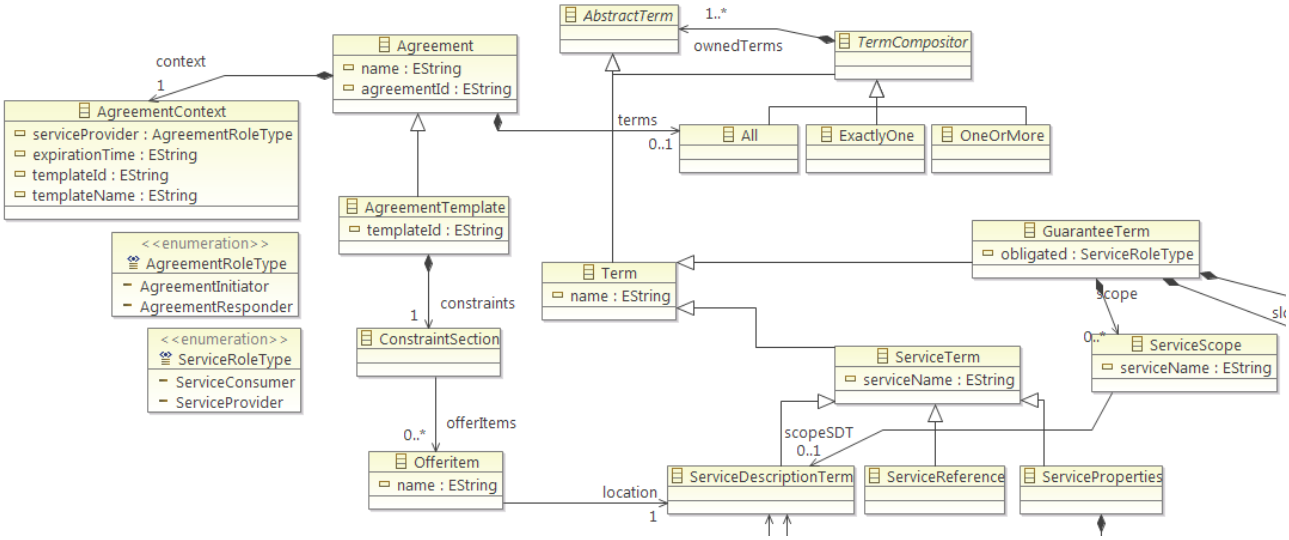


Figure 5.5: Partial WS-Agreement Ecore implementation

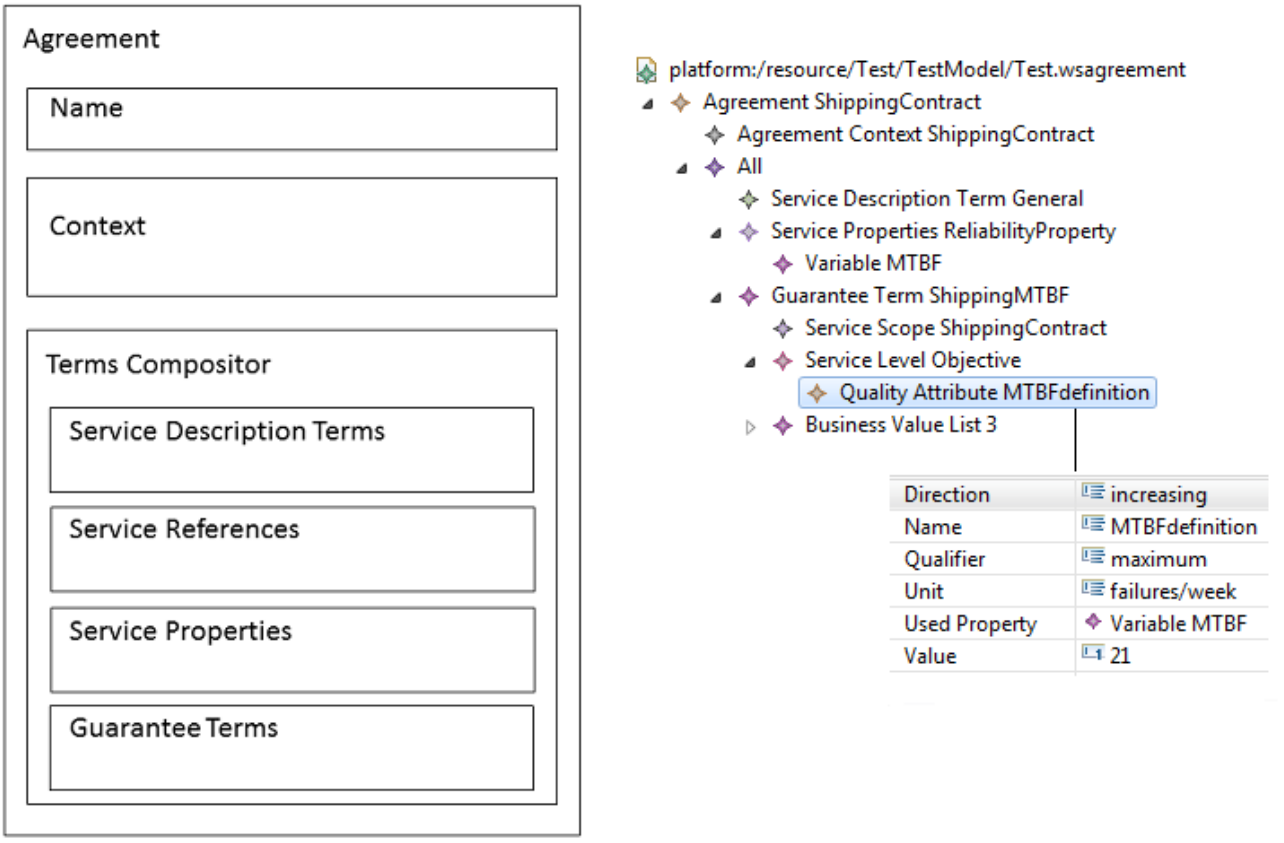


Figure 5.6: Structure WS-Agreement model [1] (l) and example model (r)

WS-Agreement allows the definition of a lot of information in a contract, such as business values, penalties, rewards, expiration times etc. We consider this data out of the scope of this project and focus on the structure and specification of quality requirements in the contract. Therefore, we try to fill in the agreement as completely as possible based on the information included in the SoaML models. Additional information can be added later on by the service provider and consumer during the negotiation.

The right side of Figure 5.6 shows a WS-Agreement that corresponds to the service contract specified in SoaML in Figure 4.14. We reuse the custom grammar, shown in the left side of Figure 5.4, to express the service level objectives.

Chapter 6

Model transformations

This chapter gives an overview of the transformation environment preparation. It further shows the mappings of SoaML to WS-Policy and WS-Agreement, and the performed model transformations of SoaML models to these PSM models.

6.1 Transformation environment

As explained in Section 3.4.1 we have chosen EMF as the environment to build and perform the transformations. In order to do so, some preparations were required.

First of all, it was needed to built the SoaML metamodel in Ecore. We also had to build a part of the QoS&FT specification and combine it with the SoaML metamodel. Furthermore we also prepared and built the PSM-level metamodels in Ecore so we could transform the SoaML models to these techniques. The second step was to rebuild the profile-based SoaML models to metamodel-based models that use our created Ecore metamodel. The last step was to create mappings and design the transformations.

This section explains the steps first two steps that were needed to prepare the environment and metamodels for the transformations.

6.1.1 Implementing the extended SoaML metamodel

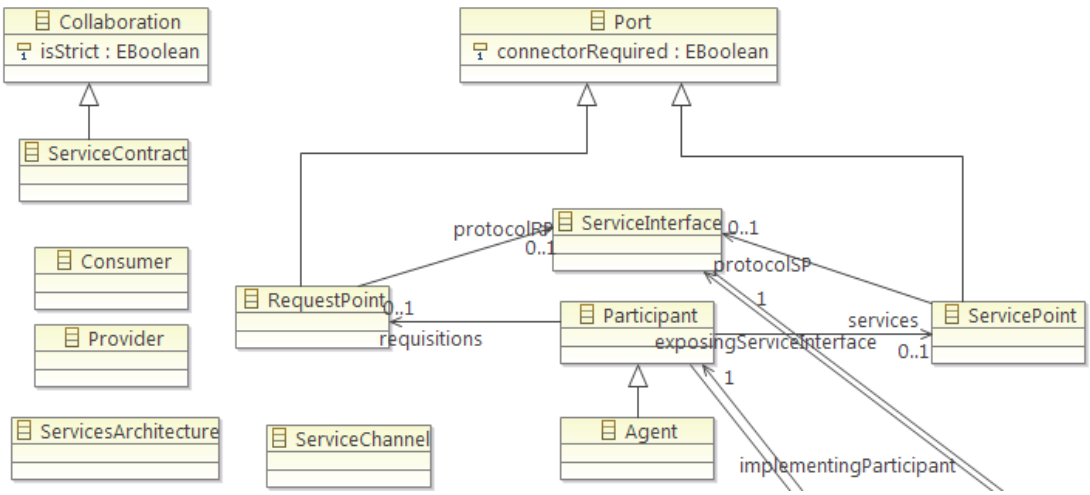


Figure 6.1: Partial SoaML Ecore implementation

The specification of SoaML briefly describes SoaML's metamodel [11]. Initially, all SoaML model elements are explained by referencing the SoaML profile. Later on in the specification, the metamodel elements are mapped to the profile elements, and thereby explains the semantics and constraints of the metamodel elements. Constraints are not described in OCL but in natural language. If we want to implement these constraints, we have to convert them to OCL, or use another technique to implement the constraints.

Our intended SoaML metamodel is similar as it is done in the work of Ali et al. [12], however, we extended it with QoS&FT modeling elements. We only implemented the modeling elements that are required to model the SoaML models as done in Section 4.5. A part of the SoaML implementation of the Ecore metamodel is shown in Figure 6.1 (the inheritance to UML metamodel elements are not shown).

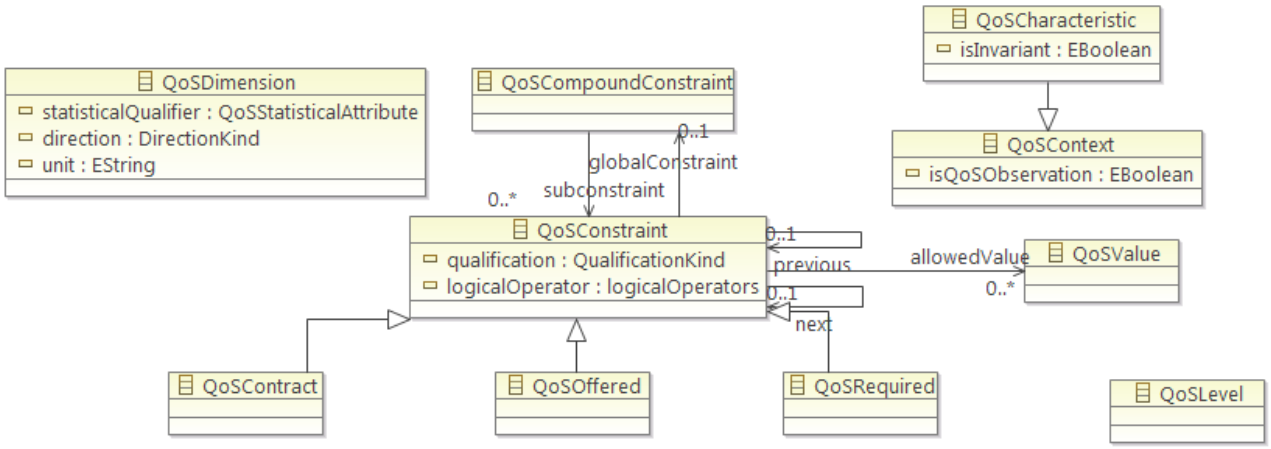


Figure 6.2: Partial QoS&FT Ecore implementation

We also implemented part of the QoS&FT specification to model the quality requirement in the same way as we have done before. With these elements the QoS requirements can be modeled and linked to the SoaML model elemenets. Figure 6.2 shows the implemented metamodel part.

6.1.2 Rebuilding the SoaML profile models

The created metamodel allows us to design metaodel-based SoaML models, include QoS specification with it, and use the models in model transformations. The SoaML models presented in Section 4.5 have been rebuilt to such metamodel-based SoaML models. Figure 6.3 shows the one-to-one correspondence of the profile-based models elements to the Ecore elements. The left side shows the profile-based model, the right side the new Ecore metamodel-based model.

There are not much differences. In fact, all stereotypes are now a model type. A subtle difference is the definition of the properties of model elements. Stereotype properties are now properties of the model element itself as shown in Figure 6.4.

The steps have also been performed for the service contract package and other packages.

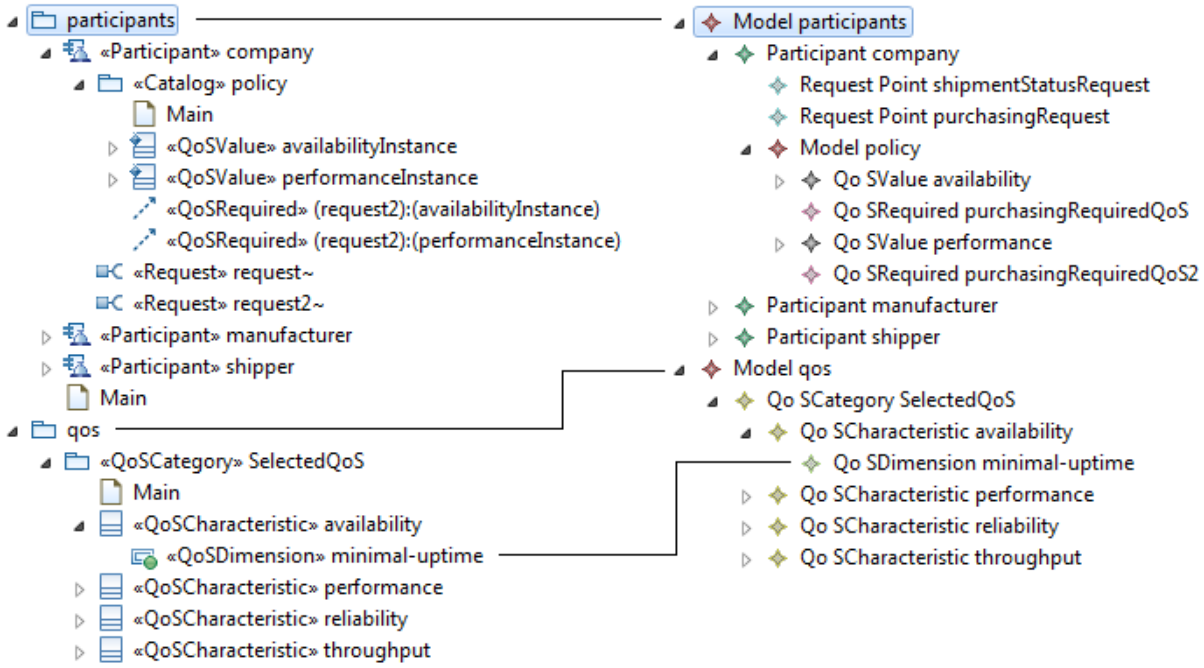


Figure 6.3: Profile-based to metamodel-based models

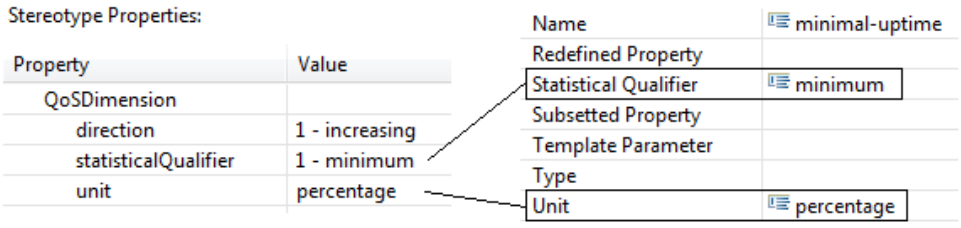


Figure 6.4: Properties of QoSDimension

6.2 Mappings

This section explains the mapping of the SoaML models to PSM-level techniques. These mappings explain which elements of the input models should be transformed to which elements of the output models. For each mapping figure in this section the following holds: the left side shows the SoaML model, the right side shows the targeted PSM model.

6.2.1 Policies

Policies are packages contained in SoaML participants. Each policy package represent a policy of a certain subject, for instance, a service request port. Therefore we map each policy package to a WS-Policy document. For convenience, some extra information can be included in the

name of the policy so that it is clear to which subject the policy belongs. An example is shown in Figure 6.5.

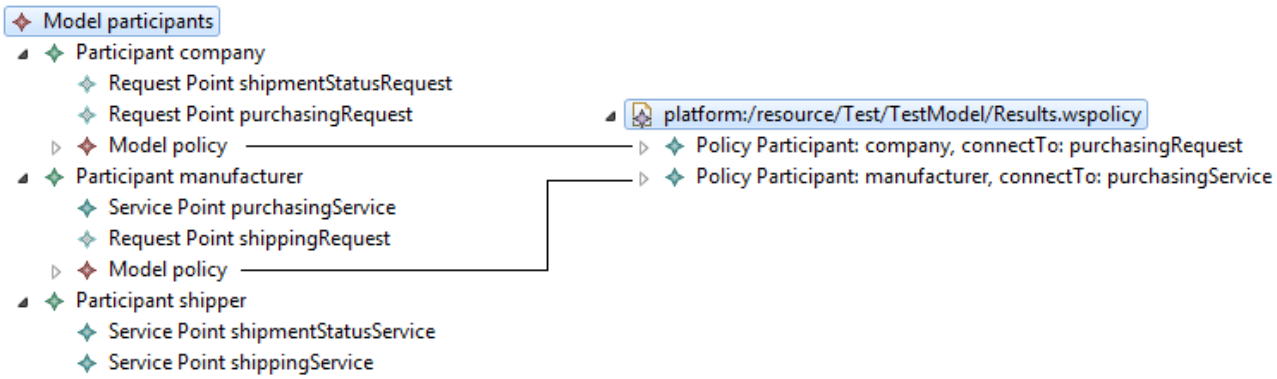


Figure 6.5: Participant policy package to WS-Policy document

In each policy package, QoS requirements are connected to the policy subject. In the SoaML models, this is done using dependencies. These dependencies have a logical operator property which can be set to "or" or "and". These value of this property is important for the structure of the WS-Policy document. In WS-Policy the structure is modeled using alternatives, i.e., with "All" and "ExactlyOne" clauses, and nesting of these elements. When all requirements are modeled as "and" dependencies, the requirements should be transformed into a WS-Policy "All" clause. I.e., all requirements must be satisfied. QoS requirements linked with "or" dependencies are transformed into an "ExactlyOne" clause. I.e., one of these requirements must be satisfied. An example is shown in Figure 6.6.

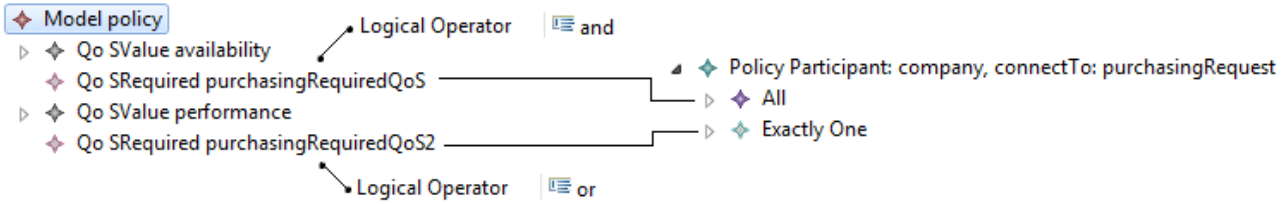


Figure 6.6: Dependency to alternatives

Nested packages are handled in a nested "All" clause. With this approach, every possible assertion combination with these operators are possible to model. Furthermore, policies that belong to another subject (e.g., another service request) should obviously be mapped to a separate WS-Policy document. Figure 6.7 shows the mapping of a nested policy, and a policy with a different subject.

All dependencies connect the subject to a certain QoS Value, which is the actual QoS requirement. Each QoS requirement is a WS-Policy assertion which we have to express with our custom grammar. Figure 6.8 shows the mapping of QoS values to QoS Aspect in our custom grammar. The actual name of the quality aspect should be the name of the QoS characteristic, not the name of the QoS value.

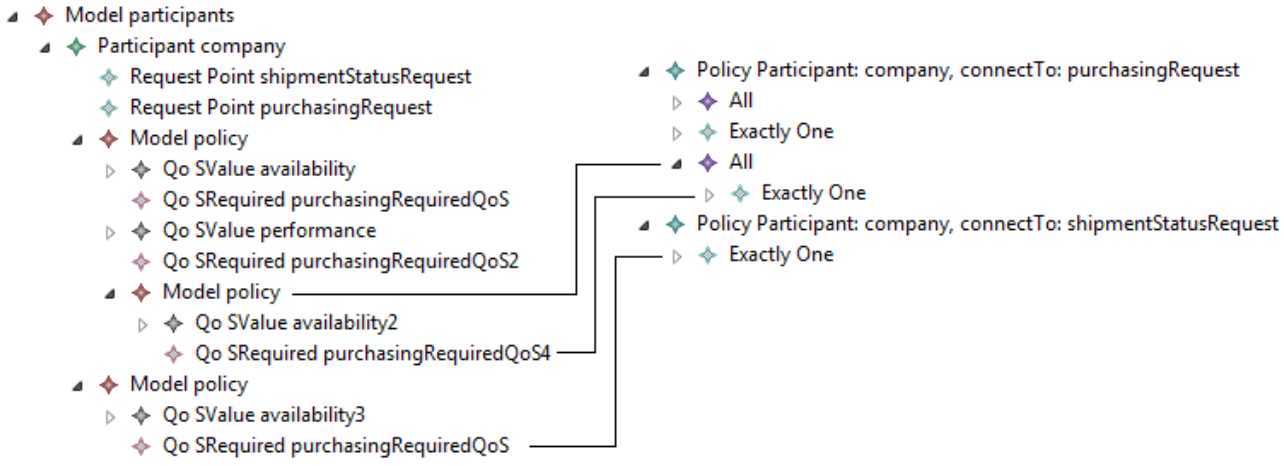


Figure 6.7: Separate policy subjects and nested policy

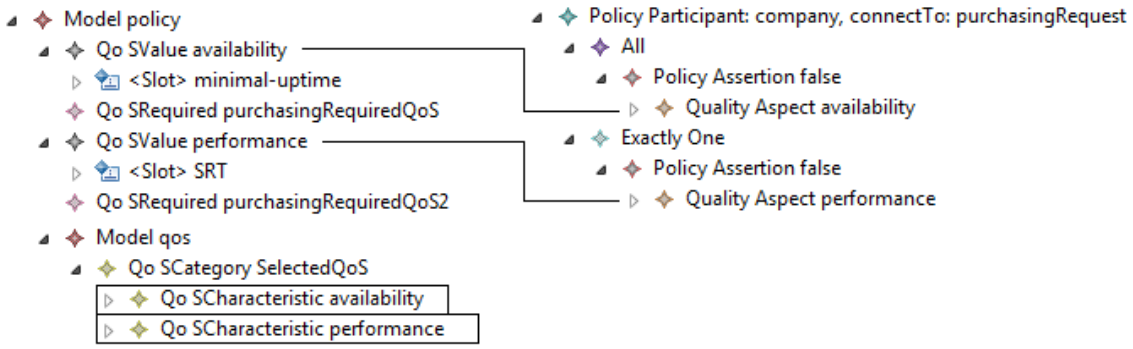


Figure 6.8: QoS values to quality aspects

The last mapping consists of mapping the actual quality requirement metric to a so-called quality attribute in the custom grammar. Also in this case, the actual name of the quality attribute should be the name of the corresponding QoS Dimension. Much of the data, such as the unit and direction, are also contained in the QoS Dimension and must therefore be mapped to the properties of the quality attribute. The value of the quality requirement is contained as a slot in the QoS Value and must be mapped to the value property of the quality attribute. Figure 6.9 shows the mapping to quality attributes and its corresponding properties that are filled with the corresponding data.

With the explained mappings, policies, in a SoaML model which is modeled properly according to our guidelines, can be transformed to WS-Policy documents. The mappings are implemented in the SoaML2WSPolicy ATL model-to-model transformation. Appendix A.1 shows the source of the implemented transformation.

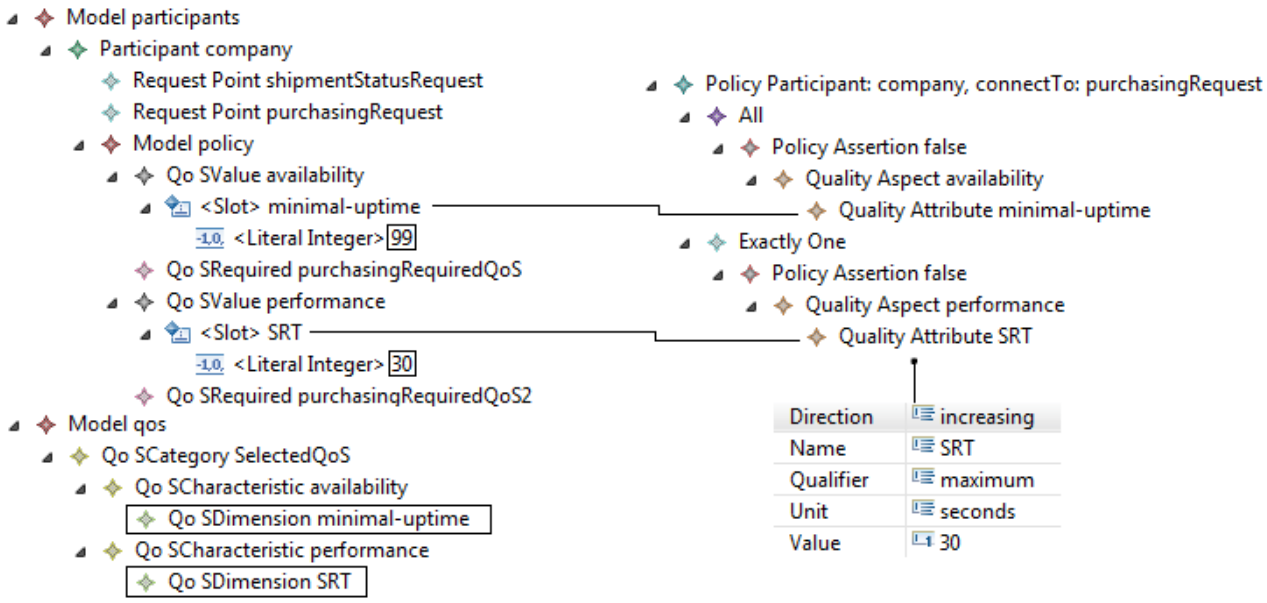


Figure 6.9: QoS value data to quality attributes

6.2.2 Service contracts

In our SoaML models, service contracts consist of a root package with the actual SoaML service contract, and a "QoS" package with the QoS requirements. We refer to this "QoS" package as the contract package. We assume each SoaML service contract is about one service and the provider and consumer. More detailed contract specifications, such as method-level contracts, or contracts involving multiple stakeholders and services are considered out of the scope. More details can be found in the future work, see Section 7.3. Each contract package is mapped to a WS-Agreement document that describes the agreement between the provider and consumer about that service.

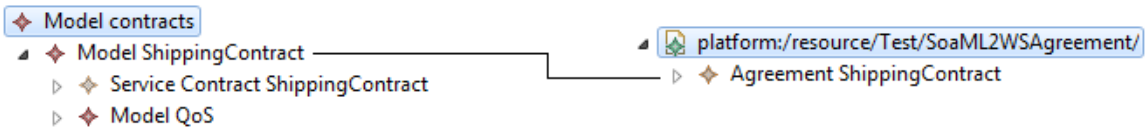


Figure 6.10: Contract specification to WS-Agreement document

The first part of a WS-Agreement document consists of the context. Figure 6.11 shows the mapping of the contract to the context. Some information is already filled in. We assume the first participant in a contract is the initiator because this is often the consumer. In the example case of Section 4.1, this means the Company is the initiator of the contract. As a consequence, the responder is the provider, which is the Shipper in this case. Other information, such as expiration time must be filled in by the stakeholders.

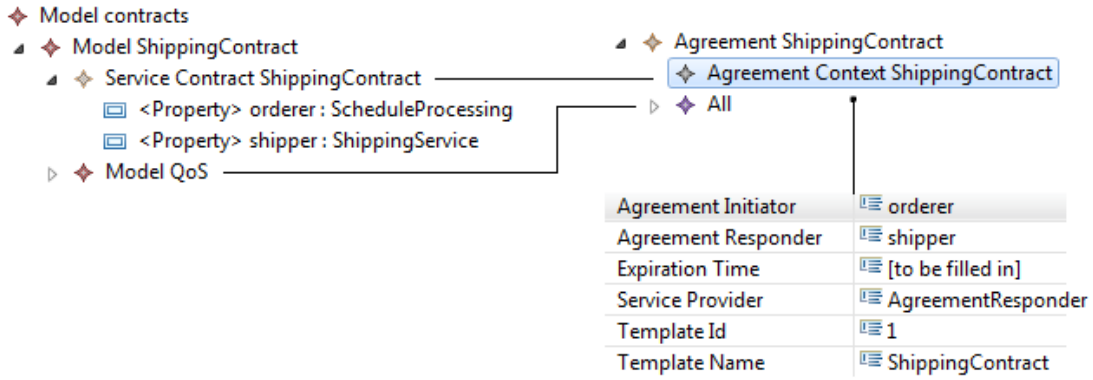


Figure 6.11: Service contract to context

Next to the context, WS-Agreement documents consist of terms, including service description terms, properties and guarantee terms. Figure 6.11 shows that all terms are mapped into a root "All" clause because all terms have to be considered. The QoS requirements specified in the SoaML models are mapped to guarantee terms which include service level objectives that are expressed with our custom grammar. This means that the guarantee terms are structured as alternatives using term compositors. Service description terms and service properties are extracted from this mapping and placed on top in the root "All" clause. We start with explaining the structuring of the guarantee terms based on the SoaML models.

In the same way as in the SoaML2Policy, QoS requirements are connected to their subjects by dependencies. In service contracts, this is done with the "QoSContract" dependency. These dependencies have a logical operator property which we can use to make requirements optional. The difference in WS-Agreement is the support for the "OneOrMore" operator, as explained in Section 5.6. In order to support this operator, we use the "next" property of the dependency element, which can link to another dependency element.

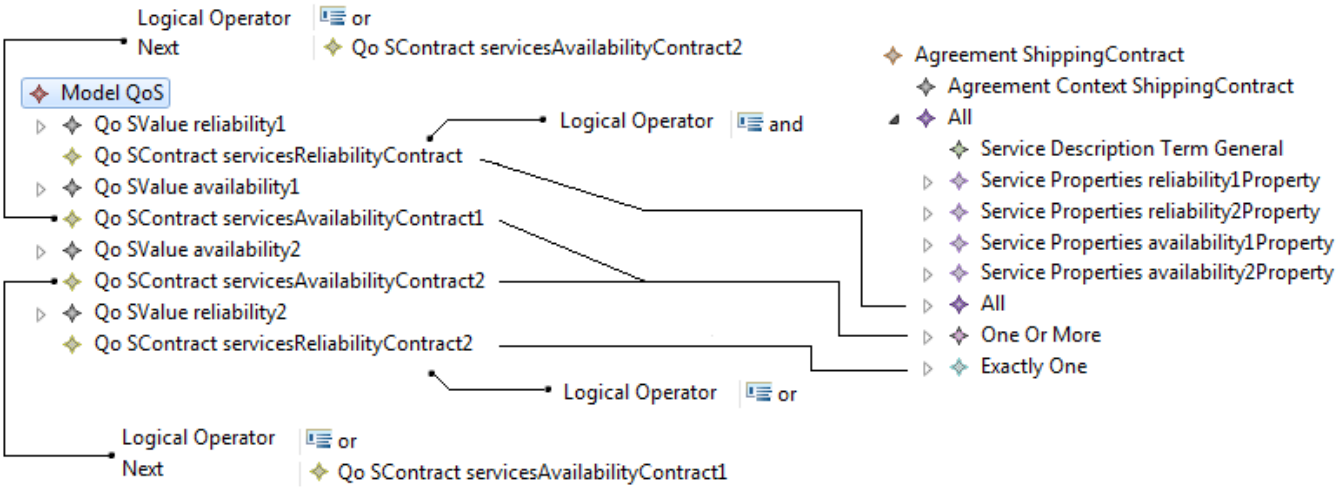


Figure 6.12: Dependency to alternatives

We defined the following guidelines to express the term compositors:

1. Map into “All” clause: all requirements where logical operator is set to “and”, or where logical operator is empty.
2. Map into “ExactlyOne” clause: all requirements where logical operator is set to “or”, and where “next” property is not set.
3. Map into “OneOrMore” clause: all requirements where logical operator is set to “or”, and where “next” property points to the next “OneOrMore” dependency.

Figure 6.12 illustrates this mapping of dependencies to alternatives using the compositors. To allow grouping of requirements and more complex combinations we support nested QoS packages. A nested QoS package will be mapped as a nested “All” clause, similarly as its done in the SoaML2WSPolicy transformation. Figure 6.13 shows the mapping of such a nested QoS package. In this example, all dependencies, reliability 1 to 3, are modeled with “or” operators.

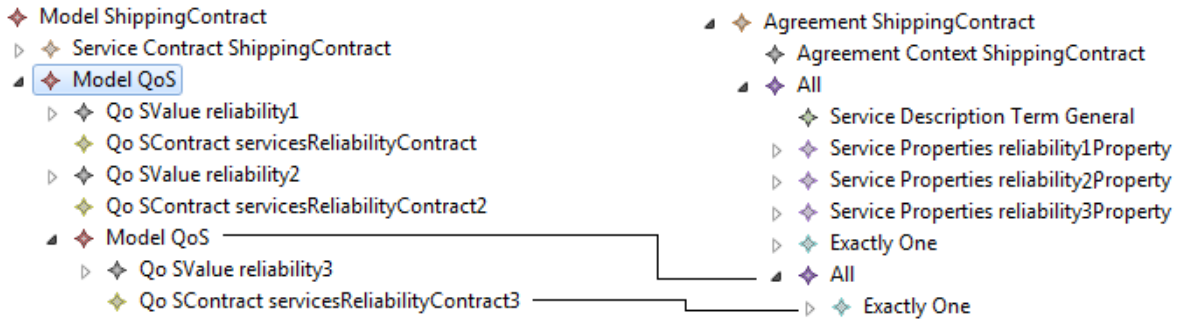


Figure 6.13: Nested QoS specifications

Figure 6.12 and 6.13 also show the service description term and service properties. One general description term is generated to describe the provided service. This term should be filled in by the stakeholders since this information is not available in the SoaML models. Furthermore service properties are generated for each QoS value. These properties define measurable and exposed properties associated with a service, such as response time and throughput. We generate properties for each value so that stakeholders are able to refine the property of a QoS requirement when this is necessary.

The last elements we have to consider are the guarantee terms themselves. These terms consist of a service scope, the service level objective and a business value list. In this mapping, the service scope is connected to the generated service description term and the business value list, with possible penalties, rewards etc., has to be filled in by the stakeholders. The service level objective is expressed with our custom grammar. We map each QoS value to a QualityAttribute element which contains all information of the QoS requirements. The QualityAttribute is automatically linked to the corresponding service property. Figure 6.14 shows the mapping of QoS values to service properties and guarantee terms.

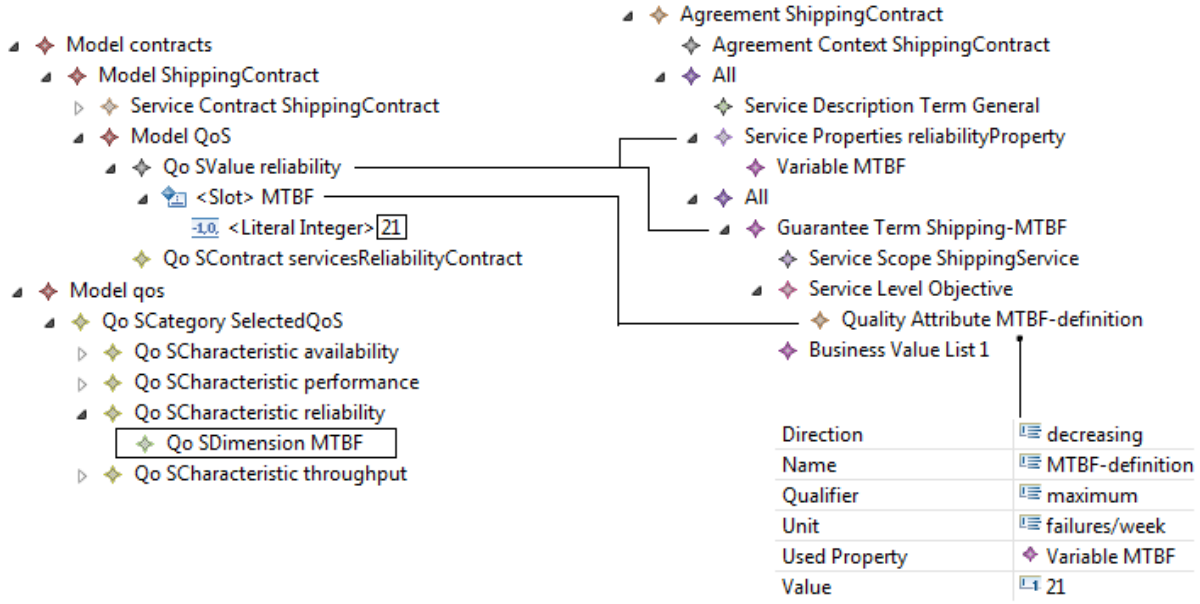


Figure 6.14: QoS value to properties and guarantee terms

With the explained mappings, service contracts, in a SoaML model modeled properly according to our guidelines, can be transformed to WS-Agreement skeleton documents. The WS-Agreement are already properly structured and can be completed by the stakeholders which are involved the service contract negotiation. The mappings are implemented in the SoaML2WSAgreement ATL model-to-model transformation. Appendix A.2 shows the source of the implemented transformation.

6.3 Transformation results

This section demonstrates the implemented transformation and shows how it can be used to optimize the stakeholder activities. We evaluate how this approach improves the activities in comparison with without specifying QoS in PIM-level models, and without automatic model transformations to answer our last research question RQ5 (see Section 1.4). We look how the transformation satisfied our expectations as explained in Section 2.10.

6.3.1 Policies

The SoaML2WSPolicy transformation (see Appendix A.1) is applied to the SoaML input models as explained in Section 6.1.2. The resulted target models (Figure 6.15) consist of two WS-Policy document which contain the QoS requirements for the corresponding participant. These policy documents could be used at the platform level to express the QoS requirements when the documents are attached to their subjects.

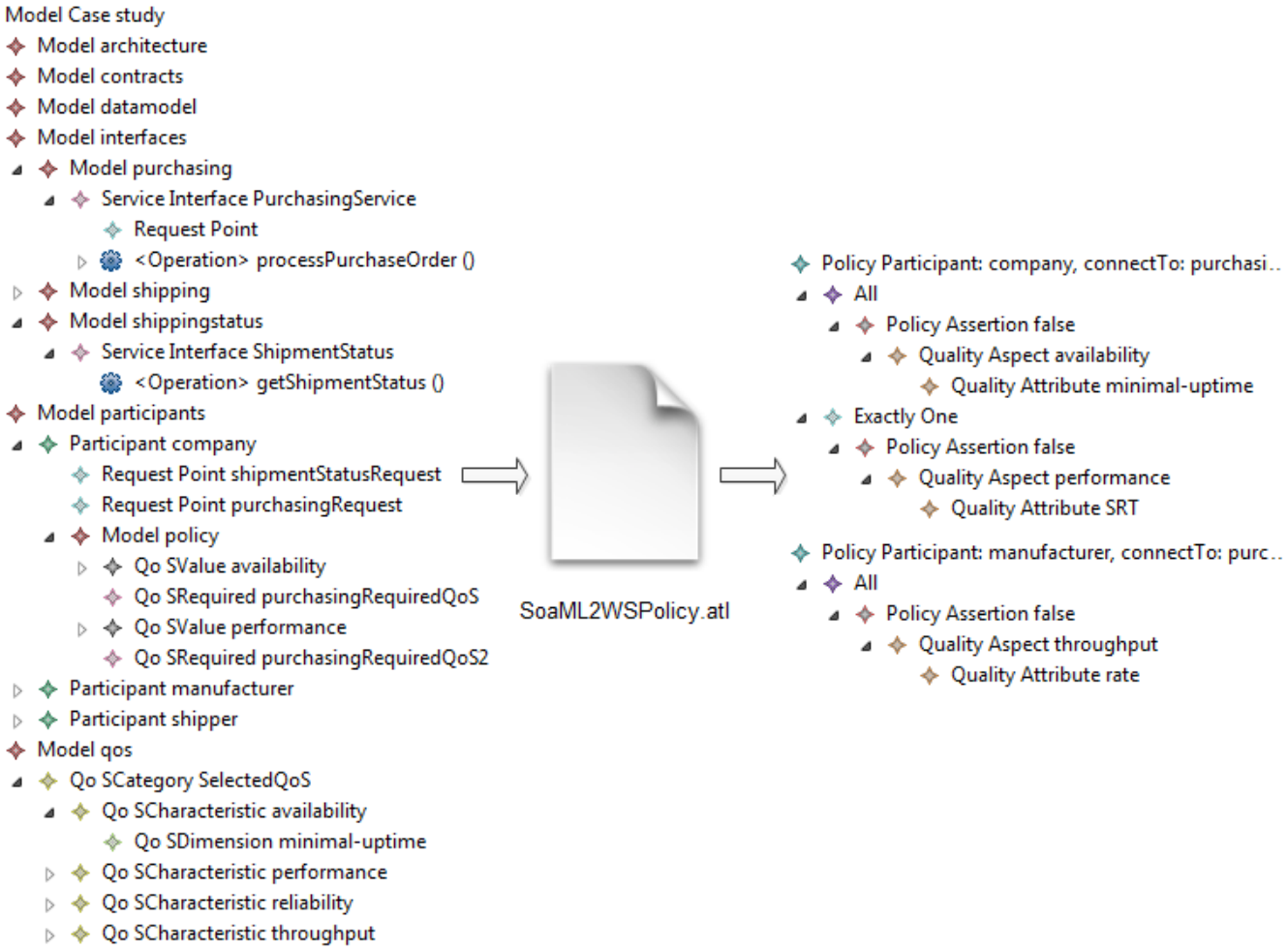


Figure 6.15: Transforming the SoaML models to WS-Policy

This demonstration shows how model transformations help to automatically generate WS-Policy PSM-level models based on information included in the SoaML PIM-level models. Referring back to Section 2.10, we see that this transformation optimizes the policy specification activity.

The service consumer, provider and application provider can benefit from this approach because they can earlier include essential model information during the service life-cycle. Furthermore the speed of development is increased because the policy documents do not have to be built from scratch. In this case, the automatic models transformations also ensure that QoS requirements do not have to be modeled twice.

6.3.2 Service contracts

Similarly, the SoaML2Agreement transformation (see Appendix A.2) is applied to the SoaML input models as explained in Section 6.1.2. The resulted target model (Figure 6.16) consist of a WS-Agreement document that can be used to instantiate a contract between the company and the shipper at the platform level.

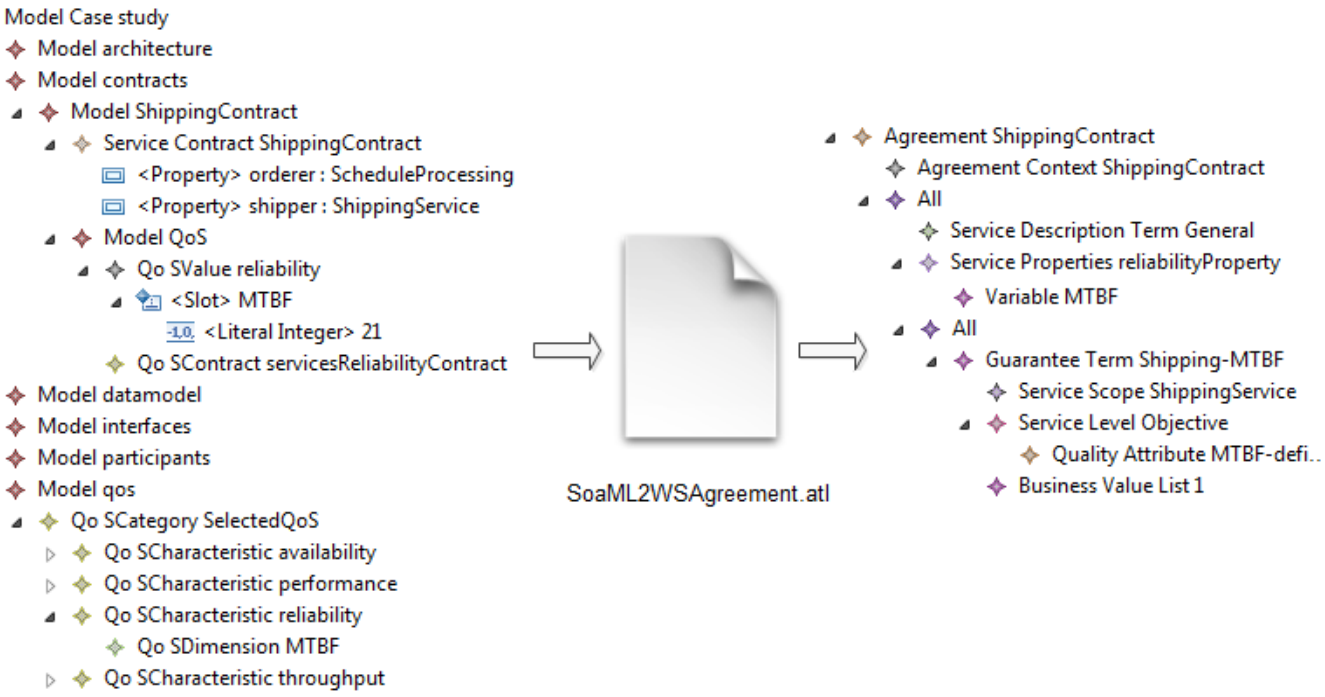


Figure 6.16: Transforming the SoaML models to WS-Agreement

This demonstration shows how automatic model transformations help to generate WS-Agreement skeleton documents from service contracts defined SoaML. Referring back to Section 2.10, we see that this transformation optimizes the service contract specification activity.

Service consumer, provider and application provider can benefit from this approach. The application provider can already model service contracts and QoS requirements at PIM-level to specify the desired SOA application. Service consumer and provider benefit because their QoS requirement are already (partly) defined in the WS-Agreement skeleton document which they can use for the negotiation. During this negotiation provider and consumer complete the WS-Agreement template and ultimately they have a complete service contract.

Chapter 7

Final remarks

In this work we investigated the transformation of SoaML PIM-level models into PSM-level models that support the specified quality requirements. This chapter discusses some related work, presents our conclusions and gives recommendations for future work.

7.1 Related work

This work is a combination of different topics, like the support of the service life-cycle, the application of MDE to SOA, and the specification of QoS requirements in SoaML and some PSM-level techniques.

Several researchers investigated which activities are involved in the service life-cycle as we have done in Chapter 2. Gu et al. [18] also consider the views of different stakeholders and their activities based on the service life-cycle. However, in our work we refine the provider into different finer-grained stakeholders because we think in practice these roles can be performed by different people. Furthermore, we try to map the stakeholders to their role in more general phases, such as design and implementation, where Gu et al. define separate phases for each stakeholder. Weinreich et al. [19] also define different activities, but they focus on the point of view from the provider at a detailed level.

There has been a lot of work done on applying MDE to SOA. Most of the work considers the functional part of SOA, e.g., transform service models into WSDL and BPEL descriptions, such as done in [95] and [96]. In our work, we consider non-functional aspects. There is also a lot of work that considers QoS specifications in their MDE processes such as [24] and [67], however, not much work has been done on the use of SoaML as the language to model SOA applications, as we do in this thesis. Some work extends SoaML for domain-specific purposes such as ambient calculus [97] and security [61]. The first one is about a functional domain-specific extension, where our work considers general non-functional aspects. The second one is more close to our work because it considers the security aspect with SoaML, however, it does not consider the transformation to the PSM-level.

Furthermore, the work of Ali et al. [12] is quite close to ours in that it applies MDA techniques to SoaML and transform the functional part of the service models to platform-specific Declarative Services (DS) models. Our research is different in that it focuses on non-functional aspects, i.e., the specification of QoS in SoaML, and the mapping to QoS modeling techniques at the PSM-level such as, e.g., WS-Policy.

7.2 Research results

Our goal was to investigate how quality requirements of service models should be specified at both PIM and PSM level and how these PIM models should be transformed to PSM models, so that productivity for the activities of stakeholders in the service life-cycle is improved. We achieved this objective by defining several guidelines and models transformation to service models. Below, we discuss the research questions and the contributions of this thesis.

Some of the stakeholders' activities in the service life-cycle (Chapter 2) can be optimized with MDE and some practical implementations reported in the literature have already shown this [95, 96]. We showed the benefits of modeling QoS at both PIM and PSM-level, and listed some common quality aspects and typical activities that could be optimized with MDE. The selected activities for this work include the specification of QoS with policies and service contracts in SOA applications (RQ1).

We observed that SoaML currently is the most complete and promising language to model PIM-level SOA applications (RQ2.1) and surveyed several QoS modeling techniques that can be used to specify the QoS requirements of stakeholders in SoaML (RQ2.2). These techniques have been evaluated with a case study to test their suitability for this project. We have chosen QoS&FT [59] to model QoS requirements because this language also allows us to interpret and use the requirements in model transformations. Furthermore, our work shows that the QoS&FT profile's tool support is limited and the profile lacks good examples of practical use. One contribution of our work are the QoS&FT examples that might help others start using this profile. We also show how to include QoS requirements (expressed with QoS&FT) in SoaML models, by modeling both policies and service contracts, and we explain the consequences of profile- versus metamodel-based models for the corresponding tools and model transformations.

We investigated several PSM-level techniques for policies and service contracts and chose the most suitable ones as target languages for the transformations (RQ3). Once we prepared our transformation environment, we defined the mappings of policies defined in SoaML to WS-Policy documents. We implemented the mapping as a ATL transformation (SoaML2WSPolicy) and showed how the transformation can be used to generate policy files. This transformation can be used to improve the process of producing policy specifications for the platform-level. Similarly, we have implemented the SoaML2WSAgreement transformation that transforms service contracts in SoaML to WS-Agreement skeleton documents (RQ4).

These automatic model transformations are beneficial in several ways for stakeholders such as the service provider, consumer and application provider. Firstly, the stakeholders are able to include model information which is essential for the functioning of the SOA application earlier in the development process. This increases consistency and understandability for other stakeholders during the evolvement of these models. Secondly, the stakeholders' productivity is increased because documents are generated automatically based on SoaML models. Productivity increase is an example of how the application of MDE to SoaML can be beneficial (RQ5).

7.3 Future work

This research's topic, applying MDE to SOA considering QoS, offers lots of directions and possibilities for further research. During this project we were able to investigate and implement just

a part of our ideas. Below, we give an overview of possible future work that can help to improve or extend this work.

Activities

In Chapter 2 of this thesis we gave an overview of several activities for various stakeholders during the service life-cycle. Because limited time was available, we have elaborated and optimized just a few of these activities in this work, i.e., specification of policies and contracts. It is interesting to investigate how that also other activities can be facilitated by the application of MDE to realize the full benefit of model-driven SOA.

Quality aspects

In this thesis we concentrated on a limited number of quality aspects (e.g., performance) and metrics (e.g., response time), as listed in Section 2.9 and Section 2.10. Other quality aspects should also be investigated to determine how to represent them at both PIM and PSM levels. For instance, modeling security might need additional notations at PIM-level to clearly specify the security requirements in a service model.

Tooling

Currently we are working with profile- and metamodel-based SoaML models because of tool limitations explained in Chapter 3. This process of developing SoaML models could be facilitated in several ways. An option would be to develop a comprehensive (metamodel-based) SoaML tool that supports visual modeling, implements the QoS support, and supports the transformation to the PSM-level. Another option would be to implement auxiliary transformations, e.g., a transformation from profile-based models to metamodel-based models, to facilitate the generation of metamodel-based models.

The SoaML and QoS Ecore metamodel implementations (see Section 6.1.1) could be extended with constraints (described in [11] and [59]). These constraints could help to describe the semantics of model elements in a systematic way, e.g., with OCL. In the end, models based on these metamodels can be validated for correctness. Furthermore, constraints can be defined to check whether the QoS is properly specified in the SoaML models, as we explained in Section 4.5 and Section 6.2.

Model transformations

The target models of our transformations (see Section 6.3) are default XMI-based models that conform to their corresponding metamodel as generated using EMF. A further improvement is to implement model-to-text (M2T) transformations that map the models to target models defined in the language's concrete syntax in order to obtain actual code.

In an ideal situation, also the functional part of the SoaML would transformed during these transformations. For example, WSDL descriptions could be generated for the service interfaces or WS-CDL descriptions could be generated for the defined choreographies [95, 96]. With respect to the policy transformation (SoaML2WSPolicy), it would then be possible to generate

attachments, which could be used to automatically attach the policies to their subjects, i.e., a service interface described in WSDL. Another option would be to generate glue code that links QoS requirements to an UDDI record, e.g., as done in [73].

At this moment, the SoaML2WSAgreement transformation is able to fill in part of the information for the WS-Agreement document, as explained in Section 5.6 and Section 6.2.2. This transformation could be extended. For example, finding a way to support the specification of business values, multi-party contracts, or creation constraints in SoaML models could be an interesting extension. For example, business values for a service contract may already be defined at the CIM-level, and could be considered in SoaML PIM-level models.

Monitoring and enforcement of QoS

In this thesis we have considered PSM techniques that express QoS aspects and requirements at PSM-level. It is possible to go a step further by considering how to enforce and monitor these requirements. It might also be possible to (automatically) generate models that support certain monitoring and enforcement platforms (such as WebSphere DataPower [77] to enforce policies), which leads to a more complete coverage of the service life-cycle.

References

[1] Fraunhaufer, "WS-Agreement Language." http://packcs-e0.scai.fraunhofer.de/wsag4j/wsag/wsag-language.html, 2012.

[2] T. Gherbi, D. Meslati, and I. Borne, "MDE between Promises and Challenges," *11th International Conference on Computer Modelling and Simulation*, 2009.

[3] Z. Stojanovic, A. Dahanayake, and H. Sol, "Modeling and Design of Service-Oriented Architecture," *2004 IEEE International Conference on Systems, Man and Cybernetics*, 2004.

[4] A. T. Zade, S. Rasulzadeh, and R. Torkashvan, "A Middleware Transparent Framework for Applying MDA to SOA," *World Academy of Science, Engineering and Technology 42*, 2008.

[5] Object Management Group, "MDA Specifications." http://www.omg.org/mda/specs.htm, 2011.

[6] Systems and Network Attack Center, "Service Oriented Architecture Security Vulnerabilities - Web Service." http://www.nsa.gov/ia/_files/factsheets/SOA_security_vulnerabilities_web.pdf, 2007.

[7] N. Bieberstein, R. G. Laird, D. K. Jones, and T. Mitra, *Executing SOA: A Practical Guide for the Service-Oriented Architect*. IBM Press, 2004.

[8] Methodologies coorporation, "Introduction to SOMF: Agile Software Modeling." http://www.modelingconcepts.com/pages/download.htm, 2011.

[9] Model Driven Solutions, "Enterprise Service Oriented Architecture Using the OMG SoaML Standard," 2009.

[10] M. Lopez-Sanz, C. J. Acuna, C. E. Cuesta, and E. Marcos, "Modelling of Service-Oriented Architectures with UML," *Electronic Notes in Theoretical Computer Science, No. 194*, 2008.

[11] Object Management Group, "Specification for the UML Profile and Metamodel for Services (UPMS)." http://www.omg.org/spec/SoaML/1.0/PDF/, 2012.

[12] N. Ali, R. Nellipaiappan, and R. Chandran, "Model Driven Support for the Service oriented architecture Modeling Language," *PESOS '10, May 2-8, 2010*, 2010.

[13] N. Schot, "Model-Driven SOA: report for research topics," 2012.

[14] M. O. Hilari, "Quality of Service (QoS) in SOA Systems. A Systematic Review." http://upcommons.upc.edu/pfc/bitstream/2099.1/7714/1/Master%20thesis%20-%20Marc%200riol.pdf, 2009.

[15] OASIS, "Reference Architecture Foundation for Service Oriented Architecture Version 1.0." http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra.html, 2011.

[16] OASIS, "Reference Model for Service Oriented Architecture 1.0." http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html, 2006.

[17] D. Ameller, X. Franch, and J. Cabot, "Dealing with Non-Functional Requirements in Model-Driven Development," 2010.

[18] Q. Gu and P. Lago, "A Stakeholder-driven Service Life Cycle Model for SOA," *IW-SOSWE 2007*, 2007.

[19] R. Weinreich, A. Wiesauer, and T. Kriechbaum, "A Service Lifecycle and Information Model for Service-Oriented Architectures," *2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*, 2009.

[20] E. Silva, J. M. Lopez, L. Ferreira Pires, and M. van Sinderen, "Defining and Prototyping a Life-cycle for Dynamic Service Composition," *Concepts and Technologies for Service Oriented Computing (ACT4SOC 2008)*, 2008.

[21] Open Group, "SOA Reference Architecture." http://www.opengroup.org/projects/soa-ref-arch/, 2011.

[22] IBM, "Web services architect: Part 1." http://www.ibm.com/developerworks/webservices/library/ws-arc1/, 2001.

[23] Z. Balfagih and M. F. Hassan, "Quality Model for Web Services from Multi-Stakeholders' Perspective," 2006.

[24] A. D'Ambrogio, "A Model-driven WSDL Extension for Describing the QoS of Web Services.," 2006.

[25] P. Bianco, G. A. Lewis, and P. Merson, "Service Level Agreements in Service-Oriented Architecture Environments," 2008.

[26] A. Al-Moayed and B. Hollunder, "Quality of Service Attributes in Web Services," *2010 Fifth International Conference on Software Engineering Advances*, 2010.

[27] S. W. Choi, J. S. Her, and S. D. Kim, "Modeling QoS Attributes and Metrics for Evaluating Services in SOA Considering Consumers Perspective as the First Class Requirement," *2007 IEEE Asia-Pacific Services Computing Conference*, 2007.

[28] A. Modelbased.net, "What is MDA?." http://www.modelbased.net/mdi/mda/mda.pdf, 2006.

[29] Object Management Group, "MDA Guide Version 1.0.1." http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf, 2003.

[30] J. A. Almeida, R. Dijkman, M. van Sinderen, and L. Ferreira Pires, "Platform-independent modelling in MDA: supporting abstract platforms," *Model Driven Architecture. European MDA Workshops: Foundations and Applications*, 2005.

[31] B. Elvesæter and A.-J. Berre, "Specifying services using the SOA modeling language (SoaML)," 2011.

[32] A.-J. Berre, D. Roman, B. Elvesester, and C. Carrez, "SoaML Tutorial @MOPAS2010, Athens, Greece," 2010.

[33] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, "ATL: A model transformation tool," 2008.

[34] Object Management Group, "Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification," 2008.

[35] A. Kleppe, J. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Professional, 2003.

[36] H. Kreger and J. Estefan, "Navigating the SOA Open Standards Landscape Around Architecture," *A paper published by The Open Group*, 2009.

[37] I. Todoran, Z. Hussain, and N. Gromov, "SOA Integration Modeling: An Evaluation of how SoaML Completes UML Modeling," 2011.

[38] R. Amir and A. Zeid, "A UML Profile for Service Oriented Architectures," 2004.

[39] A. Randak, S. Martinez, and M. Wimmer, "Extending ATL for Native UML Profile Support: An Experience Report," 2011.

[40] Open Group, "Service-Oriented Architecture Ontology." http://www.opengroup.org/projects/soa-ontology/, 2010.

[41] Y. Lemrabet, J. Touzi, D. Clin, M.Bigand, and J.-P. Bourey, "Mapping of BPMN models into UML models using SoaML profile," *8th International Conference of Modeling and Simulation - MOSIM 10*, 2010.

[42] Everware-CBDI, "Introduction to SoaML and its Place Within the CBDI-SAE Meta Model." http://everware-cbdi.com/index.php?cID=19&cType=document, 2009.

[43] G. Benguria, X. Larrucea, B. Elvesæter, T. Neple, A. Beardsmore, and M. Friess, "A Platform Independent Model for Service Oriented Architectures," *Athena project*, 2007.

[44] Everware-CBDI, "CBDI-SAE Meta Model for SOA." http://everware-cbdi.com/cbdi-sae-metamodel, 2011.

[45] L.-J. Zhang, N. Zhou, Y.-M. Chee, A. Jalaldeen, K. Ponnalagu, R. R. Sindhgatta, A. Arsanjani, and F. Bernardini, "SOMA-ME: A platform for the model-driven design of SOA solutions," *IBM Systems Journal 47*, 2008.

[46] M. Bell, *Service - Oriented Modeling Service Analysis Design and Architecture*. John Wiley & Sons, 2008.

[47] C. Dumez, A. N. sidi moh, J. Gaber, and M. Wack, "Modeling and specification of web services composition using UML-S," *4th International Conference on Next Generation Web Services Practices*, 2008.

[48] IBM, "UML 2.0 Profile for Software Services." http://www.ibm.com/developerworks/rational/library/05/419_soa/, 2005.

[49] T. Zhang, S. Ying, S. Cao, , and X. Jia, "A Modeling Framework for Service-Oriented Architecture," *Proceedings of the Sixth International Conference on Quality Software (QSIC'06)*, 2006.

[50] W3C, "Web Services Architecture." http://www.w3.org/TR/ws-arch/, 2004.

[51] M. Gebhart, M. Baumgartner, S. Oehlert, M. Blersch, and S. Abeck, "Evaluation of Service Designs Based on SoaML," 2010.

[52] Object Management Group, "SoaML Wiki - Tool support." http://www.omgwiki.org/SoaML/doku.php?id=tool_support, 2011.

[53] ModelPro, "ModelPro - Hello World tutorial." http://www.screencast.com/users/ModelDriven.org/folders/ModelPro%20Tutorials/media/4e19939c-7f07-4aa5-bb80-ea3a788217ec, 2009.

[54] IBM, "Modeling with SoaML, the Service-Oriented Architecture Modeling Language: Part 1. Service identification." http://www.ibm.com/developerworks/rational/library/09/modelingwithsoaml-1/, 2009.

[55] A. Delgado, "MINERVA: Tools Dimension / Eclipse SoaML plug-in." http://alarcos.esi.uclm.es/MINERVA/TOOLS/soamlPlugin.htm, 2010.

[56] J. Amsden, "Modeling with SoaML, the Service-Oriented Architecture Modeling Language: Part 1 to 5.." http://www.ibm.com/developerworks/rational/library/09/modelingwithsoaml-1/, 2010.

[57] IBM, "UML constraints." http://publib.boulder.ibm.com/infocenter/rsmhelp/v7r0m0/index.jsp?topic=/com.ibm.xtools.modeler.doc/topics/cconstrnt.html, 2012.

[58] ISO/IEC, "Information technology - Object Constraint Language (OCL)," 2012.

[59] Object Management Group, "UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms Specification," 2008.

[60] Rossana Abdul Carimo, "Evaluation of UML Profile for Quality of Service from the User Perspective," 2006.

[61] S. Kou, M. A. Babar, and A. Sangroya, "Modeling Security for Service Oriented Applications," *ECSA 2010*, *August 23-26*, *201*, 2010.

[62] H. Espinoza, H. Dubois, S. Gerard, J. Medina, D. C. Petriu, and M. Woodside, "Annotating UML Models with Non-Functional Properties for Quantitative Analysis," 2006.

[63] S. Frolund and J. Koistinen, "QML: A Language for Quality of Service Specification," 1998.

[64] J. I. Asensio, V. A. Villagrá, J. E. L. de Vergara, and J. J. Berrocal, "UML Profiles for the Specification and Instrumentation of QoS Management Information in Distributed Object-based Applications," 2001.

[65] J. Øyvind Aagedal and J. Earl F. Ecklund, "Modelling QoS: Towards a UML Profile," 2002.

[66] X. Gu, K. Nahrstedt, W. Yuan, D. Wichadakul, and D. Xu, "An XML-based Quality of Service Enabling Language for the Web," 2001.

[67] A. Al-Moayed and B. Hollunder, "An Approach to Model, Configure and Apply QoS Attributes to Web Services," *2011 IEEE Asia-Pacific Services Computing Conference*, 2011.

[68] Changying Dai and Zhibin Wang, "A flexible extension of WSDL to describe non-functional attributes," 2010.

[69] B. Hollunder, "WS-Policy: On Conditional and Custom Assertions," 2009.

[70] F.-Z. Belouadha, H. Omrana, and O. Roudies, "A MDA approach for defining WS-Policy semantic non-functional properties," 2010.

[71] W3C, "Web Services Policy 1.5 - Framework." http://www.w3.org/TR/ws-policy/, 2007.

[72] W3C, "Web Services Policy 1.5 - Attachment." http://www.w3.org/Submission/WS-PolicyAttachment/, 2007.

[73] S. Chaari, Y. Badr, and F. Biennier, "Enhancing Web Service Selection by QoS-Based Ontology and WS-Policy," *SAC 08*, 2008.

[74] T. Nurmela, "WS-Policy specifications," 2005.

[75] OASIS, "WS-SecurityPolicy 1.2." http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.2/ws-securitypolicy.pdf, 2007.

[76] OASIS, "Web Services Reliable Messaging (WS-ReliableMessaging) 1.2." http://docs.oasis-open.org/ws-rx/wsrm/v1.2/wsrm.pdf, 2009.

[77] IBM, "WS-MeditationPolicy 1.6." ftp://public.dhe.ibm.com/software/solutions/soa/pdfs/WSMediationPolicy1.6-20120124.pdf, 2012.

[78] Diego Zuquim Guimarães Garcia and Maria Beatriz Felgar de Toledo, "A Web Service Architecture Providing QoS Management," 2006.

[79] C.-C. Lo, D.-Y. Cheng, P.-C. Lin, and K.-M. Chao, "A Study on Representation of QoS in UDDI for Web Services Composition," *International Conference on Complex, Intelligent and Software Intensive Systems*, 2008.

[80] S. Microsystems and A. H. Anderson, "An Introduction to the Web Services Policy Language (WSPL)," 2004.

[81] Glen Daniels, "Comparing Features / Properties and WS-Policy." http://www.w3.org/2004/08/ws-cc/gdfpwsp-20040904, 2004.

[82] B. Parsia, V. Kolovski, and J. Hendler, "Expressing WS Policies in OWL," 2005.

[83] Reto Kaiser, Universität Basel, "Web Services Quality of Service," 2010.

[84] Open Grid Forum (OGF), "Web Services Agreement Specification (WS-Agreement)," 2007.

[85] Wolfgang Ziegler, "WS-Agreement," 2011.

[86] Ganna Frankova and Daniela Malfatti and and Marco Aiello, "Semantics and Extensions of WS-Agreement," 2006.

[87] IBM, "Web Service Level Agreement (WSLA) Language Specification 1.0," 2003.

[88] Keven T. Kearney and Francesco Torelli and Constantinos Kotsokalis, "SLA*: An Abstract Syntax for Service Level Agreements," 2010.

[89] Peter Chronz and Philipp Wieder, "Integrating WS-Agreement with a Framework for Service-Oriented Infrastructures," 2010.

[90] Optimus, "Analysis of Existing solutions and Requirements for SLAs," 2009.

[91] D.Davide Lamanna and James Skene and Wolfgang Emmerich, "SLAng: A Language for DefiningService Level Agreements," 2003.

[92] Adrian Paschke, "RBSLA A declarative Rule-based Service Level Agreement Language based on RuleML," 2005.

[93] Vladimir Kolovski and Bijan Parsia and Yarden Katz and and James Hendler, "Representing Web Service Policies in OWL-DL," 2005.

[94] Mi Abuela Acierta Más, "Explaining Inconsistencies of WS-Ag docs." https://miabuelaaciertamas.googlecode.com/files/Secci%C3%B3n%20de%20WS-Ag.pdf, 2010.

[95] Mardiana and Keijiro Araki and Yoichi Omori, "MDA and SOA approach to Development of Web Application Interface," 2011.

[96] Christian Emig and Karsten Krutz and Stefan Link, "Model-Driven Development of SOA Services," 2007.

[97] N. Ali and M. A. Babar, "Modeling Service Oriented Architectures of Mobile Application by Extending SoaML with Ambients," 2009.

Appendix A

ATL transformations

This appendix contains the source of the ATL [33] model-to-model transformations.

A.1 SoaML2WSPolicy

Listing A.1: SoaML to WS-Policy ATL model-to-model transformation

```
1
2  ------------------------------------------------------------------------------
3  -- Transformation file to generate several WS-Policy documents from policies specified in SoaML models.
4  -- v.0.1
5  ------------------------------------------------------------------------------
6
7  module SoaML2WSPolicy;
8  create OUT : WSPolicy from IN : SOAML;
9
10 ----------------
11 -- Helpers
12 ----------------
13
14 -- Helper to determine if a QoS constraint uses an 'and' operator
15 helper context SOAML!QoSConstraint def: usesAndOperator : Boolean =
16     if (self.logicalOperator.toString() = 'and' or self.logicalOperator.toString() = '') then
17         true
18     else
19         false
20     endif;
21
22 -- Helper to convert (IN!name)-kind to (name)-kind strings
23 helper context String def: convertToId : String =
24         self.substring(4, self.size());
25
26 -- Helper to get ID of the corresponding assertion referenced by dependency
27 helper context SOAML!QoSConstraint def: getAssertionName : String =
28         self.supplier.first().toString().convertToId;
29
30 -- Helper to get ID of the corresponding QoS characteristic referenced by QoS value definition
31 helper context SOAML!QoSValue def: getCharacteristicName : String =
32         self.classifier.first().toString().convertToId;
33
34 -- Helper to check if object is a QoS dependency
35 helper context OclAny def : isDependency : Boolean = (self.oclIsTypeOf(SOAML!QoSRequired) or self.oclIsTypeOf(
       SOAML!QoSOffered));
36
37 -- Helper to check if object is a policy
38 helper context OclAny def : isPolicy : Boolean = (self.oclIsTypeOf(SOAML!SoaMLModel) and self.name='policy');
39
40 -- Helper that maps source statistical qualifiers to assertion qualifiers
41 helper def : qualifierMap : Map(OclAny,OclAny) = Map{(#maximum,#maximum),(#minimum,#minimum),(#range,
       OclUndefined)};
42
43 -- Helper that maps source directions to assertion directions
```

```
44  helper def : directionMap : Map(OclAny,OclAny) = Map{(#increasing,#increasing),(#decreasing,#decreasing)};
45
46
47  ----------------
48  -- Starting rule
49  ----------------
50
51  -- Transform each policy package in a participant to WS-Policy document
52
53  rule Participant2Policy {
54      from
55          policy : SOAML!Participant ( policy.packagedElement->exists( d | d.isPolicy) )
56      do {
57          policy.packagedElement->select( p | p.name='policy')->collect( c | thisModule.Policy2Policy(c) );
58      }
59
60  }
61
62
63  ----------------
64  -- Lazy rules
65  ----------------
66
67  -- Transform each policy package to a WS-Policy document
68
69  lazy rule Policy2Policy {
70      from
71          policy : SOAML!SoaMLModel
72          using {
73              policyName : String = policy.refImmediateComposite().name;
74              policyPackagesElements : OclAny = policy.packagedElement;
75              qoSDependencies : OclAny = policyPackagesElements->select( b | b.isDependency );
76              nestedPolicies : OclAny = policyPackagesElements->select( b | b.isPolicy );
77              requiredAssertions : OclAny = qoSDependencies->select( c | c.usesAndOperator );
78              optionalAssertions : OclAny = qoSDependencies->select( d | not d.usesAndOperator );
79              dependency : OclAny = if not qoSDependencies.isEmpty() then qoSDependencies.first().client.first()
                    else OclUndefined endif;
80          }
81      to
82          policydoc : WSPolicy!Policy
83          (
84              -- Give policy name of the corresponding participant
85              name <- 'Participant: '.concat(policyName).concat(', connectTo: ').concat( dependency.toString().
                    convertToId ),
86              -- Transform all QoS dependencies to alternative elements
87              alternatives <- policyPackagesElements->collect( e |
88                  -- Transform assertions that must be satisfied in 'All' clause
89                  if (requiredAssertions.first() = e) then
90                     thisModule.LogicalOperatorAll2Alternative(e)
91                  else
92                      -- Transform all assertions of which at least one must be satisfied to 'ExactlyOne' clause
93                      if(optionalAssertions.first() = e) then
94                          thisModule.LogicalOperatorOr2Alternative(e)
95                      else
96                          -- If there are nested policies
97                          if( e.oclIsTypeOf(SOAML!SoaMLModel) and e.name='policy' ) then
98                                  thisModule.NestedPolicy2Policy(e)
99                          else OclUndefined endif
100                     endif
101                 endif
102             )
103         )
104 }
105
106
```

```
107 -- Transform each nested policy package to an All clause inside root clause
108
109 lazy rule NestedPolicy2Policy {
110     from
111         policy : SOAML!SoaMLModel
112         using {
113             qoSDependencies : OclAny = policy.packagedElement->select( b | b.isDependency );
114             requiredAssertions : OclAny = qoSDependencies->select( c | c.usesAndOperator );
115             optionalAssertions : OclAny = qoSDependencies->select( d | not d.usesAndOperator );
116             dependency : OclAny = qoSDependencies.first().client.first();
117         }
118     to
119         allclause : WSPolicy!All
120         (
121             -- Transform all QoS dependencies to alternative elements
122             nestedAlternative <- qoSDependencies->collect( e |
123                 -- Transform assertions that must be satisfied in 'All' clause
124                 if (requiredAssertions.first() = e) then
125                    thisModule.LogicalOperatorAll2Alternative(e)
126                 else
127                     -- Transform all assertions of which at least one must be satisfied to 'ExactlyOne' clause
128                     if(optionalAssertions.first()=e) then
129                         thisModule.LogicalOperatorOr2Alternative(e)
130                     else
131                         -- If there are nested policies
132                         if( policy.packagedElement->exists( d | d.oclIsTypeOf(SOAML!SoaMLModel) and d.name='
                                policy' ) ) then
133                             policy.packagedElement->collect( g |
134                                 if( g.oclIsTypeOf(SOAML!SoaMLModel) and g.name='policy' ) then
135                                    thisModule.NestedPolicy2Policy(g)
136                                 else OclUndefined endif
137                             )
138
139                         else OclUndefined endif
140                     endif
141                 endif
142             )
143         )
144 }
145
146
147 -- Creates an 'All' clause for corresponding QoS constraints
148
149 lazy rule LogicalOperatorAll2Alternative {
150     from
151         -- Transform the QoS policy of the provider or consumer
152         qosconstraint : SOAML!QoSConstraint
153     to
154         andoperator : WSPolicy!All (
155             assertions <- SOAML!QoSValue->allInstances()->collect( a |
156                -- Transform assertions that belong in this clause
157                if a.name=qosconstraint.getAssertionName then
158                   thisModule.QoSValue2PolicyAssertion(a)
159                else
160                     -- If another constraint is in the same package as reference constraint, also include his
                          referenced assertion
161                     if ( qosconstraint.refImmediateComposite().packagedElement->exists( d | d.isDependency ) )
                          then
162                         SOAML!QoSConstraint->allInstances()->collect( b |
163                             -- If constraint uses 'AND' operator and has the same ID
164                             if (b.usesAndOperator) and ( qosconstraint.refImmediateComposite().packagedElement
                                  ->one( d | d.name=b.name ) ) then
165                                -- Check the assertions with current assertion, if OK; include this assertion
                                      in same clause
```

```
166                                 if(qosconstraint.reflmmediateComposite().packagedElement->any( d | d.name=b.
                                        name ).getAssertionName = a.name ) then
167                                     thisModule.QoSValue2PolicyAssertion(a)
168                                 else OclUndefined endif
169                             else OclUndefined endif
170                         )
171                     else OclUndefined endif
172                 endif
173             )
174         )
175 }
176
177
178 -- Creates an 'ExactlyOne' clause for corresponding QoS constraints
179
180 lazy rule LogicalOperatorOr2Alternative {
181     from
182         -- Transform the QoS policy of the provider or consumer
183         qosconstraint : SOAML!QoSConstraint
184     to
185         oroperator : WSPolicy!ExactlyOne (
186             assertions <- SOAML!QoSValue->allInstances()->collect( a |
187                 -- Transform assertions that belong in this clause
188                 if a.name=qosconstraint.getAssertionName then
189                     thisModule.QoSValue2PolicyAssertion(a)
190                 else
191                     -- If another constraint is in the same package as reference constraint, also include his
                            referenced assertion
192                     if (qosconstraint.reflmmediateComposite().packagedElement->exists( d | d.isDependency ) )
                            then
193                         SOAML!QoSConstraint->allInstances()->collect(b |
194                             -- If constraint uses 'ExactlyOne' operator and has the same ID
195                             if (not b.usesAndOperator) and ( qosconstraint.reflmmediateComposite().
                                    packagedElement->one( d | d.name=b.name ) ) then
196                                 -- Check the assertions with current assertion, if OK; include this assertion
                                        in same clause
197                                 if( qosconstraint.reflmmediateComposite().packagedElement->any( d | d.name=b.
                                        name ).getAssertionName = a.name ) then
198                                     thisModule.QoSValue2PolicyAssertion(a)
199                                 else OclUndefined endif
200                             else OclUndefined endif
201                         )
202                     else OclUndefined endif
203                 endif
204             )
205         )
206 }
207
208
209 -- Maps each QoS Value name (instance of the QoS characteristic) to an assertion, and then continues with
        aspect mapping
210
211 lazy rule QoSValue2PolicyAssertion {
212     from
213         soamlQoSValue : SOAML!QoSValue
214     to
215         qualityAspect: WSPolicy!PolicyAssertion (
216             assertions <- thisModule.QoSValue2QualityAspect(soamlQoSValue)
217         )
218
219  }
220
221
222 ---------------
223 -- Custom grammar rules
```

```
224 ||  ----------------
225 ||
226 ||  -- Maps each QoS Value name (instance of the QoS characteristic) to quality aspect (QualityAspect elements in
    ||       our custom grammar))
227 ||
228 ||  lazy rule QoSValue2QualityAspect {
229 ||     from
230 ||          soamlQoSValue : SOAML!QoSValue
231 ||     to
232 ||          qualityAspect: WSPolicy!QualityAspect (
233 ||              name <- soamlQoSValue.classifier.first().name,
234 ||              -- Iterate over all slots
235 ||              attributes <- soamlQoSValue.slot->collect( a | thisModule.Slot2QualityAttribute(a) )
236 ||          )
237 ||
238 ||   }
239 ||
240 ||
241 ||  -- Maps each slot to an assertion (QualityAttribute elements in our custom grammar)
242 ||
243 ||  lazy rule Slot2QualityAttribute {
244 ||     from
245 ||          soamlSlot : SOAML!Slot
246 ||     to
247 ||          qualityAttribute : WSPolicy!QualityAttribute (
248 ||              name <- soamlSlot.definingFeature.name,
249 ||              value <- soamlSlot.value.first().value,
250 ||              unit <- soamlSlot.definingFeature.unit,
251 ||              direction <- thisModule.directionMap.get(soamlSlot.definingFeature.direction),
252 ||              qualifier <- thisModule.qualifierMap.get(soamlSlot.definingFeature.statisticalQualifier)
253 ||          )
254 ||  }
```

In this transformation element names are used to identify elements. When using this transformation, do not forget to uniquely name elements inside a policy package.

A.2 SoaML2WSAgreement

Listing A.2: SoaML to WS-Agreement ATL model-to-model transformation

```
1  ||
2  ||  ------------------------------------------------------------------------------
3  ||  -- Transformation file to generate WS-Agreement skeleton documents from contracts specified in SoaML models.
4  ||  -- v.0.1
5  ||  ------------------------------------------------------------------------------
6  ||
7  ||  module SoaML2WSAgreement;
8  ||  create OUT : WSAgreement from IN : SOAML;
9  ||
10 ||  ----------------
11 ||  -- Helpers
12 ||  ----------------
13 ||
14 ||  -- Helper to determine if a QoS constraint uses an 'and' operator (to WS-Agreement 'All' clause)
15 ||  helper context SOAML!QoSConstraint def: usesAndOperator : Boolean =
16 ||      if (self.logicalOperator.toString() = 'and' or self.logicalOperator.toString() = '') then
17 ||          true
18 ||      else
19 ||          false
20 ||      endif;
21 ||
```

```
22 -- Helper to determine if a QoS constraint uses an 'or' operator and has not a next property (to WS-Agreement '
       ExactlyOne' clause)
23 helper context SOAML!QoSConstraint def: usesOrOperator : Boolean =
24     if (self.logicalOperator.toString() = 'or' and self.next.oclIsUndefined()) then
25         true
26     else
27         false
28     endif;
29
30 -- Helper to determine if a QoS constraint uses an 'or' operator and has a next property (to WS-Agreement '
       OneOrMore' clause)
31 helper context SOAML!QoSConstraint def: usesOrOperatorAndNext : Boolean =
32     if (self.logicalOperator.toString() = 'or' and not self.next.oclIsUndefined()) then
33         true
34     else
35         false
36     endif;
37
38 -- Helper to convert (IN!name)-kind to (name)-kind strings
39 helper context String def: convertToId : String =
40         self.substring(4, self.size());
41
42 -- Helper to get ID of the corresponding assertion referenced by dependency
43 helper context SOAML!QoSConstraint def: getAssertionName : String =
44         self.supplier.first().toString().convertToId;
45
46 -- Helper to get ID of the corresponding QoS characteristic referenced by QoS value definition
47 helper context SOAML!QoSValue def: getCharacteristicName : String =
48         self.classifier.first().toString().convertToId;
49
50 -- Helper to check if object is a QoS dependency
51 helper context OclAny def : isDependency : Boolean = (self.oclIsTypeOf(SOAML!QoSContract));
52
53 -- Helper to check if object is a slot
54 helper context OclAny def : isSlot : Boolean = (self.oclIsTypeOf(SOAML!Slot));
55
56 -- Helper to check if object is a QoS specification in contract package
57 helper context OclAny def : isQoSSpec : Boolean = (self.oclIsTypeOf(SOAML!SoaMLModel) and self.name='QoS');
58
59 -- Helper to check if object is a QoS specification in contract package
60 helper context OclAny def : isQoSValue : Boolean = (self.oclIsTypeOf(SOAML!QoSValue));
61
62 -- Helper to check if object is the ServiceContract in contract package
63 helper context OclAny def : isServiceContract : Boolean = (self.oclIsTypeOf(SOAML!ServiceContract));
64
65 -- Helper that maps source statistical qualifiers to assertion qualifiers
66 helper def : qualifierMap : Map(OclAny,OclAny) = Map{(#maximum,#maximum),(#minimum,#minimum),(#range,
       OclUndefined)};
67
68 -- Helper that maps source directions to assertion directions
69 helper def : directionMap : Map(OclAny,OclAny) = Map{(#increasing,#increasing),(#decreasing,#decreasing)};
70
71 -- Template counter
72 helper def: counter : Integer = 1;
73
74 -- Contract subject
75 helper def: contractSubject : String = 'subject';
76
77 -- Root QoS specification
78 helper def: rootQoSSpec : SOAML!SoaMLModel = OclUndefined;
79
80
81 ----------------
82 -- Starting rule
83 ----------------
```

```
84
85  -- Transform each QoS package in a service contract to a WS-Agreement document
86
87  rule ServiceContract2Agreement {
88      from
89          qosspec : SOAML!ServiceContract ( qosspec.reflmmediateComposite().packagedElement->exists( d | d.
                isQoSSpec ) )
90      do {
91          -- Remember what the root QoS specification is
92          thisModule.rootQoSSpec <- qosspec.reflmmediateComposite().packagedElement->any( p | p.name='QoS' );
93          -- Remember contract name
94          thisModule.contractSubject <- qosspec.name.substring(1,qosspec.name.size()-8);
95          -- Call first rule that transform QoS package into agreement
96          thisModule.QoSPackage2Agreement(thisModule.rootQoSSpec);
97          -- Increment template counter
98          thisModule.counter <- thisModule.counter + 1;
99      }
100 }
101
102
103 ---------------
104 -- Lazy rules
105 ---------------
106
107 -- Transform each QoS package to a WS-Agreement document
108
109 lazy rule QoSPackage2Agreement {
110     from
111         qosspec : SOAML!SoaMLModel
112         using {
113             contractName : String = qosspec.reflmmediateComposite().name;
114         }
115     to
116         qualityAspect : WSAgreement!Agreement
117         (
118             -- Give contract of the service contract
119             name <- contractName,
120
121             context <- thisModule.QoSPackage2Context(qosspec),
122             terms <- thisModule.QoSPackage2RootAll(qosspec)
123         )
124 }
125
126
127 -- Transforms QoS Package to an agreement context
128
129 lazy rule QoSPackage2Context {
130     from
131         qosspec : SOAML!SoaMLModel
132     to
133         agreementcontext : WSAgreement!AgreementContext
134         (
135             expirationTime <- '[to be filled in]',
136             templateName <- qosspec.reflmmediateComposite().name,
137             templateId <- thisModule.counter.toString(),
138             -- First role is considered as initiator
139             agreementInitiator <- qosspec.reflmmediateComposite().packagedElement->any( a | a.isServiceContract
                    ).ownedAttribute.first().name,
140             -- Second role is considered as responder
141             agreementResponder <-  qosspec.reflmmediateComposite().packagedElement->any( a | a.
                    isServiceContract ).ownedAttribute.at(2).name,
142             serviceProvider <- #AgreementResponder
143         )
144 }
145
```

```
146
147 -- Transforms QoS to root All clause
148
149 lazy rule QoSPackage2RootAll {
150     from
151         qosspec : SOAML!SoaMLModel
152         using {
153             qoSDependencies : OclAny = qosspec.packagedElement->select( b | b.isDependency );
154             requiredAssertions : OclAny = qoSDependencies->select( c | c.usesAndOperator );
155             oneAssertions : OclAny = qoSDependencies->select( d | d.usesOrOperator );
156             oneOrMoreAssertions : OclAny = qoSDependencies->select( d | d.usesOrOperatorAndNext );
157             dependency : OclAny = qoSDependencies.first().client.first();
158         }
159     to
160         rootclause : WSAgreement!All
161         (
162             -- Generate service description from contract
163             ownedTerms <- thisModule.QoSPackage2ServiceDescriptionTerm(thisModule.rootQoSSpec),
164
165             -- Transform all QoS Values to service properties
166             ownedTerms <- thisModule.rootQoSSpec.packagedElement->collect( a |
167                 if (a.isQoSValue) then
168                     thisModule.QoSValue2ServiceProperties(a)
169                 else OclUndefined endif
170             ),
171
172             -- Also transform possible nested service properties and place them in root
173             ownedTerms <- if(thisModule.rootQoSSpec.packagedElement->one( z | z.oclIsTypeOf(SOAML!SoaMLModel)
                    and z.name='QoS' ) ) then
174             thisModule.rootQoSSpec.packagedElement->flatten()->select( z | z.oclIsTypeOf(SOAML!SoaMLModel) and
                    z.name='QoS' )->collect( a | a.packagedElement->collect( b |
175                 if (b.isQoSValue) then
176                     thisModule.QoSValue2ServiceProperties(b)
177                 else OclUndefined endif
178             ) )
179             else OclUndefined endif,
180
181
182             -- Transform slots to guarantee terms
183             ownedTerms <- qosspec.packagedElement->collect( e |
184                 -- Transform assertions that must be satisfied in 'All' clause
185                 if (requiredAssertions.first() = e) then
186                     thisModule.LogicalOperatorAll2Alternative(e)
187                 else
188                     -- Transform all assertions of which at least one must be satisfied to 'OneOrMore' clause
189                     if(oneOrMoreAssertions.first() = e) then
190                         thisModule.LogicalOperatorOrAndNext2Alternative(e)
191                     else
192                         -- Transform all assertions of which one must be satisfied to 'ExactlyOne' clause
193                         if(oneAssertions.first() = e) then
194                             thisModule.LogicalOperatorOr2Alternative(e)
195                         else
196                             -- If there are nested QoS specifications
197                             if( e.oclIsTypeOf(SOAML!SoaMLModel) and e.name='QoS' ) then
198                                 thisModule.NestedQoSPackage2All(e)
199                             else OclUndefined endif
200                         endif
201                     endif
202                 endif
203             )
204
205         )
206 }
207
208
```

```
209 -- Transform each nested QoS package to an 'All' clause inside root clause
210
211 lazy rule NestedQoSPackage2All {
212     from
213         qosspec : SOAML!SoaMLModel
214         using {
215             qoSDependencies : OclAny = qosspec.packagedElement->select( b | b.isDependency );
216             requiredAssertions : OclAny = qoSDependencies->select( c | c.usesAndOperator );
217             oneAssertions : OclAny = qoSDependencies->select( d | d.usesOrOperator );
218             oneOrMoreAssertions : OclAny = qoSDependencies->select( a | a.usesOrOperatorAndNext );
219             dependency : OclAny = qoSDependencies.first().client.first();
220         }
221     to
222         allclause : WSAgreement!All
223         (
224             -- Transform all QoS dependencies to alternative elements
225             ownedTerms <- qoSDependencies->collect( e |
226                 -- Transform assertions that must be satisfied in 'All' clause
227                 if (requiredAssertions.first() = e) then
228                    thisModule.LogicalOperatorAll2Alternative(e)
229                 else
230                     -- Transform all assertions of which at least one must be satisfied to 'OneOrMore' clause
231                     if(oneOrMoreAssertions.first() = e) then
232                         thisModule.LogicalOperatorOrAndNext2Alternative(e)
233                     else
234                         -- Transform all assertions of which at least one must be satisfied to 'ExactlyOne'
                                clause
235                         if(oneAssertions.first() = e) then
236                             thisModule.LogicalOperatorOr2Alternative(e)
237                         else
238                             -- If there are nested QoS specifications
239                             if(qosspec.packagedElement->exists( d | d.oclIsTypeOf(SOAML!SoaMLModel) and d.name=
                                    'QoS' ) ) then
240                                 qosspec.packagedElement->collect( g |
241                                     if( g.oclIsTypeOf(SOAML!SoaMLModel) and g.name='QoS' ) then
242                                        thisModule.NestedQoSPackage2All(g)
243                                     else OclUndefined endif
244                                 )
245
246                             else OclUndefined endif
247                         endif
248                     endif
249                 endif
250             )
251         )
252 }
253
254
255 -- Creates an 'All' clause for corresponding guarantee terms
256
257 lazy rule LogicalOperatorAll2Alternative {
258     from
259         qosconstraint : SOAML!QoSConstraint
260     to
261         andoperator : WSAgreement!All (
262             ownedTerms <- SOAML!QoSValue->allInstances()->collect( a |
263                -- Transform terms that belong in this clause
264                if a.name=qosconstraint.getAssertionName then
265                  a.slot->collect( x | thisModule.Slot2GuaranteeTerm(x) )
266                else
267                     -- If another constraint is in the same package as reference constraint, also include this
                            assertion
268                     if ( qosconstraint.refImmediateComposite().packagedElement->exists( d | d.isDependency ) )
                            then
269                         SOAML!QoSConstraint->allInstances()->collect( b |
```

```
                                -- If constraint uses 'AND' operator and has the same ID
                                if ( (b.usesAndOperator) and ( qosconstraint.reflImmediateComposite().
                                    packagedElement->one( d | d.name = b.name ) ) ) then
                                -- Check the term with current term, if OK; include this term in same clause
                                if( qosconstraint.reflImmediateComposite().packagedElement->any( d | d.name=b.
                                    name ).getAssertionName = a.name ) then
                                a.slot->collect( z | thisModule.Slot2GuaranteeTerm(z) )
                                else OclUndefined endif
                            else OclUndefined endif
                        )
                    else OclUndefined endif
                endif
            )
        )
}


-- Creates an 'ExactlyOne' clause for corresponding guarantee terms

lazy rule LogicalOperatorOr2Alternative {
    from
        qosconstraint : SOAML!QoSConstraint
    to
        oroperator : WSAgreement!ExactlyOne (
            ownedTerms <- SOAML!QoSValue->allInstances()->collect( a |
                -- Transform assertions that belong in this clause
                if a.name=qosconstraint.getAssertionName then
                    a.slot->collect( x | thisModule.Slot2GuaranteeTerm(x) )
                else
                    -- If another constraint is in the same package as reference constraint, also include his
                        referenced assertion
                    if (qosconstraint.reflImmediateComposite().packagedElement->exists( d | d.isDependency) )
                        then
                        SOAML!QoSConstraint->allInstances()->collect( b |
                            -- If constraint uses 'ExactlyOne' operator and has the same ID
                            if ( (b.usesOrOperator) and ( qosconstraint.reflImmediateComposite().
                                packagedElement->one( d | d.name = b.name ) ) ) then
                            -- Check the assertions with current assertion, if OK; include this assertion
                                in same clause
                            if( qosconstraint.reflImmediateComposite().packagedElement->any( d | d.name = b.
                                name).getAssertionName = a.name ) then
                                a.slot->collect( z | thisModule.Slot2GuaranteeTerm(z) )
                            else OclUndefined endif
                        else OclUndefined endif
                        )
                    else OclUndefined endif
                endif
            )
        )
}


-- Creates an 'OneOrMore' clause for corresponding guarantee terms

lazy rule LogicalOperatorOrAndNext2Alternative {
    from
        qosconstraint : SOAML!QoSConstraint
    to
        oroperator : WSAgreement!OneOrMore (
            ownedTerms <- SOAML!QoSValue->allInstances()->collect( a |
                -- Transform assertions that belong in this clause
                if a.name=qosconstraint.getAssertionName then
                    a.slot->collect( x | thisModule.Slot2GuaranteeTerm(x) )
                else
```

```
327                     -- If another constraint is in the same package as reference constraint, also include his
                            referenced assertion
328                     if ( qosconstraint.reflmmediateComposite().packagedElement->exists( d | d.isDependency ) )
                            then
329                         SOAML!QoSConstraint->allInstances()->collect( b |
330                             -- If constraint uses 'OnOreMore' operator and has the same ID
331                             if ( (b.usesOrOperatorAndNext) and ( qosconstraint.reflmmediateComposite().
                                    packagedElement->one( d | d.name = b.name ) ) ) then
332                                 -- Check the assertions with current assertion, if OK; include this assertion
                                        in same clause
333                                 if( qosconstraint.reflmmediateComposite().packagedElement->any( d | d.name = b.
                                        name ).getAssertionName = a.name) then
334                                     a.slot->collect( z | thisModule.Slot2GuaranteeTerm(z) )
335                                 else OclUndefined endif
336                             else OclUndefined endif
337                         )
338                     else OclUndefined endif
339                 endif
340             )
341         )
342 }
343
344
345 -- Transform slots into guarantee terms
346
347 lazy rule Slot2GuaranteeTerm {
348     from
349         soamlSlot : SOAML!Slot
350     to
351         agreementcontext : WSAgreement!GuaranteeTerm
352         (
353             name <- thisModule.contractSubject.concat('-').concat(soamlSlot.definingFeature.name),
354             -- Service provider is responsible by default
355             obligated <- #ServiceProvider,
356
357             slo <- thisModule.Slot2ServiceLevelObjective(soamlSlot),
358             bvl <- thisModule.Slot2BusinessValueList(soamlSlot),
359             scope <- thisModule.Slot2ServiceScope(soamlSlot)
360         )
361 }
362
363
364 -- Generate service scope element for guarantee term
365
366 lazy rule Slot2ServiceScope {
367     from
368         soamlSlot : SOAML!Slot
369     to
370         bv : WSAgreement!ServiceScope
371         (
372             serviceName <- thisModule.contractSubject.concat('Service'),
373             scopeSDT <- thisModule.QoSPackage2ServiceDescriptionTerm(thisModule.rootQoSSpec)
374         )
375 }
376
377
378 -- Generates general service description for service contract subject
379
380 unique lazy rule QoSPackage2ServiceDescriptionTerm {
381     from
382         qosspec : SOAML!SoaMLModel
383     to
384         agreementcontext : WSAgreement!ServiceDescriptionTerm
385         (
386             serviceName <-  thisModule.contractSubject.concat('Service'),
```

```
387             name <- 'General'
388         )
389 }
390
391
392 -- Generates service properties definition from QoS value
393
394 unique lazy rule QoSValue2ServiceProperties {
395     from
396         soamlQoSValue : SOAML!QoSValue
397     to
398         serviceproperties : WSAgreement!ServiceProperties
399         (
400             name <- soamlQoSValue.name.concat('Property'),
401             serviceName <- thisModule.contractSubject.concat('Service'),
402             -- Define variables of the service property
403             variableSet <- soamlQoSValue.slot->collect( a | thisModule.QoSDimension2Variable(a) )
404         )
405 }
406
407
408 -- Transforms dimension refered by slot to property variable
409
410 unique lazy rule QoSDimension2Variable {
411     from
412         soamlSlot : SOAML!Slot
413     to
414         variable : WSAgreement!Variable
415         (
416             name <- soamlSlot.definingFeature.name,
417             location <- thisModule.QoSPackage2ServiceDescriptionTerm(thisModule.rootQoSSpec)
418         )
419 }
420
421
422 -- Transform slot into Service Level Objective
423
424 lazy rule Slot2ServiceLevelObjective {
425     from
426         soamlSlot : SOAML!Slot
427     to
428         slo : WSAgreement!ServiceLevelObjective
429         (
430             assertion <- thisModule.Slot2QualityAttribute(soamlSlot)
431         )
432 }
433
434
435 -- Transform slot into to be filled-in business values
436
437 lazy rule Slot2BusinessValueList {
438     from
439         soamlSlot : SOAML!Slot
440     to
441         bv : WSAgreement!BusinessValueList
442         (
443             importance <- 1
444         )
445 }
446
447
448 ------------------
449 -- Custom grammar rules
450 ------------------
451
```

```
452 -- Maps each slot to an assertion (QualityAttribute elements in our custom grammar)
453
454 lazy rule Slot2QualityAttribute {
455     from
456         soamlSlot : SOAML!Slot
457     to
458         qualityAttribute : WSAgreement!QualityAttribute (
459             name <- soamlSlot.definingFeature.name.concat('-definition'),
460             value <- soamlSlot.value.first().value,
461             unit <- soamlSlot.definingFeature.unit,
462             qualifier <- thisModule.qualifierMap.get(soamlSlot.definingFeature.statisticalQualifier),
463             direction <- thisModule.directionMap.get(soamlSlot.definingFeature.direction),
464             usedProperty <- thisModule.QoSDimension2Variable(soamlSlot)
465         )
466 }
```

In this transformation element names are used to identify elements. When using this transformation, do not forget to uniquely name elements inside a QoS package.