Model-Driven SOA

Report for research topics

Author:
Niels Schot

Study:
Computer Science - Software Engineering

Student number:
s1048406

University of Twente
EEMCS - Software Engineering Group

April 2012

version 1.0
# Contents

1 Introduction 5  
1.1 Motivation .................................. 5  
1.2 Objectives .................................. 6  
1.3 Relevant MDE principles ................. 6  
1.3.1 Metamodels ................................ 7  
1.4 Literature study scope ..................... 8  
1.5 Approach .................................. 8  
1.6 Structure of report ......................... 9  

2 The service life-cycle 10  
2.1 Criteria identification ...................... 10  
2.2 Terminology ................................ 11  
2.3 Service life-cycle ......................... 11  
2.3.1 Service design .......................... 12  
2.3.2 Service discovery ....................... 12  
2.3.3 Service composition ..................... 12  
2.3.4 Service implementation ................. 13  
2.3.5 Service publishing ...................... 13  
2.3.6 Service provision and negotiation .... 13  
2.3.7 Service monitoring ...................... 14  
2.3.8 Service management ..................... 14  
2.4 Stakeholders ............................... 14  
2.4.1 Beneficiary .............................. 14  
2.4.2 Participants ............................. 14  
2.4.3 Service developer ....................... 15  
2.4.4 Service provider ......................... 16  
2.4.5 Service consumer ....................... 17  
2.4.6 Service broker .......................... 19  
2.4.7 Non-participants ......................... 19  
2.4.8 Overview stakeholders ................. 20  

3 SOA modeling criteria 21  
3.1 Approach .................................. 21  
3.2 Functional SOA concepts ................. 22  
3.2.1 Service ................................ 22  
3.2.2 Service interface and implementation ... 22  
3.2.3 Service discovery and service descriptions . 23  
3.2.4 Service contracting and interaction .... 23  
3.2.5 SOA application ......................... 25  
3.2.6 Functional overview ..................... 25  
3.3 Non-functional SOA concepts ............ 26
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.1 Policies</td>
<td>26</td>
</tr>
<tr>
<td>3.3.2 Non-functional overview</td>
<td>27</td>
</tr>
<tr>
<td>3.4 Criteria overview</td>
<td>28</td>
</tr>
<tr>
<td>3.5 Conclusion</td>
<td>30</td>
</tr>
<tr>
<td>4 Service-oriented modeling</td>
<td>32</td>
</tr>
<tr>
<td>4.1 Role of UML</td>
<td>32</td>
</tr>
<tr>
<td>4.2 Metamodel and UML profiles</td>
<td>33</td>
</tr>
<tr>
<td>4.3 Overview SOA modeling approaches</td>
<td>33</td>
</tr>
<tr>
<td>4.3.1 Conclusion</td>
<td>38</td>
</tr>
<tr>
<td>5 SoaML evaluation</td>
<td>40</td>
</tr>
<tr>
<td>5.1 Criteria evaluation</td>
<td>40</td>
</tr>
<tr>
<td>5.1.1 The case</td>
<td>40</td>
</tr>
<tr>
<td>5.1.2 Services and contracting</td>
<td>41</td>
</tr>
<tr>
<td>5.1.3 SOA application architecture</td>
<td>45</td>
</tr>
<tr>
<td>5.1.4 Service composition</td>
<td>46</td>
</tr>
<tr>
<td>5.1.5 Service discovery</td>
<td>47</td>
</tr>
<tr>
<td>5.1.6 Evaluation conclusion</td>
<td>48</td>
</tr>
<tr>
<td>5.2 Tool support</td>
<td>49</td>
</tr>
<tr>
<td>5.2.1 Tool conclusion</td>
<td>50</td>
</tr>
<tr>
<td>6 Conclusion</td>
<td>52</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This chapter gives an introduction to the purpose of this document. It explains the motivation, the objectives, the used approach and the structure of the report.

1.1 Motivation

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 [1]. 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 for implementing a model-driven approach by providing a set of tools that manage models proposed by the Object Management Group (OMG) [2]. It proposes a separation of the development process in different abstraction levels, such as a PIM (platform independent model) and a PSM (platform specific model). The first model abstracts from platform-specific details, which are considered in the second one.

MDE is more comprehensive than MDA because it considers multiple modeling dimensions in addition to the platform-independent/platform-specific dimensions, such as different kind of models and organizational issues [1]. The real motivation for MDE is to preserve analysis, conception and development efforts, improve productivity and facilitate the migration of applications from one platform to another [1].

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, which is beneficial because in software development the focus changes from applications to reusable services. Consumers can access SOA services in a standardized way and without needing to understand how the service is implemented [3].

Because SOAs can consist of several artefacts such as services, contracts, participants and relations, they could become very complex. A good way to understand them would be to somehow model these architectures. SOA models would help explain, formalize and understand these architectures. There are several initiatives to model SOAs [4] [5] [2].

A promising approach would be to apply MDE to the development of SOAs [6] [7] [8]. A potential benefit is that the SOA application would be 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 models from technology independent ones [9]. This could increase the speed of development of SOAs and thus reducing software costs. Migrating to other or newer technologies should cost less effort when following the MDE approach [1].

1.2 Objectives

The goal of this document is to give an overview of the state of the art of model-driven SOA. It identifies several topics and aspects related to MDE applied to SOA. The ultimate goal is to find an interesting research topic for a Master’s assignment.

The questions to be answered first in this report are:

- What is the state of the art of model-driven SOA?
- What is required for stakeholders to apply MDE to SOA?
  - What languages/tools are needed for developers to apply MDE to SOA?
  - What knowledge is needed/important for companies to apply MDE to SOA?
- Which languages, methodologies and/or tools are available for modeling SOA applications?
  - What are the constraints and benefits of these approaches, which SOA aspects are not properly supported?

1.3 Relevant MDE principles

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 abstraction levels:

1. Computation-independent models (CIM) are a view of a system from the computation independent viewpoint. Computation independent viewpoint focuses on the on the environment of the system, and the requirements for the system; the details of the structure and processing of the system are hidden or as 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 [10].

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 [10].
3. Platform-specific models (PSM) are a view of a system from the platform specific viewpoint. The platform specific viewpoint combines the platform independent viewpoint with an additional focus on the detail of the use of a specific platform by a system. A PSM combines the specifications in the PIM with the details that specify how that system uses a particular type of platform [10].

In principle the MDA process starts with defining CIM or domain models, then these models can be transformed by professionals to PIM and PSM models. These transformation are called vertical transformations. A direct vertical transformation is not always possible because there 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 [11] supporting the platform-specific realization of a model.

When applied to SOA, 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/technology being used. It includes models of service interfaces, service contracts, service enactments, participants, etc [8]. Models at this abstraction level are considered as the software specification models.

The PSM-level describes SOA 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) [12]. Another example is the Business Process Execution Language (BPEL) for specifying business process behavior based on web services [13].

1.3.1 Metamodels

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

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.

Firstly, a mechanism is needed to unambiguously define modeling languages, so that a transformation tool can then read, write, and understand the models. Secondly, the transformation rules that define a transformation describe how a model in a source language can be transformed into a model in a target language. These rules use the metamodels of the source and target languages to define the transformation [14].

These model transformations can then 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) [15] and Query/View/Transformation (QVT) [16].
1.4 Literature study scope

This report mainly focuses on the PIM abstraction level for modeling SOAs. These models consider the SOA concepts which are interesting for us. These concepts depend on the purpose of the model. Modeling always should have a purpose, for instance, for generating an implementation, refining or analysis.

This further means that this document mainly focuses on (technical) service models rather than business process 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.

1.5 Approach

A literature study has been performed to answer the questions formulated in section 1.2. We want to explore the current state-of-the-art of model-driven SOA and in order to do this, we defined criteria to select interesting languages/tools for further evaluation. As input we use research papers and consolidated knowledge from SOA standardization work.

These criteria are defined using a top-down approach and have been identifier from an analysis of the service life-cycle. A complete service life-cycle covers all the activities stakeholders have to deal with, and therefore covers all essential criteria, starting from service design to service management. In order to improve the development of SOA applications with MDE, we have to apply MDE somewhere in the service life-cycle.

Therefore, we first defined a detailed and as complete as possible service life-cycle by studying relevant literature. We then identified the stakeholders involved in the phases of this life-cycle because we were interested in the criteria from their point of view. We have then positioned these criteria at various dimensions, looked at the details, and investigated whether MDE could improve the activities.

With these criteria we evaluated current model-driven SOA developments and identified missing aspects and opportunities for further research. Figure 1.1 illustrates the used approach.

![Figure 1.1: Approach literature study](image)

The main references for this report were collected from OMG (www.omgwiki.org/SoaML), OASIS (www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm) and the Open Group (www.opengroup.org/projects/soa/), which are the websites of the standardization groups. Additional literature was collected from IEEE (www.ieee.org).
CHAPTER 1. INTRODUCTION

It includes specification documents such as reference architectures, reference models, ontologies and language specifications of SOA standards. Additional journal articles and conference papers about SOA and MDE were found via keyword search.

1.6 Structure of report

Chapter 2 explains the service life-cycle and the corresponding stakeholders. The activities of the stakeholders are analyzed to identify SOA concepts which are important for them, and how these activities should be supported.

Chapter 3 identifies criteria for modeling SOA applications based on the findings of Chapter 2. It explains important concepts of SOA to define criteria which are used to analyze SOA modeling techniques.

Chapter 4 discusses SOA modeling. The chapter analyzes current developments with respect to SOA modeling based on the identified criteria. SOA modeling approaches for further research are selected in this chapter.

Chapter 5 evaluates SoaML in detail by using the identified criteria, in terms of aspects that are covered and well-supported, and aspects that are missing. The language is applied to an example case. The chapter also gives an overview of the tool support.

Chapter 6 answers the research question defined in section 1.2 and gives a conclusion. In the conclusion the current state-of-the-art is summarized and it identifies which interesting aspects require further research.
Chapter 2

The service life-cycle

This chapter introduces the service life-cycle and its stakeholders. It explains when, why and which activities are performed by the stakeholders and what they need to perform these activities.

2.1 Criteria identification

In this report we are particularly interested in how to model applications designed according to the SOA principles. With our approach formulated in section 1.5 we define a complete set of criteria for model-driven SOA. Many imaginable service models for all kind of purposes can be considered in this literature study, but we only consider models as far as they are relevant for the activities of stakeholders in the service life-cycle. In chapter 4 different SOA modeling developments are analyzed based on the criteria.

Figure 2.1 shows the approach used to define the criteria. In order to validate the criteria, we searched literature for concepts which are important for modeling SOA applications. We look at the consolidated knowledge in SOA standards (from OASIS, OMG and the Open Group), and to new developments in relevant research papers (found via keyword search).

Figure 2.1: Approach identifying criteria
CHAPTER 2. THE SERVICE LIFE-CYCLE

2.2 Terminology

Figure 2.2 introduces the terminology used in this report. We use the term service description to describe all models of a service, describing the functionality, interface, terms, conditions and behavior of the service. The “SOA application” is the application that uses services, which is developed for the end-user (the potential customer). The end-user may be the company if an employee of the company uses SOA to improve its business. A SOA application could involve multiple services and stakeholders. Services have a service interface, which is publicly available for others, and a 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 the contract (so that the contract is instantiated), we call it a service contract.

![Figure 2.2: SOA terminology](image)

2.3 Service life-cycle

A service life-cycle describes through which phases a service goes from design to management. Some commonly identified phases of a service life-cycle [17] [18] [19] are listed below, where the stakeholders’ points of view are not considered yet. Figure 2.8 shows the phases we have identified from the literature.

The black arrows connect the phases through which a service generally goes. The grey dashed arrows show that certain phases are related to each other. For instance, a service composition phase can be part of the service design phase, and service discovery only makes sense after services are published.
2.3.1 Service design

The first step is to make a design of a service which complies with functional and non-functional requirements. This should be done when requirements and business processes have been defined, but as explained in section 1.4, we assume these are known. In the first place this means, that a potential user should search for already available services which can be reused, and then decide whether to use/modify an existing service or built a new one [17]. In this step we recognize that service discovery is needed to identify possible reusable services. We also might need service composition to compose new services from other services.

For this study service design is an important phase because in this phase we need to model the service that we want to be realized. It is a step where we want to model the SOA application at PIM-level, which is our primary goal. This means we want to specify how a service must behave and how it must be used.

When this phase ends, several service aspects must have been modeled. Such as, e.g. the specification of the service interface(s), the interactions between service and client, possible specification of quality requirements and possible specification of service composition.

2.3.2 Service discovery

Service discovery is a phase where published services are discovered via a service registry. This phase is used by stakeholders to find published services. It can also be considered as a (sub)phase or part of the service design phase, to make potential reuse of other services possible. These services might be used in a service composition, in which case service design stakeholders also want to discover services.

2.3.3 Service composition

Service composition can also be considered as a (sub)phase or part of the service design phase to make the design of a composite service possible. In this case, other discovered services are
reused to build a new composite service. Alternately, services may be composed at runtime, which means multiple services are selected and used together to fulfill a certain business goal in a SOA application [17][19] after services have been deployed.

2.3.4 Service implementation

In this phase, the service is implemented based on the service design. This service may be built from scratch or by reusing/modifying other services using service composition, as specified in the design phase. In this step we want to realize the models, i.e. implement the design for a certain platform. The step between the service design and service implementation phase is a typical step where model transformations (e.g., PIM to PSM) can be used to speed up the implementation phase [9][20].

The service implementation phase may also include testing of the service. This comprises testing the requirements of the service, including the possible quality requirements of the service design.

2.3.5 Service publishing

When services are implemented and tested they can be published to a service registry to make it available to stakeholders who want to use the service. The registry needs a description of the service containing information about the effects, functionality, interface, requirements and other criteria.

2.3.6 Service provision and negotiation

![General service negotiation view](image)

Figure 2.4: General service negotiation view
When services are published, potential users 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 [17]. When all involved parties agree on the contract, the service can be provided according to the contract and used in the SOA application.

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 service he provides. These offers and requirements can relate to many aspect such as security, privacy, availability and functionality. We can consider these requirements/offers as a policy of that provider.

2.3.7 Service monitoring

When services are published and used, we want to verify if the agreed contract is met. In order to do so, monitoring the behavior of the service helps to check whether the service still operates within the specified bounds.

2.3.8 Service management

Furthermore services might need maintenance. Services need to be managed so that when changes occur, the impact of change on clients is reduced to a minimum [17].

2.4 Stakeholders

In this section we identify the involved stakeholders according to the service life-cycle as depicted in 2.8. In principle, each participant is responsible for part of the service life-cycle.

2.4.1 Beneficiary

A company normally uses SOA applications to improve its business processes, i.e. reduce costs, improve productivity and fulfill business goals [21]. The term “SOA governance” is often used for all activities related to exercising control over services in a SOA [22].

For this literature study we are not directly interested in criteria at that abstraction level, but it indirectly relates to the service life-cycle. We are interested in the technical services and SOA applications, and do not consider the business processes. We are more interested in companies, company departments, or employees who play the role of a participant. That means, stakeholders which are involved in the modeling of a SOA application.

2.4.2 Participants

In order to model a SOA, participants should be included in the model. Participants are the entities that provide or use services. Participants could represent people, information system components or organizations [8].
A participant that provides a service is generally called a “provider”. A participant that uses a service is called a “consumer”. A SOA consists of participants who consume and provide services [23] [8]. The provider and consumer are the main roles in a SOA.

A service provider refers to the role of, for instance, a development company or department that produces and publishes services which are ready to be executed. Service providers are the owners of services [17] [22]. Providers must know how to design, implement, deploy and publish services [23].

To be more specific, we can distinguish the service developers from the actual service provider. A development company or department can often play both roles of developer and provider, but a certain development company or department may also develop a service, while another party publishes and provides it.

2.4.3 Service developer

Figure 2.5 shows the phases that are relevant for the service developer. We can distinguish the subroles service designer and service implementer since these activities can be performed by different parties.

In the service design phase a service designer needs to specify the service [17] 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 is able to specify the service’s functionality, interfaces, interaction protocols, messages and other criteria. This service specification should be at PIM-level.

When a service requires bi-directional interaction with other parties, 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 [8] which refers to the required interaction. In that case all parties should agree on the contract before the service can be used. The specification of service contract is not an activity for the 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. In this case, service composition is part of the design and implementation phase of the service, and the developer needs to learn how other services can be used in the new service. In this case, the role of the developer involves both service design and implementation. We can refer to this role as the composition designer. When no services are reused, the service implementer describes the service process according to the service designs with, for instance, programming code.

The service implementer may apply MDE in the development process. Especially the transition between design and implementation (and tests) is interesting to apply 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 [9]. Languages and tools to implement the service in the “service implementation” phase are necessary [17].
The service designer needs to be able to:

1. Discover services
2. Specify the service interface(s)
   Possibly specify the required interaction with other participants (in the form of service contract)
3. Possibly compose services according to service design (in the role of composition designer)

The service implementer needs to be able to:

1. Implement services according to service design

2.4.4 Service provider

Figure 2.6 shows the phases in which the service provider plays a role. The service provider publishes the service to the service registry in order to make it findable for potential consumers. A provider should be able to define constraints/conditions on the use of the provided service [22], which could be included in a service template. In case some party wants to use the published service and agreed to the contract template, a contract is established and the service provider has to provide the service in the “service provision” phase according to the agreed contract.

When the service is provided, monitoring is needed to check the agreements in the contract are met. For instance, if the contract defines that the service should have an availability of 99%, then service monitoring should measure and check if the service is available for that time. 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 [17], e.g. in an agreement.
CHAPTER 2. THE SERVICE LIFE-CYCLE

The service provider needs to be able to:

1. Publish services
2. Specify service description (according to design)
   Possibly specify a more advanced “service contract” with his offers
3. Monitor services
4. Maintain services

2.4.5 Service consumer

Entities that use services are the service consumers. They are considered to be a service consumer when they use services (possible found in a service registry). A service consumer uses services for the realization of a SOA application (as depicted in figure 2.2). 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 [8] [23] [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 [8].

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.
Figure 2.7: Service life-cycle service consumer

Figure 2.7 shows that the consumer is involved in service discovery, composition, the negotiation, monitoring and management. A consumer needs service discovery to find suitable services to fulfill its requirements. When a potential service is found in the registry, a negotiation between provider and consumer starts to agree on the usage of the service. The result of this negotiation can be a service contract.

A service consumer might use other services, and thus use service composition to compose a service for his needs [19]. A service consumer 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 (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.

Modeling and maintaining an SOA application is also an activity of the service consumer who is responsible for the SOA application. That stakeholder needs a language to model the SOA application. For instance, he might need a language to describe the application architecture (high-level architecture) if this helps to improve the business process and understanding for all involved stakeholders.

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

The service consumer needs to be able to:

1. Discover services
2. Agree on the service contract and use services
3. Possibly model an SOA application
   - Possibly compose services (in SOA application) in the role of composition designer
4. Possibly monitor service
5. Possibly maintain SOA application
2.4.6 Service broker

A possible third role is that of a service broker (or mediator), which acts as an intermediary role between a provider and a consumer. The main role of a service broker is to provide service information which is contained in a service registry to support service discovery. We assume that services should be visible for consumers somehow [22] [8] 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 [24] [23].

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 other participants as less 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:

1. Support service discovery (with a service registry)
2. Maintain the service registry

2.4.7 Non-participants

Furthermore there may be non-participants, which are entities not directly involved in (or part of) the SOA itself [23], but that have an interest in the SOA. The interest of a non-participant stakeholder may be in realizing the benefits of a well-functioning SOA system and not suffering unwanted consequences. Example of these stakeholders are potential customers, affected third parties, as well as potential parties who might seek a negative impact on the SOA (such as hackers or criminals) [23].
A criterion for non-participants may be the real world effect of an SOA application. But this real-world effect is indirectly already a criterion for the service consumer. Because the consumer wants to build a SOA application with a certain purpose, for instance, to build an application that uses a service that allows end-users to buy items, the real world effect is then that an end-user can buy items with the developed SOA application. Since these non-participants do not play a role in the service life-cycle and they require no models, we do not further consider them in this document.

### 2.4.8 Overview stakeholders

Below we list the stakeholders we identified according to the service life-cycle. Each stakeholder is somehow involved in (or has advantage of) the service-life cycle, but the developer, provider and consumer are most important. The service developers need to define the service descriptions. Service providers need to know how to publish and provide services. Service consumers need to know how they can use available services. In the context of this study, these are the stakeholders who need models for the activities they perform during the life-cycle of a service. The activities of the service broker must be present in order to realize the SOA in a real world application, and the company wants to realize its business goals in the best way possible.

1. Company
2. Participants
   - **Providing services**
     - **Service developer**
       - Service designer
       - Service implemener
     - **Service provider**
   - **Consuming services**
     - **Service consumer**
     - **Service broker**
3. Non-participants
Chapter 3

SOA modeling criteria

This chapter introduces important concepts of model-driven SOA. It explains the criteria for the evaluation and selection of languages and tools in this literature study.

3.1 Approach

The SOA concepts identified in chapter 2 are elaborated below. In this chapter we organize criteria for evaluating SOA models according to the following dimensions:

• Syntax ↔ semantics

We considered the dimension of syntax and semantics in these modeling languages. All languages need a syntax which defines the combinations of symbols that are considered to be correctly structured programs in that language. Languages also have semantics, which refers to the meaning of the symbols arranged with that structure. We investigated how SOA concepts can be represented and how we can define their meaning in the SOA models.

• Abstract ↔ concrete

We also looked at the level of abstraction in which these concepts are modeled. Are they modeled at an abstract level, or concrete with more detail. For instance, a consumer might need a model of the service with much detail to consume the service, which includes the specification of available operations and the interaction protocol. However, in some cases, abstract views of services may be sufficient, e.g., to show how participants work together in a SOA application.

• Functional ↔ non-functional

Criteria can relate to the functional concepts which are used to define an operating SOA application. In addition, other criteria relate to non-functional aspects of SOAs, because modeling non-functional aspects is often necessary to describe requirements on the quality of service. Examples are availability, performance and security.

Structure ↔ behavior

We can further categorize the SOA models into structure and behavior. Models are used to represent structures (data structures, components) or behaviors (what must happen in
the service or SOA application being modeled). We consider the purposes a SOA model has with respect to this dimension.

- Stakeholders’ viewpoints

The fourth dimension is the viewpoint of the stakeholders. We can look from the viewpoints of the different stakeholders (with respect to the service life-cycle) as we have identified in chapter 2.

The functional/non-functional dimension is leading in the analysis. First the SOA concepts with a functional purpose are explained, then the non-functional concepts are explained. The other dimensions are addressed inside the context of the functional and non-functional concepts.

### 3.2 Functional SOA concepts

#### 3.2.1 Service

The key concept of an SOA is a service. A service is realized by a given application or system, and represents the external behavior of the application or system [19]. A service should have the capability to perform work, a specification of the work offered, and the offer to perform work [22] [8] [21]. Services are essential and involved in the activities of all stakeholders. Ultimately services have to be modeled both syntactically and semantically at different abstraction levels. Both structural and behavioral views are needed to model all necessary information, as explained below.

Some important functional aspects of a service which should be modeled are [22]:

1. For both a service consumer and provider it is important to know how to communicate with each other, i.e. how a consumer can use services provided by a provider. This, for instance, requires specifications of the service interfaces, messages, and required interaction protocol.
2. For a consumer it is important to know the (real world) conditions and effects of using the service. Pre- and postconditions of a services must be defined.

#### 3.2.2 Service interface and implementation

A service developer needs to specify the service interfaces. This specification includes the interface that the provider offers as well as the interface, if any, it expects from the consumer [8]. A complete service interface specification should describe all information necessary for a consumer to use the service.

For separation of concerns, the interface is independent but consistent with how the developer implements the service [8] and how the consumer consumes it. So we can distinguish the outside (interface) and the inside (implementation) of a service. For potential consumers it is important to know how they can use the service in terms of the required message format (structure) and order of messages and operations (behavior). These consumers do not need information about the inside of the service and can thus use abstract models with respect to the service internal structure.
CHAPTER 3. SOA MODELING CRITERIA

From a behavioral point of view we can look whether it is possible for the service designer to model the internal service implementation in a platform-independent way. A service designer might want to model the service implementation at PIM-level to profit from the MDE benefits. For instance, modeling service behavior at PIM-level improves understanding for other stakeholders, such as the service implementer, and it facilitates possible transformations to PSM-level models.

3.2.3 Service discovery and service descriptions

The provider, consumer and possible broker are involved in service discovery. Realizing service discovery via a service broker needs service descriptions and a service registry.

The service description is used by a service provider to publish the service. The information added to the registry should give information about the interface, effects, location, and possible additional requirements of the service. A (potential) consumer uses this information, possibly found in the registry, to consider using the service, and to use it.

3.2.4 Service contracting and interaction

Sometimes a more advanced specification is needed to model more complex (e.g., bi-directional) behavior than a simple request-response scenario, for instance, in terms of a service contract. These service contracts can refer to a specification of the required interaction between participants. Such a contract can help specify how participants (and possible other roles) work together to exchange value. When consumers agree on these contracts, they agree to use the service as defined in the contract, and it should be possible to identify how these contracts are enforced [23].

Service contract behavior

![Figure 3.1: Service contract instantiation](image)

The service contract behavior should be modeled by the developer, this is often done in the form of a choreography. The choreography serves to establish an agreement between multiple
stakeholders in terms of message exchanges. It allows them to understand how their systems should interact. With a choreography it is possible to define what information is sent and in what order between provider(s) and consumer(s) at a concrete level. A choreography specifies the interaction between participants in detail. Figure 3.1 illustrates what functional aspects are included in a service contract.

In [22] a design is expected to describe how interaction is mediated and the execution context required to support interaction. With other words, the expected behavior of participants in order to use the service should be modeled. Service discovery and real-world conditions and effects are also important in this case.

Service composition

Service composition is an important concept used by the service developer and consumer. SOA fosters the re-use of available services as components in service compositions [19]. Service composition is identified as an additional criteria for the service consumer. A service composition is an aggregate of services collectively composed to automate a particular task or business process [25]. Service composition is the act of composing a service from one or more other services.

We consider two points of view with respect to service composition. From the point of view of the service developer, service composition is used at design time to make a new service from existing services. In this way a service developer makes the implementation of a new service based on other services.

From the point of view of the service consumer, dynamic service composition may be used to let several services work together (at runtime) to fulfill a certain task in an SOA application. In this case, the consumer plays the role of a composition designer.

Orchestration

A concept used to describe a composition is orchestration. Similarly to a choreography, an orchestration describes service behavior, but differs from choreography in that it describes a
process flow between services, controlled by a single party [27]. Therefore, both terms are related to refer to the description of service collaborations, but differently. A service orchestration guarantees that services in an SOA application can be coordinated to work together to fulfill business requirements. An orchestration is a detailed, and possibly executable specification of a composite service.

Figure 3.2 illustrates the difference between choreography and orchestration. From a choreography perspective we look at the public process between multiple parties, while with an orchestration we describe the private interaction process from the perspective of a single certain party.

The stakeholder responsible for the SOA application can use orchestration to model parts of the SOA application. For example, this stakeholder can use an orchestration to make a service composition. With an orchestration language the stakeholder could model a composite service at PIM-level, so that the stakeholder plays the role of a service designer.

### 3.2.5 SOA application

Some service stakeholders are responsible for modeling an SOA application that uses services. Describing the collaboration of used services is needed, but a model to understand the overall SOA application might also be needed in more advanced SOA applications when multiple services, providers and consumers are involved [8]. Such a model should give an overview of the architecture describing how participants (with their possible roles as provider or consumer) work together at a more abstract level.

### 3.2.6 Functional overview

The functional SOA concepts identified up to here, which should be modeled in SOA-based applications, are listed below.

1. **Services**
   - Service interface (structure, behavior)
   - Service implementation (behavior)
   - Service composition for developer (behavior)
2. **Service contract (when necessary)**
   - Contains information about service interface(s) (behavior)
   - Choreography (behavior)
   - Contract enforcement (behavior)
3. **Service visibility (behavior)**
   - Service discovery
   - Service broker/registry (structure, behavior)
   - Service description providing information about service interfaces, location and real-world conditions and effects (behavior)
4. **SOA application**
   - Service composition for SOA application provider (behavior)
   - Overview of participants and services (structure, behavior)
3.3 Non-functional SOA concepts

Non-functional requirements specify criteria that can be used to judge the operation of a system, rather than specific behaviors. Examples are constraints that explicitly and intentionally restrict the system or process, and quality requirements which describe wanted qualities of the product that are not directly related to functionality [28].

When modeling SOA applications and services, often the term quality of service (QoS) is used to refer to the collection of quality constraints and requirements for a service. QoS refers to a level of service that is satisfactory to some user. It is important that QoS can be modeled [29] because these quality constraints and requirements can then be acknowledged by stakeholders. With respect to SOA it often covers security, performance, availability, and privacy. The importance of a certain quality factor 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 conditions, constraints and requirements on these aspects can be specified in the models [23]. Some important and interesting concepts are mentioned below.

3.3.1 Policies

Service developers, providers and consumers might need to specify the QoS on the usage of services. A concept often used to specify constraints on models is a policy [8][22]. 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: security, privacy, manageability, and so on [22]. For example, the provider may require service invocations through secure connections. Therefore we must look whether policies (or similar mechanisms) can be used to specify constraints for various concepts in SOA modeling languages. Policies relate to, but are different from, service contracts mentioned in section 3.2.4. 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 [22]. 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 and respected.

Figure 3.3 visualizes how the policies and contracts relate to each other. On the left side the proposed service contract of the service provider is illustrated. 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 a certain proposal, which leads, after a possible negotiation, to a final service contract that describes the agreements on the usage of the service.
Service level agreement (SLA) is a term often used in literature to denote a template for service contracts [23]. An instantiated SLA can represent the agreement on the policies as depicted in figure 3.3.

An SLA describes the intended boundaries for a service, particularly with regard to non-functional properties. SLA relates to QoS because SLA's are used when a certain level of verifiable quality is required. The key to crafting SLAs is to provide enough information or verifiable metrics for a service consumer to preselect services based on the desired level of quality [30]. Typically, SLAs are defined in plain text, using templates or toolkits. 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 he can consume the service. An SLA is part of the service description, and could be instantiated in the service contract [23].

Aggregated QoS

Furthermore we consider how the QoS of a composite service is defined, also known as the aggregated QoS. These are based on and influenced by the QoS of the component services [29]. Calculating the aggregated QoS would be helpful for the provider of the composite service to define a policy.

3.3.2 Non-functional overview

1. Specify policies (defining constraints, conditions and other requirements on the usage of services)
   - QoS modeling (with policies and/or SLAs)
   - Policy and contract enforcement (with service monitoring)
2. Service composition
   - Policies of composite service

Figure 3.3: Service contract instantiation with policies
### 3.4 Criteria overview

Table 3.1 summarizes the criteria we identified, with a short description and the involved stakeholders.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Description</th>
<th>Stakeholders</th>
</tr>
</thead>
</table>
| Services          | Possibility to specify service interfaces for service developer.  
                   
                   *Information necessary for consumers to communicate with the service, in terms of operations, parameters and messages.*                                                                                     | Service designer, service implementer, service consumer |
|                   | Support modeling the implementation of services  
                   
                   *Necessary for developers to define behavior of service (e.g. define implementation and composition of service at PIM level).*                                                                          | Service designer, service implementer, service consumer |
| Service contracting | Possibility to specify interaction between providers and consumers. Service description includes choreography and interfaces to specify interaction.  
                   
                   *In some cases the behavior and semantics of interaction between participants must be specified in more detail (e.g. in the case of bi-directional behavior). What information and in what order should be send between participants must then be described in the service description. Agreements on this interaction could be included in the service contract. All parties must agree on the contract before the service can be used.* | Service designer, service implementer, service provider, service consumer |
### CHAPTER 3. SOA MODELING CRITERIA

<table>
<thead>
<tr>
<th><strong>Possibility to instantiate policies in service contract.</strong>&lt;br&gt;<em>Necessary to include non-functional constraints from participants’ policies in the service contract. For instance needed to specify required QoS.</em></th>
<th><strong>Service designer, service provider, service consumer</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Possibility to monitor the fulfillment of service contracts.</strong>&lt;br&gt;<em>Necessary to monitor the contract. So that parties who agreed the contract may not violate it. Service monitoring can be used to enable contract enforcement.</em></td>
<td><strong>Service provider, service consumer</strong></td>
</tr>
<tr>
<td><strong>Service discovery</strong>&lt;br&gt;Supporting service discovery with service broker/registry&lt;br&gt;<em>Necessary so that consumers can find and use services provided by providers.</em></td>
<td><strong>Service provider, service consumer, service broker</strong></td>
</tr>
<tr>
<td><strong>Be able to include information about interfaces requirements, conditions, pre- and postconditions and real-world effects in the service registry.</strong>&lt;br&gt;<em>Necessary so that consumers can negotiate about and consider using services provided by providers.</em></td>
<td><strong>Service provider, service consumer, service broker</strong></td>
</tr>
<tr>
<td><strong>Policies</strong>&lt;br&gt;Be able to specify policies.&lt;br&gt;<em>Necessary so that conditions, constraints and requirements can be specified on the quality of service (security, privacy, manageability, etc.). This is necessary for participants to model required qualities for the usage or provision of a service.</em></td>
<td><strong>Service provider, service consumer</strong></td>
</tr>
</tbody>
</table>
CHAPTER 3. SOA MODELING CRITERIA

| Service composition | Service developer needs to be able to compose new services from other services.  
|                     | Necessary to reuse other services and build new services based on them. | Service developer |
|                     | Support for calculation of aggregated QoS when composing a new service.  
|                     | Necessary to calculate QoS of composite service based on component services. | Service consumer |
| SOA application     | Modeling the architecture of SOA application.  
|                     | Necessary for common understanding among all stakeholders of SOA application. | Service designer, service implementer, service consumer, service provider |

3.5 Conclusion

By searching the literature for SOA concepts and defining the criteria some of the subquestions can already be answered.

- What is required for stakeholders to apply MDE to SOA?

- What languages/tools are needed for developers to apply MDE to SOA?

The exact requirements on languages/tools to build SOA applications depend on the roles of the identified stakeholders and their activities.

A service developer needs a modeling language that supports the SOA concepts, as described in the criteria, to model the design of services. This developer also needs corresponding tools which facilitate the modeling of services. 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.

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). 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
CHAPTER 3. SOA MODELING CRITERIA

to PSM-level models to use the advantages of MDE.

Furthermore it is important for the stakeholders, responsible for the SOA application, to have a SOA modeling language and tools available which can be used to specify the SOA application. This includes the specification of the SOA architecture and detailed composition of services. The development of this SOA application can also be optimized with MDE, for instance with model transformations, when realizing the SOA application according to the design.

Furthermore there are additional criteria such as service information for consumers and the presence of a service registry that supports the discovery and negotiation on the usage of services in a real-world application.

- What knowledge is needed/important for companies to apply MDE to SOA?

For stakeholders it is important to know how to properly apply MDE to their activities in building 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 of a SOA application involves many parties, models of a certain stakeholder must be understood by their possible team members, other participants 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, instantiating the service contract for provider and consumer and realizing the SOA application according to the PIM-level designs.
Chapter 4

Service-oriented modeling

This chapter identifies and explains a selection of SOA modeling approaches based on the criteria mentioned in table 3.1.

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 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. [31] wrote an overview document that explains and positions these SOA architectural standards. Next to the open standards work, several SOA modeling initiatives come from researchers and research groups of universities. We have been primarily searching for SOA modeling languages, but we also looked at specifications of related approaches, such as reference models and reference architectures, since these can be used to create SOA modeling languages.

In this chapter the criteria of chapter 3 are used to classify the approaches and to identify in which ways these SOA modeling approaches support those SOA concepts. This chapter answers the following question:

- What are the constraints and benefits of the available SOA modeling approaches, and which SOA aspects are not properly supported?

4.1 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 [6]. Recent versions allow to describe many aspects of SOA applications [8]. However, [6] and [32] also emphasize the need to able to model a service component, which is impossible to model 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 [32]. This means that with standard UML it is not possible to formally model the SOA concepts we have mentioned in our criteria.
CHAPTER 4. SERVICE-ORIENTED MODELING

4.2 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 [33], [7] and [8].

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.

In our case 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 [34] and [15]. Furthermore the 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 [9]. This is beneficial for possible future work.

4.3 Overview SOA modeling approaches

Table 4.1 shows several SOA modeling approaches that have been briefly evaluated. 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. A short evaluation is done based on the criteria, and the most promising and relevant approaches are selected for further analysis.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Purpose</th>
</tr>
</thead>
</table>
| OASIS Reference Model (SOA RM) (2006) [22] | The SOA RM is intended to capture the “essence” of SOA, as well as provide a vocabulary and common understanding of SOA [31]. It is written at a high abstraction level and it support much of the SOA modeling criteria defined in chapter 3. | Explaining SOA core concepts  
Used to understand SOA concepts, no modeling language. |
| Open Group SOA Ontology (2010) [35] | 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 [31] [35]. | Explaining SOA core concepts  
Used to understand SOA concepts, no modeling language. |

Table 4.1: SOA modeling approaches

33
### OASIS Reference Architecture (OASIS RA) (2011) [23]

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 [31].

### 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 [31].

### Understanding SOA from different viewpoints

*Used to understand SOA concepts, less abstract than the reference model, no concrete modeling language.*

*Used to understand SOA concepts, much aimed at business integration.*
## 4. Service-Oriented Modeling

| **Soa Modeling Language (SoaML)** (2012) [8] | 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 [31]. 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 [8]. It keeps the main SOA modeling principles of PIM4SOA [36]. 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 [37]. | 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. Policies can be represented as UML constraints, QoS constraints can be represented with OMG QoS specification (not included in the SoaML standard)* |
|---|---|---|
| **A Platform Independent Model for SOA (PIM4SOA)** (2007) [38] | 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 [39] 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) [4] | 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 [4]. 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 [40]. 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 [2]. 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) [41] [5]

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 [41] 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) [7]

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) [42]

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 [42].

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.*
Wada et. al. proposed an UML profile for modeling non-functional aspects in service-oriented architectures. This graphically specifies and maintains non-functional aspects in SOA in an platform-independent manner [43].

Some SOA modeling approaches were not further evaluated because they are outdated or irrelevant. Examples are IBM’s “UML 2.0 Profile for Software Services” [44] which is deprecated (and replaced by SoaML), UML-RT, governance frameworks, maturity models and various other papers such as Zhang et. al [45] which seem to be overtaken by the SoaML standard.

Also the “Web Services Architecture” from W3C [46] was not evaluated further in detail because it just focuses on characteristics of web services, and does not propose a SOA modeling approach.

### 4.3.1 Conclusion

Table 4.1 shows there are various approaches to model SOAs. Several approaches (with and without a specific modeling language) are specifically aimed at SOAs.

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 [32] [37]. 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.

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.

The research work and metamodel proposed by Lopez-Sanz et. al. [7] 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.

SoaML is the first obvious choice as the SOA modeling language to be evaluated more deeply. The language is a recently proposed standard which is becoming increasingly popular. SoaML
can be used to model most of the mentioned criteria. However, we have observed that SoaML
does not directly support all of the criteria. For instance, service discovery and implementing
policies (incl. QoS specifications) might be an interesting area for further research in relation
with SoaML. It is also relevant and interesting to look at the tool support for SoaML because
tools are needed to support the activities of the stakeholders. To which degree do these tools
support and implement the SoaML specification, and are they suitable to support the MDE
principles.
Chapter 5

SoaML evaluation

This chapter evaluates the selected language SoaML in more detail. The language is applied to an example case. The chapter also provides an overview of the available tool support for SoaML.

5.1 Criteria evaluation

In this section we look in more detail how SoaML supports the criteria we defined in chapter 3. Throughout the chapter modeling of the SOA concepts will be applied to the following case.

5.1.1 The case

The used example case is a simplified example of the use case explained in a IBM tutorial [47]. The case is complex enough to test which SOA concepts are well covered by SoaML and which not. The following situation (high-level view) should be modeled: a particular company wants to order products from a manufacturer. The ordering process involves scheduling, invoicing and shipping of the ordered products.

![Figure 5.1: High-level view of the example](image-url)

40
Figure 5.1 illustrates the example case. Since the business process specifications are out of the scope of this research, we assume the following should be modeled: a service for the invoicing, scheduling and shipping, and a composite service that uses these services to order products.

The example case is modeled using the tool IBM Rational Software Architect. The models may look slightly different than the SoaML models in the SoaML specification, as explained in section 5.2.

5.1.2 Services and contracting

SoaML supports different approaches to specify services. This has resulted in the definition of different but overlapping language constructs. The specification distinguishes between three different approaches to specify a service:

1. The simple interface-based approach represents a one-way interactions provided by a service provider on a port represented as a UML interface. SoaML allows detailed specification of the required operations (including arguments and message types). The provider receives operations on this port and may provide results to the consumer. This approach can be used with “anonymous” consumers, and the provider makes no assumptions about the consumers or the choreography of the service [12].

2. The service interface-based approach represents binary and n-ary service interactions, requiring the designer to specify a set of related (simple) interfaces as one service specification. This approach uses UML components and allows the interconnection between these components through ports. In order to connect components through ports, the ports must specify both required and provided interfaces [12].

   The service interface-based approach is most applicable where existing capabilities are directly exposed as services and then used in various ways, or in situations that involve one or two parties in the service protocol [8].

   Interaction can be modeled, for instance, with a service choreography. SoaML is agnostic with regards to behavioral modeling, and mentions that any UML behavior, e.g., interaction models, activity models or state machines, can be used to model the choreography [12]. The service port then owns such a choreography model. The specification does not prescribe when which behavior model is best suitable.

3. The service contract based-approach extends a UML collaboration to specify binary or n-ary service interactions. The service contract-based approach is quite similar to the service interface based-approach in that it also represents binary and n-ary service interactions, requiring the designer to specify a set of related (simple) interfaces as one service specification. The service contract-based approach prescribes the definition of service specifications that define the roles each participant plays in the service (such as provider and consumer) and the interfaces they implement to play that role in that service.

   The service contract approach is useful when specifying interactions between two or more roles that imply the establishment of some agreement, e.g., through message exchanges. For instance, the service contract can contain a choreography model which specifies the interaction protocol between multiple participants. The fundamental difference between the contract-based and interface based-approach is whether the interaction be-
between participants is defined separately from the participants in a service contract, defining the obligations of all the participants, or individually on each participants’ service and request [8], respectively.

[12] shows that the service interface-based approach is suitable as a more refined model of a service contract. The collaboration diagrams in the service contract-based approach are often regarded as more appropriate for modeling the high-level service design, whereas the service interface-based approach is suitable for the more detailed models [12]. This is similar as the relation between choreographies and orchestrations as explained in section 3.2.4.

We start the example case with a model of the scheduling service. This service can be modeled as a simple interface because it only offers two one-way operations as depicted in figure 5.2. The service does not require any interfaces of the consumer.

The shipping service is more complex. The consumer of this service requests a shipping, but does not directly receive a shipping schedule. Therefore, this service could be implemented with a callback protocol, which requires bi-directional communication. This service can be modeled with a service-interface-based approach in SoaML. Figure 5.2 shows that the shipping service requires the consumer to accept a signal to acknowledge that the scheduling of the shipping is complete. The signal stereotype shows that the consumer receives an event when scheduling is complete. In this way SoaML supports an asynchronous request/response or callback pattern that is typical of many service protocols.

![Figure 5.2: Simple interface, and service interface-based approach SoaML](image)

The shipping service can also be modeled as service contract. Figure 5.3 shows the contract which defines the orderer (consumer) role and the shipper (provider) role. In figure 5.3, we have modeled the required interaction protocol as choreography. A sequence diagram is owned by the service contract and the defined behavior must be followed by the roles.

42
Instead of a sequence diagram, the required behavior may also be modeled with other behavior diagrams. Figure 5.4 shows an interaction protocol for the invoicing service expressed as an UML activity diagram.

Figure 5.3: Service contract SoaML

Figure 5.4: Service behavior with an activity diagram
Up to here we observed that SoaML fully supports the specification of service interfaces and service contracts in the functional sense. That means it supports the description of service operations and the required interaction between participants (and their roles) in various ways. The enforcement of functional aspects in a service interface/contract can also be realized with the concept of UML port. Ports of a provider and consumer should be compatible with the interface and interaction protocol.

Service implementation

SoaML itself does not directly cover modeling of the service implementation in the technical (PSM) sense, since service runtime is out the of scope of the SoaML standard. SoaML is designed in such a way that the service specifications are independent of how the service is implemented. SoaML relies on OMG MDA techniques to separate the logical implementation of a service from its possible physical realizations on various platforms [8].

Developers can specify possible compositions of services, and how services should be used (choreography) at PIM-level. Specifying the behavior of service interface operations is also possible at PIM-level with SoaML. A participant should implement each of its provided service operations in one the following ways:

1. Via a method, defined by, for instance, an interaction, activity or state diagram. The SoaML specification mentions that, when necessary, a PSM-level implementation can be included in the model as OpaqueBehavior (e.g., as Java code included in the model).
2. Via event handling. This allows participants to control when they are willing to respond to an event or service request, which can be used to realize asynchronous request/response.
3. Via delegation, allowing participants to delegate a service to a service provided by one of its parts, or to a user [8].

![Figure 5.5: PIM-level implementation of method](image)
Figure 5.5 shows the implementation of the operation of the composite purchasing service expressed with an activity diagram at PIM-level. When a method is complex it could help to model these operations at PIM-level because others stakeholders can use this model to understand the operation. The service implemener can also use these specifications to transform the service to the PSM-level.

The SoaML specification does prescribe which models might be suitable for model transformations and/or code generation, e.g., for the step of going from design to implementation performed by a service developer. SoaML claims to be suitable to MDA purposes because of SoaML’s platform-independence design and availability of a metamodel.

Policies and non-functional modeling

The SoaML specification does not address the specification of non-functional constraints. The specification describes that policies are constraints that can be owned rules of any model element, including service ports and service participant components. The actual form of these policies are out of scope for the SoaML specification [8]. Owned rules of UML model elements contain constraints. These UML constraints are an extension mechanism that enables you to refine the semantics of a UML model element. UML constraints can be used in the service interface and contract-based approaches to model policies. These UML constraints can be specified in, for instance, the Object Constraint Language (OCL) or in natural language. In addition, the OMG QoS specification [48] 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 perspectives 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 applicable constraints, policies, or qualities of service that are applicable to the categorized element. For example, a model element in the service layer of a SOA might have additional policies for distribution, security, and transaction scope [8].

In summary, SoaML gives some hints on the specification of constraints on service models, but it does not describe the actual form or give any examples. An instantiation of a policy in a service contract is possible with UML constraints, but the SoaML specification gives no further details 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 it belongs in the platform-specific domain.

5.1.3 SOA application architecture

Another criterion prescribes the need to model the architecture of SOA application, i.e., how participants and services work together. This is supported by SoaML in the form of a Community Services Architecture. The community services architecture is a top level view of how independent participants work together for some purpose [8]. The services architecture of a community is also modeled with a UML collaboration diagram.

Figure 5.6 shows a community services architecture of the example case. It shows how the roles and contracts are connected to each other. This diagram is an example of a more abstract diagram (than, for instance, the interface and participant diagrams) showing the external view of the parts. The goal of the community diagram is to understand how roles and services work together.
**5.1.4 Service composition**

In our criteria we have described the need to compose services from two points of view. First of all, a service developer needs to be able to reuse other services. That means the service developer develops a new service based on some other services. We also mentioned that a service consumer needs to be able to compose services to fulfill business goal “at runtime”. The main difference is that the service developer wants to make an explicit new service, and can also abstract from the composition and define the composition in the service implementation (at PSM level).

For both stakeholders this composition can be described with the SoaML’s Participant ServicesArchitecture and service contracts. The SoaML Participant ServicesArchitecture collaboration diagram can be used to specify the behavior of calling other services from that participant’s point of view. Such a model might be implemented with BPEL at PSM level, but the SoaML specification does not prescribe any specific PSM techniques. Both a service developer and consumer can use this kind of model to specify their behavior in the SOA application. The PIM-level implementation of the supposed composition, as possibly defined in such the contracts and architecture, can be modeled in a component diagram (e.g., in the internal structure of a Participant).

For the example case we have modeled the composite service used by the company to order the products. This service uses the other three services to process the purchased items. Figure 5.7 shows how the implementation of composite service can be modeled in SoaML in a Participant. The figure also shows how the roles from the service architecture are related to the participants.

Furthermore, SoaML supports nesting of service contracts and interfaces. For instance, there is the concept of compound service contracts, in which service contracts consist of other service contracts. A compound services contract should not be confused with a service that is implemented by calling other services, such as it may be specified with a Participant ServicesArchitecture and/or implemented with BPEL. A compound services contract defines a more granular
contract based on other service contracts [8]. It does not directly relate to service composition in the technical sense, but it is a composition feature of SoaML that promotes reuse and structure.

In the example case a compound service contract can be defined for the composite service. This contract should then include the contract for the invoicing, scheduling and shipping. Figure 5.8 shows a simplified example of a compound service contract expressing a purchasing service contract that uses the invoicing and scheduling contract. The client and manufacturer play roles in the nested contracts.

---

**Figure 5.7: Service composition using Participant diagram in SoaML**

**Figure 5.8: Compound service contract**

We also identified QoS-aware composition as a criterion. Since SoaML does not cover the specification of policies and non-functional constraints, this is also out of the scope of the SoaML specification.

### 5.1.5 Service discovery

We further acknowledge that the SoaML specification does not cover service discovery. Details on service registry (or a service broker) and service descriptions are not prescribed in the SoaML specifications, because these concepts belong to the platform-specific domain.
The specification explains that mechanisms for discovering existing services and the service descriptions that consumers would use to determine the applicability or availability of existing services for their needs (awareness), are out of the scope of SoaML, and are therefore not covered in the specification [8].

According to the SoaML standard, a specification of the service interface basically provides all the information needed by a consumer to consider using a service. These service specification then must include possible constraints associated with the service that define its non-functional characteristics or warranted quality of service. This information may also be used by potential consumers to determine if the service meets their needs [8]. This means that in a real-world application, first service descriptions have to be generated from the service specifications, which can then be published in the service registry.

When services are discovered, a negotiation on the contract might be required. In SoaML, the SOA application is presented as UML models and these models are generally considered to be static, however any of SoaML constructs could just as well be constructed dynamically in response to changing conditions. The semantics of SoaML are independent of the design-time, deployment-time, or runtime decision. For example, a new or specialized ServiceContract could be negotiated on the fly and immediately used between the specific participants. The ability of technology infrastructures to support such dynamic behavior is just emerging, but SoaML can support these infrastructures as it evolves [8].

5.1.6 Evaluation conclusion

We can conclude that SoaML is suitable to specify both services and the SOA application, 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 any 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 supporting all aspects of SOAs, but rather to be a small core that can be extended and integrated with other modeling languages, e.g., BPMN for behavioral modeling [12].

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. The extensions provide the required syntax to model SOA concepts, and can be used to express the semantics of the corresponding SOA concepts.

SoaML is also focused on loose coupling. Loosely coupled systems imply that services should be designed with little or no knowledge about particular consumers. Consumers may have a very different view of what to do with a service based on what they are trying to accomplish [8]. For example, service contract 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 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 easily be modeled by different stakeholders. For instance, a SOA application stakeholder models the overall architecture while a service implementer models some service operations. Furthermore SoaML allows nesting of several concepts, which further promotes modeling at different
abstraction levels.

For example, SoaML does not support service discovery. Service discovery this 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 activities that need refinements of PIM models are the design to implementation of a design as technical service (possibly including composition), and the enforcement of choreographies and other agreements in the contract.

5.2 Tool support

Since SoaML is a standard published in March 2012, its tool support is still limited. OMG’s SoaML wiki [49] lists some of the available tool support for SoaML.

Where possible, we prefer open-source tooling for our work because we are then not limited by licenses. Furthermore we also prefer tools that use the SoaML metamodel (as explained in section 4.2) instead of the UML profile.

Table 5.1 gives an overview of the currently available SoaML modeling tools.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Licensing</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModelPro</td>
<td>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 [49].</td>
<td>Open source</td>
</tr>
<tr>
<td>Cameo SOA+ suite (NoMagic MagicDraw)</td>
<td>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) [49][50]</td>
<td>Commercial</td>
</tr>
<tr>
<td>Modelio</td>
<td>Commercial modeling tool with an open-source SoaML designer extension. Code generation is supported</td>
<td>Commercial (partly open-source)</td>
</tr>
<tr>
<td>Tool Name</td>
<td>Description</td>
<td>Availability</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>--------------</td>
</tr>
<tr>
<td>IBM Rational Software Architect</td>
<td>Commercial software architect tool supporting SoaML modeling. To be used in combination with other IBM Rational products [51]. Very complete implementation of SoaML based on its profile. Also support transformations to PSM-level models.</td>
<td>Commercial</td>
</tr>
<tr>
<td>SoaML Eclipse plug-in by Delgado et. al. [52]</td>
<td>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. [52]</td>
<td>Open-source (source-code not available yet)</td>
</tr>
<tr>
<td>SoaML Eclipse plug-in by Ali et. al. [9]</td>
<td>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) [9].</td>
<td>Source-code not available</td>
</tr>
</tbody>
</table>

### 5.2.1 Tool conclusion

Most of the tools in table 5.2 are commercial and some tools are results from research projects. Most tools do not strictly cover SoaML as specified in the SoaML specification [8]. For instance, the syntax is a little bit different, some functionality is missing or some constraints on the models are not implemented.

The IBM Rational Software Architect seems to be the most complete and mature SoaML tool. Therefore we used this tool to model the example case as explained in this chapter. A drawback is its commercial support and licenses.

The open-source ModelPro project looks promising because [50] discusses how they used SoaML models with the MDA. A drawback is that SoaML is implemented as profile, and all documentation is focused on ModelPro in combination with the Cameo SOA+ suite, which is commercial.

The work of Delgado et. al. [52] 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.

An appropriate choice for a possible further study seems to be using EMF as done in [9].
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 probably 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.
Chapter 6

Conclusion

In this literature study we have identified criteria for SOA modeling languages based on all relevant activities of stakeholders in the service life-cycle. In these activities we looked whether MDE can be used to optimize the process and what stakeholders need to model to achieve this. We evaluated current developments from both research work and standardization groups. We selected the most promising development with respect to modeling services and SOA applications at PIM-level, SoaML, and identified which aspects are still missing or not covered in this standard.

During this report we have answered the research question (what is the state of the art of model-driven SOA?). In summary, in the last decade there were several developments with respect to modeling SOA applications. This led to several standardization documents explaining the concepts of SOA. Multiple companies and several research groups defined SOA modeling languages (metamodels) to model SOA applications. The most promising modeling language (for modeling PIM-level SOA applications and services) seems to be SoaML (final version 1.0 March 2012) contributed by many SOA-related companies and researchers.

When we specifically look at the usage of MDE in modeling SOA applications, the literature is still limited. Most research work is focused on the generation of code based on service designs (of their own proposed metamodels), such as in [53] and [20]. Next to this, several companies provide MDE methodologies support the modeling of SOAs. The SoaML specification explains that SoaML is designed in such a way that can be used to support MDA. Because SoaML is a recently proposed language, not much research papers are available yet. However, there is some recent research work available investigating how SoaML can be used in MDA processes such as [9] and [54]. This work is limited and only focuses on a small subset of all possible activities of stakeholders.

Some interesting further areas of research, which are not covered by SoaML or useful to test SoaML for some purpose, are the following:

1. The implementation of SoaML in EMF (Ecore metamodel and graphical editor, such as done in the work of Ali et al. [9]) so that SoaML can easily be tested in a case study and in possible model transformations, using the benefits of the EMF framework. An example, that could be realized with such a implementation is the generation of PSM models (e.g. WSDL code for the service developer) based on SoaML models.
2. The modeling of non-functional constraints and policies is possible with SoaML, but the specification does not give any details on how we can exactly do this. We can look in which ways this is supported in existing SoaML modeling tools with a case study. It may also be interesting to investigate how this can be enforced at runtime.
3. Another related but broader idea is how can we enforce service contracts specified in SoaML. Service contracts in SoaML contain all the service information, choreography, and
any other terms and conditions of a certain service. When we implement a SOA application based on the SoaML design we want to ensure that the contract is met. For instance, how can we know that the choreography is used properly by all participants, and that the interfaces are implemented correctly?

4. We could also look how service discovery can be realized in practice based on SoaML models. Can we extract suitable service descriptions models from SoaML. And which service registries are suitable to work with these descriptions?

5. The SoaML is open about the way behavior of services can be modeled. It states that any UML behavior, e.g. interaction models, activity models or state machines, can be used. A case study might give more information which kind of models are most suitable with respect to the context of projects.
References


REFERENCES


55


REFERENCES

