Managing Service Dependencies in Service Compositions

Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von

M.Sc. Matthias Winkler
geboren am 14. August 1979
in Dresden

Betreuender Hochschullehrer:
Prof. Dr. rer. nat. habil. Dr. h.c. Alexander Schill

Gutachter:

Prof. Dr. rer. nat. habil. Dr. h.c. Alexander Schill	TU Dresden
Prof. Dr. Jorge Cardoso	University of Coimbra

Tag der Einreichung: 09.04.2010
Tag der Verteidigung: 21.06.2010

Dresden im Juni 2010

Acknowledgment

The research for this thesis was executed as part of my work at SAP Research Center Dresden in collaboration with the Institute for Systems Architecture at the TU Dresden. I would like to thank Prof. Dr. rer. nat. habil. Dr. h.c. Alexander Schill for the supervision of this thesis, especially for the discussions and comments that helped to improve the quality of this thesis. Furthermore I would like to thank Prof. Dr. Jorge Cardoso and Prof. Dr. rer. pol. Susanne Strahringer for their feedback and encouragement.

I would also like to thank Dr. Thomas Springer and Dr. Steffen Göbel for the many discussions we had and for helpful comments, furthermore Sebastian Schneider, Edmundo David Trigos, and Anna Grebeneva for their work as interns and thesis students, as well as my wife Viktoria, Konrad Voigt, Matthias Heinrich, Birgit Grammel, and Eldad Louw for proofreading my papers and my thesis.

During my time at SAP Research Center Dresden I was part of the EMODE and THESEUS TEXO project teams. During these projects I had the chance to work with some great colleagues, who not only put a lot of enthusiasm into our daily work, but also created a very nice working environment. Special thanks go to Matthias Heinrich and Konrad Voigt for bringing lots of fun into our daily work.

Finally, I would like to thank my wife and my whole family for supporting me during these last years and for understanding that my time was often occupied by work. I especially like to thank my parents Maria and Christian who helped me in many ways.

Dresden, April 2010

Matthias Winkler

Contents

List of Figures

List of Tables

List of Algorithms

Abstract

In the Internet of Services (IoS) providers and consumers of services engage in business interactions on service marketplaces. Provisioning and consumption of services are regulated by service level agreements (SLA), which are negotiated between providers and consumers. Trading composite services requires the providers to manage the SLAs that are negotiated with the providers of atomic services and the consumers of the composition. The management of SLAs involves the negotiation and renegotiation of SLAs as well as their monitoring during service provisioning. The complexity of this task arises due to the fact that dependencies exist between the different services in a composition. Dependencies between services occur because the complex task of a composition is distributed between atomic services. Thus, the successful provisioning of the composite service depends on its atomic building blocks. At the same time, atomic services depend on other atomic services, e.g. because of data or resource requirements, or time relationships. These dependencies need to be considered for the management of composite service SLAs.

This thesis aims at developing a management approach for dependencies between services in service compositions to support SLA management. Information about service dependencies is not explicitly available. Instead it is implicitly contained in the workflow description of a composite service, the negotiated SLAs of the composite service, and as application domain knowledge of experts, which makes the handling of this information more complex. Thus, the dependency management approach needs to capture this dependency information in an explicit way. The dependency information is then used to support SLA management in three ways. First of all dependency information is used during SLA negotiation the to ensure that the different SLAs enable the successful collaboration of the services to achieve the composite service goal. Secondly, during SLA renegotiation dependency information is used to determine which effects the renegotiation has on other SLAs. Finally, dependency information is used during SLA monitoring to determine the effects of detected violations on other services.

Based on a literature study and two use cases from the logistics and healthcare domains different types of dependencies were analyzed and classified.

The results from this analysis were used as a basis for the development of an approach to analyze and represent dependency information according to the different dependency properties. Furthermore, a lifecycle and architecture for managing dependency information was developed. In an iterative approach the different artifacts were implemented, tested based on two use cases, and refined according to the test results Finally, the prototype was evaluated with regard to detailed test cases and performance measurements were executed.

The resulting dependency management approach has four main contributions. Firstly, it represents a holistic approach for managing service dependencies with regard to composite SLA management. It extends existing work by supporting the handling of dependencies between atomic services as well as atomic and composite services at design time and during service provisioning. Secondly, a semi-automatic approach to capturing dependency information is provided. It helps to achieve a higher degree of automation as compared to other approaches. Thirdly, a metamodel for representing dependency information for SLA management is shown. Dependency information is kept separately from SLA information to achieve a better separation of concerns. This facilitates the utilization of the dependency management functionality with different SLA management approaches. Fourthly, a dependency management architecture is presented. The design of the architecture ensures that the components can be integrated with different SLA management approaches. The test case based evaluation of the dependency management approach showed its feasibility and correct functioning in two different application domains. Furthermore, the performance evaluation showed that the automated dependency management tasks are executed within the range of milliseconds for both use cases.

The dependency management approach is suited to support the different SLA management tasks. It supports the work of composite service providers by facilitating the SLA management of complex service compositions.

1 Introduction

During the last years Service-Oriented Architecture (SOA) as an architectural style for building enterprise applications has become an important topic. The SOA reference model by OASIS describes SOA as "a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains" [MLM+06]. One important example is SAP's Enterprise Service-oriented Architecture (ESOA) [Mal07]. Applications are built from reusable components - services - which are stored in an enterprise service repository.

Within the TEXO project [Pre08] the vision of the Internet of Services (IoS) [HAW08] is explored. In the IoS vision, services are tradable goods that can be offered by service providers and consumed by service consumers. Internet service marketplaces act as a platform for trading services where consumers and providers of services are brought together to engage in business interactions [CVW08].

Services are offered via a marketplace where consumers can search for services that are suitable for their needs. Once a consumer has found a suitable service it will negotiate a formal contract with the provider of the service. Such a contract, which is called a Service Level Agreement (SLA), specifies, among other aspects, which service is provided, how provisioning takes place, and what the rights and duties of the consumer and the provider are. Any such statement is called a Service Level Objective (SLO). Multiple SLOs make up the SLA. Based on this contract the service provisioning takes place. During the provisioning of the service, constant monitoring of

the service execution is necessary to make sure that the single SLOs within the SLA are met.

1.1 Background Information

When services are available as tradable goods in a service marketplace, they can be purchased either for direct utilization by consumers or to be integrated into a service composition. The creation of service compositions enables the development of more complex services by reusing existing services. Service compositions can then be offered as services on the service marketplace. Consumers of a service may not be aware of the fact that they are consuming a composite service.

One important aspect of offering composite services is the requirement to manage the different services in such a way that they fulfill the overall goal of the composition. This means that achieving the composite service goal depends on the different services forming the composition. Achieving the composite service goal also depends on the proper collaboration between the composed services. This collaboration implies that the composed services also have dependencies among them. Examples are the requirement of one service for certain goods or data, which are made available by a different service, or the need to finish the execution of one task before another one starts.

To ensure a proper collaboration, the composite service creator negotiates SLAs with the providers of services contributing to the composition as well as with each customer of the composite service. The composite service provider needs to ensure that all these SLAs enable achieving the goal of the composite service and that the fulfillment of the different contracts can be achieved. As a simplification of terms, the services, which are used to form the composition, are considered as atomic services throughout this thesis. A SLA, which was negotiated for a composite service, is called composite SLA (CSLA). SLAs negotiated for the services forming the composition are called atomic SLAs (ASLA). These terms are adopted from the work done by Ludwig [Lud09].

Throughout the lifecycle of a composite service the CSLA and ASLAs need to be managed [Lud09]. This includes the negotiation, renegotiation, and monitoring of SLAs. This is a challenging task because of the dependencies that exist between the different services and thus between the different SLAs. Existing approaches to manage SLAs mostly focus on the handling of individual SLAs [LDK04, BPSMS07, AF08, FTS06]. However, some first work exists, which considers service dependencies in a limited fashion. Ludwig [Lud09] and Bodenstaff et al. [BWRJ08] provided first approaches

to the handling of dependencies between atomic services and the service composition. Karänke et al. discuss SLA management of service resources in value chains [KMK09]. The handling of dependencies often requires the explicit modeling of dependency information. Few approaches enable the automatic discovery of some dependency information (e.g. [Lud09]). In order to improve the handling of dependencies, further steps are necessary to cover additional types of dependencies and to enhance the automatic handling of dependencies.

1.2 Problem Statement

The management of service level agreements of service compositions is a challenging task because of dependencies that exist between the different services. However, current approaches for managing SLAs in service compositions only provide limited or no support for the handling of dependencies. These approaches are limited with regard to the supported dependency types as well as the efficient handling of dependency information. Most approaches require explicit modeling of dependency information, i.e. the level of automation is rather low. Furthermore, they have a tight integration of SLA management and dependency management, which makes it more difficult to reuse dependency handling with other SLA management approaches.

1.3 Motivation

The goal of this thesis is to enhance the functionality of composite service SLA management by providing advanced support for the handling of service dependency information. The handling of dependency information will enable the determination of effects of problems created by one service. Such problems can affect the composite service as well as the atomic services that are a part of the composition. This will enable providers of composite services to better manage the composition:

1. Reduction of analysis overhead for users: The solution will point users to relevant SLAs and SLOs which need to be considered for further handling. The user will then be able to determine possible steps to handle the situation. It relieves the composite service provider from having to do a full analysis of the composite service, i.e. searching for SLAs that are affected by violation or renegotiation events.

2. Reduced chance of errors: Whenever different SLA management tasks require a specific handling, there is a chance that the person

responsible for handling the event misses something, e.g. forgets to adapt a contract with an involved party. Automating parts of this process will help to reduce the chance for errors.

3. Faster handling of occurring violation events and renegotiation requests: Information regarding all services, which are affected by a violation event or renegotiation request, is available immediately. This helps to speed up the process of triggering a suitable reaction.

Moreover, improving the dependency handling functionality will facilitate the work of the composite service provider. A partial automation of the process of capturing dependency information reduces the necessary work, which is introduced when dependency information is considered for SLA management. Enhancing the scope of considered dependencies improves the management of SLAs.

1.4 Thesis Scope and Research Questions

It is the goal of this thesis to develop an approach for the management of dependencies between services in service compositions. The focus of the approach is to utilize dependency information to support the composite SLA management. In this thesis dependency information is used to support the following SLA management activities:

- SLA negotiation: ensure that negotiated contracts enable a proper collaboration between the different services as well as the fulfillment of all negotiated contracts
- SLA renegotiation: evaluate the impact of changing one SLA on other SLAs
- Monitoring: determine the effects of service failure on other services in the composition

In order to do so, an approach is proposed where the dependencies between services in a composition are analyzed in a semi-automatic fashion at design time. They are captured in a dependency model. The dependency model is used at design time to validate the compatibility of the SLAs under negotiation. At runtime the dependency model is applied in order to determine services that are affected by events such as service failure or SLA renegotiation. This information supports providers of composite services to determine possible steps in order to react to the different events. The determination of resulting actions is not part of the thesis scope. The

dependency management approach is integrated with SLA management functionality.

While there is some related work, which deals with the handling of service dependencies in the light of composite SLA management, this thesis extends the state-of-the-art by answering the following core research questions:

1. **What would be a suitable method for managing dependencies in service compositions at design time and during service provisioning?** The handling of service dependencies covers different tasks including the capturing of dependency information and the evaluation to support composite SLA management. The different tasks occur during different stages of the service lifecycle. A suitable approach needs to coordinate the different dependency and SLA management tasks.

2. **How can dependencies between services be determined in an efficient manner?** Information regarding dependencies between different services is implicitly contained in the ASLAs and the CSLA as well as the process description of the composite service. Further information may be available as domain knowledge. Capturing this information in an explicit representation facilitates the handling of dependencies during SLA management. The modeling of dependency information is one way to achieve that. However, automating this process or parts of it would improve its efficiency.

3. **What is a suitable way to represent dependency information?** In order to facilitate the handling of dependencies, a suitable representation of dependency information is necessary. A representation is suitable if it captures all the information required for the management of dependencies in service compositions with regard to managing composite service SLAs. At the same time a separation of dependency information from service and SLA information helps to ensure the separation of concerns and provides sufficient independence of dependency management and SLA management. This will help to facilitate changes of one aspect, while minimizing effects on the respective other aspects. For example, changes to the languages for describing services or expressing SLAs should not require changes to the model for describing dependency information.

4. **What would be a suitable architecture to support the handling of service dependencies?** It is important that the dependency management functionality is integrated with components for SLA management. However, they should also be sufficiently decoupled from the

specific SLA management approach to provide flexibility to work together with different SLA management approaches.

1.5 Research Methodology

In order to achieve the goals of this thesis, the design science paradigm was applied. Hevner et al. [HMP04] describe design science as "...a problem solving paradigm". The goal of design science research is the creation of an innovative artifact that solves a specific problem. Hevner et al. provide seven guidelines for good design science research [HMP04]:

1. **Design as an artifact:** The research goal is the creation of an innovative artifact.
2. **Problem relevance:** The artifact is created to solve a specific problem.
3. **Design evaluation:** A thorough evaluation needs to show that artifact fulfills the purpose for which it was designed.
4. **Research contributions:** the artifact solves an unsolved problem or provides a better solution than existing work. The novelty of the artifact provides a contribution to science.
5. **Research rigor:** During the design and evaluation process rigorous methods are applied.
6. **Design as a search process:** The solution is created in an iterative design process of artifact generation and testing.
7. **Communication of research:** The research results are presented to technical as well as management-oriented audiences.

The research work for this thesis was executed based on these guidelines. (1) The goal of this thesis was the creation of a method for managing service dependencies in service compositions (see Section 4.4). A prototype implementing this method was developed (see Section 5.2). The method as well as the prototype are both artifacts in a design science perspective. Further artifacts created as part of this thesis include a metamodel for representing dependency information and methods for the discovery, validation, and evaluation of service dependencies. (2) The different artifacts were developed to solve the problem of missing support for dependency handling for composite SLA management (see Section 1.2). (3) In order to demonstrate the suitability of the different artifacts to solve this problem, they were evaluated based on a detailed set of test cases (see Section 5.3). The different

test cases were developed based on two use cases from the logistics and healthcare domain. (4) The developed artifacts extend existing work by improving several shortcomings (see Section 6.1). Thus, the results of this thesis provide a novel and valuable contribution to the existing work. (5) These results build on existing work in the area of SLA management, workflow description, and dependency handling. A formal validation of the research results was executed. The suitability of the solution was demonstrated based on different test cases. A performance evaluation showed the applicability of the approach. (6) The research for this thesis was executed in an iterative approach. Early artifacts were tested regarding their suitability to solve the problem. They were then improved based on the test results. (7) The results of this research work have been published in the scientific community. They have also been communicated to project partners and the project sponsor.

1.6 Outline of this Document

This thesis is structured as follows:

- Chapter 2 provides information about the Internet of Services vision and introduces the topics of SLA management as well as service dependencies. Furthermore, two scenarios are presented, which serve as guiding examples for this thesis. Finally, different requirements are discussed for the dependency management approach.
- Chapter 3 provides an overview of the state-of-the-art of SLA management and the handling of dependency information. Different approaches for the handling of dependency information are discussed and compared to the proposed dependency management approach. Shortcomings of existing solutions are outlined.
- Chapter 4 presents the dependency management approach. A lifecycle for handling dependency information is outlined and the different phases are detailed. Furthermore, a conceptual architecture is presented, which realizes the different lifecycle phases.
- Chapter 5 presents the validation of the different concepts developed as part of this thesis.
- Chapter 6 summarizes the thesis and discusses the main achievements. Furthermore, an outlook on future work is presented.

2

Foundations and Requirements Analysis

This chapter provides background information regarding services in the IoS vision (Section 2.1), the handling of service level agreements (Section 2.2), and service monitoring (Section 2.3). Furthermore, two use cases from the field of logistics and healthcare are introduced (Section 2.4). Based on the two use cases a first introduction to service dependencies is provided and requirements for managing service dependencies are derived (Section 2.5).

2.1 Services in the Internet of Services

In the IoS service consumers and providers engage in business interactions by offering and consuming services. While the internet serves as a medium to offer and search for services, the provisioning of these services is not necessarily via the internet. Instead, services may be provided based entirely on human tasks, as a combination of automated and human tasks, or fully automated. According to [CVW08] services (also referred to as business services), e-services, and Web services can be distinguished. Business services are described by [BGO04] as "...business activities that often result in intangible outcomes or benefits; they are offered by a service provider to its environment." An example would be the painting of a house or the transport of goods from one location to another.

E-services are described by [HBCS03] as "...a collection of network-resident software services accessible via standardized protocols, whose functionality can be automatically discovered and integrated into applications or composed to form more complex services." An example for an e-service would be a B2B service via EDI [OEC06].

Finally, Web services are considered to be e-services which are accessed using standard web based protocols [CVW08], e.g. HTTP. Examples would be SOAP or RESTful Web services providing a credit card check. As such, Web services are a subclass of e-services, which are again a subclass of (business) services. Within the IoS vision (business) services are offered on service marketplaces. This also includes e-services and Web services. Throughout this thesis the term service is used to talk about any kind of service including e-services and Web services.

Figure 2.1 illustrates how providers and consumers of services interact via a marketplace. Providers offer their services on a marketplace in order to make them available for consumers.

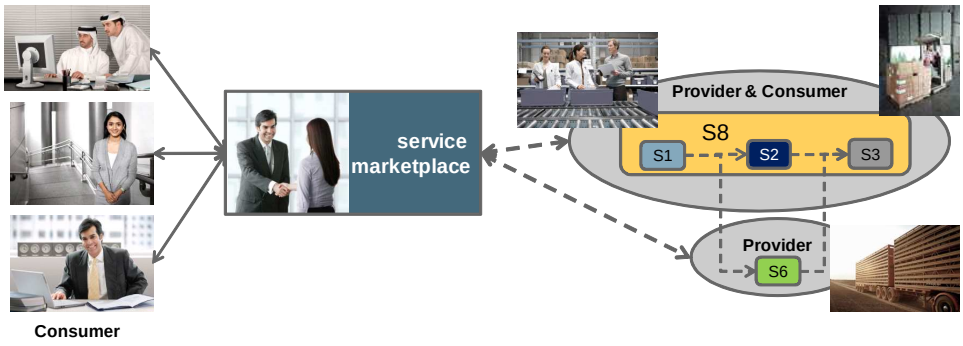

Figure 2.1: Trading services via a service marketplace

2.1.1 Composite Services

Providers of services may also act as consumers of other services. This is the case when the consumed service is integrated with other services to form a service composition and the composition is sold, again via the marketplace.

When services are integrated into a composition, they form a business process. The process describes how the different services collaborate in order to achieve the composite service goal. Cardoso and Lenič describe a business process as "...a set of activities that represent all the alternative methods of performing the work needed to achieve a business objective..."

[CL06]. The business process is described by a workflow graph. According to Vanhatalo et al. a workflow graph $G = (N, E)$ is described by a set of nodes (N) and a set of edges (E) connecting these nodes [VVL07]. The set of nodes contains *start* and *end nodes*, a number of activities, and a set of *fork*, *join*, *decision*, and *merge* nodes.

Different notations for representing workflows exist. Two examples are petri nets and the Business Process Modeling Notation (BPMN). According to [Aal97] a petri net is a 3-tuple (P, T, F) where P represents a set of places, T represents a set of transitions, and F represents a set of arcs connecting places with transitions and transitions with places. The state of a petri net is described by its marking M, i.e. the assignment of tokens to its places. The firing of a transition leads the petri net from one state to another one. A sequence of transitions leading from M_1 to M_n is called a firing sequence [Aal97]. Petri nets have a well defined mathematical base. For that reason they are applied by various approaches to proof the formal correctness of workflows [Aal97, RPU+07].

The BPMN Specification is a standard, which defines a graphical notation for the modeling of workflows [Gro09]. It was developed by the Object Management Group (OMG). BPMN is targeted at business users. Sample BPMN elements are shown in the following list. More details can be found in the BPMN Specification [Gro09].

- The *Start Event* and *End Event* represent the starting and the end point of a workflow.
- The *Task* element is used to represent the different activities of a workflow.
- The *Sub-Process* element enables the specification of sub-processes within a workflow containing a set of tasks.
- A *Sequence Flow* connects activities and gateways and thus shows the order of activity execution.
- *Exclusive, Inclusive, and Parallel Gateway* (also XOR, OR, AND) allow the splitting and joining of the sequence flow in a workflow.

The handling of composite services and their underlying workflow is an important aspect of this thesis. As a base for the development of the concepts of this thesis the BPMN notation was used because it is now well known and widely accepted in the business community. More precisely, only a small subset of the BPMN elements was used including start and end events, tasks, sub-process, sequence flow, AND/OR/XOR splits and joins. Furthermore, the workflow descriptions are limited to one start and one end event. Loops are not supported.

The formal mathematical base of petri nets, which is useful e.g. for workflow verification, is not required for this thesis. However, it is important to note that the concepts are not designed for or limited to BPMN. It could be replaced by other workflow modeling languages.

2.1.2 The Service Lifecycle

In order to enable the vision of tradable services, it is necessary that new services are developed according to the needs of future consumers. Services are then offered so that consumers may find and consume them. The consumption of services needs to be monitored. There are a number of important phases that a service goes through, forming the lifecycle of a service. The service lifecycle of the TEXO project [OBB09] comprises five different phases. It is presented in Figure 2.2.

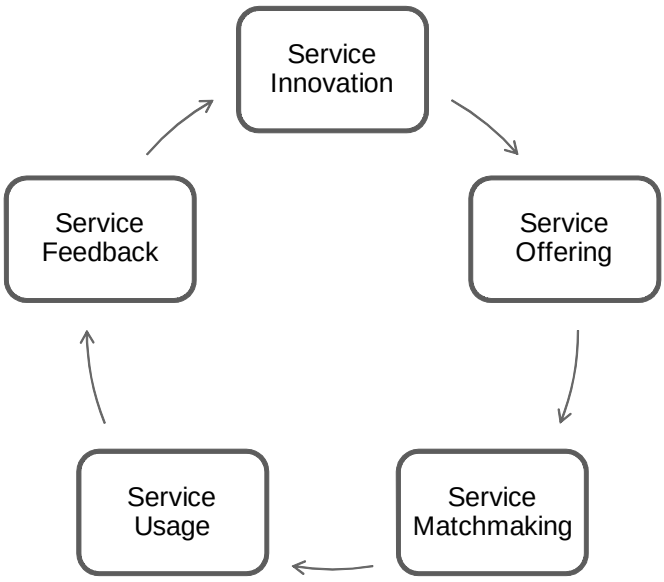

Figure 2.2: The TEXO service lifecycle

1. **Service Innovation:** During the service innovation phase new ideas for services are collected and evaluated. This can be done e.g. by the identification of relevant interests or needs of potential consumers in internet forums and blogs and their subsequent evaluation [SFRM08].

2. **Service Offering:** During the service offering phase a new service is developed. The service functionality is realized by e.g. by a service composition of different (atomic) services. The description of the

service properties, which are needed during later phases of the lifecycle, is created. This information includes quality of service (QoS) information, legal aspects, and pricing information, to mention only a few (see Section 2.1.3 for more details). The new service can then be registered and offered in a service marketplace.

3. **Service Matchmaking:** Once a service is registered in a service marketplace, users can find the service. During the service matchmaking phase users search for services according to different criteria (e.g functionality, pricing information, rating). Different approaches for searching services are applied based on the service description and the search query of a user. The search results are presented to the user, who can then explore the resulting set of services and select the service which best fits his needs. To use the service a service level agreement needs to be negotiated between the consumer and the provider of the service.

4. **Service Usage:** During the service usage phase the service is executed. Monitoring data is collected to assure that the negotiated service level agreement is respected by all parties.

5. **Service Feedback:** After the execution of a service the user may provide feedback about the service execution. This information can be used for the rating of a service, which informs potential future users about the satisfaction of prior consumers of the service. It can also serve as a base for future improvements of the service by the provider.

2.1.3 The Need for Service Description

The trading of services in marketplaces involves, among other steps, the offering and search for services. To enable providers to offer their services and consumers to find them, it is necessary to describe them in a suitable way. Traditionally service description is done using the Web Service Description Language (WSDL) [CMRW07], which enables the specification of the service functionality. Different researchers have stated, though, that there is a strong need for service description including not only its functionality but also its non-functional properties [O'S06, BDB05]. Non-functional properties include quality of service (QoS) aspects, support for legal issues (e.g. terms of use), as well as pricing and payment models for service consumption. The quality of service describes the performance of a service using a set of QoS parameters [Lud09]. Typical examples of QoS parameters include reliability, execution time, throughput, availability, and encryption level [BKLW95, CMSA04, JRGM04, Lud09]. Throughout this thesis the term QoS is often used instead of always listing a selection of different

QoS parameters. However, to provide more concrete examples when explaining the different concepts of this thesis, the QoS parameters response time, availability, throughput, and encryption level are used. They were taken from the COSMA approach, which forms the base for the handling of QoS information in the dependency management approach presented in this thesis.

Besides supporting the offering and search for services, the service description can serve as a basis for negotiating service level agreements and monitoring the service at the time of provisioning.

The Unified Service Description Language

The Unified Service Description Language was developed as a language for describing business, technical, and operational aspects of services [CWV09]. It covers aspects that are common for most services independent of their nature (rather technical or more business oriented).

There are a number of existing approaches to describe services. Many of these approaches cover mainly technical service aspects. The Web Service Description language (WSDL) [CMRW07] allows the description of service interfaces including their input and output parameters, the communication protocol for accessing the service, as well as the endpoint where the service can be found. SA-WSDL [FL07] uses semantic annotations to WSDL elements to describe the meaning of input and output parameters as well as faults and thus provides an understanding of the functionality of the operations. These formal annotations allow machines to reason about the service. Zschaler [Zsc04] presents an approach for the semantic description of quality of service aspects. All these approaches lack the capability to describe business related service aspects.

While there are many different approaches to describing technical service aspects, there is also some work available which focuses on the description of non-functional service properties with a strong focus on the business perspective. O'Sullivan provides a comprehensive taxonomy of non-functional service properties, covering business aspects including pricing and payment information, legal aspects, and quality of service, to mention a few [O'S06]. Another service description approach, the Publicly Available Specification 1018 [MH01], was created as a specification for describing services for tender offers. It covers non-functional service aspects as well as a functional service description.

The special objective that distinguishes USDL from other service description approaches is the combination of the technical and the business perspective, while other approaches are rather limited in their scope. At the

same time USDL only covers the most important aspects which are necessary to describe services independent of their nature (business vs. technical service) or their specific application domain.

USDL was designed with the objective to trade services in service marketplaces. Thus, it was important to consider a number of service aspects to be described. The following list summarizes the three USDL perspectives and the descriptive aspects that are covered.

Business perspective:

- **Provider and consumer information:** Information about the service provider and a profile of targeted consumers are provided.
- **Service level:** The service level includes QoS information (i.e. performance, dependability and security parameters), and a service rating.
- **Legal information:** The rights and obligations of consumers and providers are described, as well as penalties that occur in the case of any party not respecting their obligations or the other party's rights.
- **Pricing and payment:** Information regarding different pricing options as well as payment methods for service usage are described.
- **Interaction:** The means for invocation and execution of services are described. They may be realized in technical or non-technical ways.

Operational perspective:

- **Functional description:** The functional description allows describing what the service does, e.g. by using a classification scheme.
- **Operations:** The operations of a service are described by interfaces with input and output parameters.

Technical perspective: The description of how a service is invoked and how service execution takes place is achieved via the specification of different protocols to be used for this interaction.

2.2 Service Level Agreements

In the IoS vision *service level agreements (SLA)* provide a formal contract regulating the provisioning and consumption of services between the service provider and consumer. This section provides a formal definition for the term SLA and describes the lifecycle of SLAs in the Internet of Services.

2.2.1 Defining Service Level Agreements

Various definitions of service level agreements exist. Modica et al. describe a SLA as "...a formal negotiated agreement between a Service Provider and a Service Customer ..." [MTV09]. Bianco et al. [BLM08] describe a SLA as "...a specification of the verifiable quality characteristics that the service will provide". A SLA is formally negotiated between a provider and a consumer of a service prior to service provisioning and consumption. It describes the provided service and a number of quality attributes that are guaranteed during service provisioning. It also specifies information regarding the different parties involved in service provisioning and consumption as well as their rights and obligations concerning the provisioning and consumption process. Finally, pricing information and penalties for not conforming with the negotiated SLA are specified.

SLAs are used to regulate the processes of service provisioning and consumption. In order to verify that an SLA is met, it needs to be monitored. This is realized by monitoring and evaluating the single service level objectives of a SLA.

SLOs are important building blocks of SLAs. They are the single measurable guarantees of a SLA. Service level agreements contain one or more service level objectives. SLOs refer to a quality attribute called key performance indicator (KPI), such as the response time of a service. They specify a value or value range and a unit (e.g. milliseconds) for the KPI along with an operator (e.g. greater than, equals, ...) to express the expected value of the KPI. Examples for simple service level objectives are listed below:

$$responseTime \leq 500ms \tag{2.1}$$
$$availability \geq 95\% \tag{2.2}$$

In many cases SLOs specify more refined information, e.g. that a KPI will be met in 95% of all cases, or a time frame is given within which the KPI is guaranteed and can be evaluated.

$$responseTime \leq 500ms \quad - \quad 95\%executions \tag{2.3}$$
$$availability \geq 95\% \quad - \quad 1month \tag{2.4}$$

Rosario et al. proposed an alternative approach to specifying SLOs using percentage values (quantiles). They argue that the usage of probability

functions to express values for KPIs will enable more optimistic specification of guaranteed terms [RBHJ07].

The general building blocks of a SLA, as described above, are presented in Figure 2.3.

Figure 2.3: Building blocks of a service level agreement

Since SLA describes the service and a number of different aspects of provisioning, it can be seen as a special instance of a service description, namely one that describes a single instance of a service at runtime. The negotiated parameters are based on a more general service description, e.g. based on USDL (see Section 2.1.3).

2.2.2 A SLA Lifecycle for the Internet of Services

The work presented in this thesis is based on a lifecycle for handling service level agreements (see Figure 2.4) [WS09]. It was developed based on the work presented in [BLM08]. A few adaptations were made. A first change was the integration of support for SLA renegotiation. For that reason the SLA negotiation phase was extended to allow for SLA renegotiation in case the SLA needs to be adapted.

A second adaptation was the integration of a loop from the *Execution & Monitoring* phase to the *SLA (Re-)Negotiation* phase to enable the adaptation of a SLA based on monitoring information. This enables a composite service provider to react to problems while the service is still in execution. This is only possible for limited SLA aspects, but it is nevertheless an important aspect of SLA management for business services. SLA renegotiation is only successful if both involved parties agree.

Finally, the assessment phase presented in [BLM08], which enables the long-term evaluation of SLA fulfillment as well as strategic considerations,

was removed from the SLA lifecycle. It is rather seen as an aspect of service management and should be handled as part of the *Service Feedback* phase of the service lifecycle.

The following phases are the building blocks of the adapted SLA lifecycle:

1. SLA Template Development: The SLA template forms the base for SLA negotiation. It is created as part of the service engineering process and is made available for SLA negotiation during the deployment of a service.

2. SLA (Re-)Negotiation: This phase supports the initial negotiation of a SLA based on a SLA template as well as the adaptation of an existing SLA. During this phase the different quality of service parameters, pricing and legal information, as well as other details about service execution are negotiated between the service provider and the consumer.

3. Preparation: During the preparation phase the information regarding the newly negotiated SLA or the update through renegotiation is propagated to relevant components (e.g. monitoring components). They will then retrieve the new SLA and prepare the service execution and monitoring.

4. Execution and Monitoring: The execution of a service is triggered once a consumer with a valid SLA requests the service. The monitoring of the service execution and the validation of the SLA and its contained service level objectives occurs at the same time. Once information regarding SLO violations is available, further actions can be initiated, e.g. the adaptation of the service.

5. Termination and Decommissioning: SLAs are usually valid for a predefined period of time (e.g. single service call, one month). Once this period is over, the SLA cannot be used as the basis for service execution any more. If the provider of a service decides to discontinue the service offering under the current terms, the SLA template for the service is removed. This way no new contracts can be negotiated based on that specific SLA template.

2.3 Monitoring services

Services that are traded on a service marketplace undergo a service lifecycle. In [BRS+08] a service lifecycle of four phases is described: service

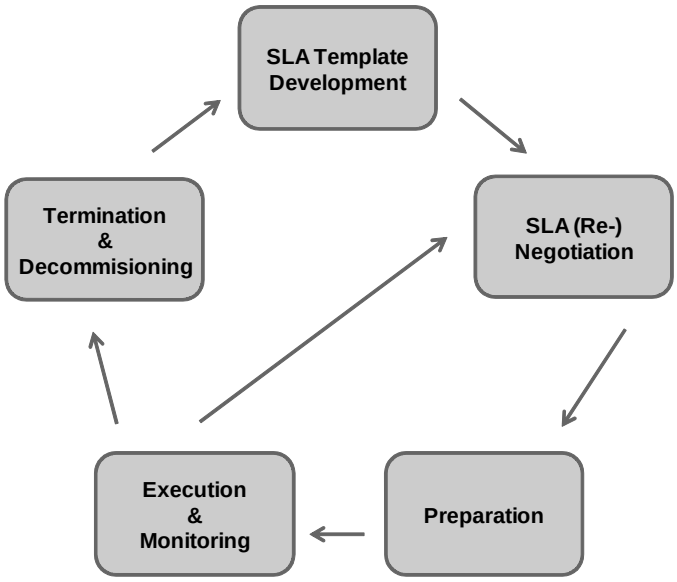

Figure 2.4: The SLA lifecycle

modeling, service usage preparation, service usage, and service usage post-processing. One important task during service usage is the monitoring of guarantees that the involved parties agreed upon in the SLA.

The IT Infrastructure Library V3 (ITIL V3) is a framework for service management, which was developed by the Office of Government Commerce. It is currently available in Version 3. ITIL V3 provides best practices for managing IT services which support the execution of business processes. ITIL V3 describes a service lifecycle [BVGM07] consisting of five phases: service strategy, service design, service transition, service operation, and continual service improvement. Monitoring is seen as one important part of the service operation phase where the operation of services is managed, in order to assure stable service provisioning. To ensure good quality and enable the optimization of the provisioning process ITIL emphasizes the need to monitor the KPIs [BVGM07].

There are a number of reasons why monitoring is a very important task in the service lifecycle:

- Pay for what you get: Consumers of services have certain quality expectations regarding service provisioning. Service providers may, for example, offer the same service with different quality attributes for different prices. Once a service consumer has decided for a certain quality, payment for the service will depend upon reaching the agreed quality. Otherwise the amount due may be reduced. In or-

der to determine if the guaranteed quality was achieved a suitable monitoring strategy is necessary.

- Trust between service consumer and provider: On a service marketplace providers and consumers of services are brought together on a national or international scale. This implies that in many cases they will not know each other. Thus, it is very important to establish trust between the parties involved in a business interaction. Monitoring can help to establish trust by making the achieved quality as well as possible problems visible to all involved parties.

- Guarantee of quality: When services are provided by an external party, the consumer cannot control the quality of service provisioning. This imposes a high risk especially when a service consumer acts as a service provider at the same time by making the consumed service available to consumers within a service composition. Therefore, it is important that service consumers have a good insight into the quality aspects of the service provisioning procedure. This enables them to manage risks on their side.

- Continuous improvement of service quality: Services are offered to consumers under a certain quality. While it is important to meet quality goals specified in a SLA, this is not sufficient. ITIL V3 describes the Continual Service Improvement phase as part of the service lifecycle [BVGM07]. The goal is to continually improve service quality over time. Monitoring can deliver the necessary information serving as the basis for decisions regarding what needs to be improved. To achieve that, the right performance figures need to be monitored. Also, in [MPA+07] the authors emphasize the importance of business process monitoring to ensure competitive advantage as well as standards compliance.

2.4 Use Cases

As guiding examples for this thesis two composite services from the logistics and healthcare domains will now be introduced. A composite service provider creates the underlying business process from a number of services offered by other service providers on a service marketplace. The two scenarios will help to illustrate the vision of tradable services and provide a better understanding of the problem space of this thesis, namely the management of dependencies between different services within a service composition. Furthermore, they will be used to validate the concepts developed as part of this thesis.

2.4.1 Use Case: Logistics

Logistics is described as the "…process of planning, implementing and controlling the efficient, cost-effective flow and storage of raw materials, in-process inventory, finished goods, and related information from point of origin to point of consumption for the purpose of conforming to customers' requirement" by the American Council of Logistics Management [Bri].

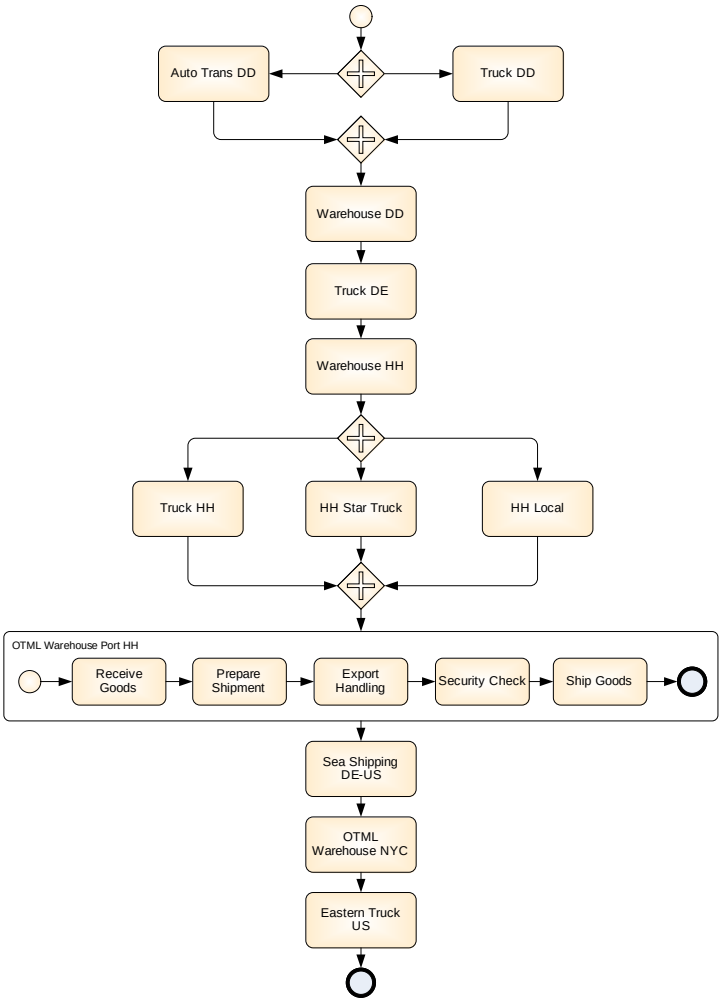

Figure 2.5: Logistics scenario

The scenario describes a multimodal process for transporting goods (see Figure 2.5) between Dresden, Germany and Philadelphia, US. Different logistics services (e.g. goods transport as well as storage) are offered by different service providers on the *i-o-Log marketplace* for logistics services. The transportation process is organized and managed by *OnTheMove Logistics*

(OTML). It is partially realized by *OTML* as well as by services contracted via the *i-o-Log marketplace*.

This general scenario of an IoS-based logistics marketplace, the different stakeholders as well as the transportation process are fictional and were designed specifically for the purpose of illustrating different research aspects of the internet of services vision. However, different steps of the transportation process are based on scenarios as they occur in reality.

OnTheMove Logistics is a provider of national and international logistics solutions including the transportation and storage of goods (e.g. *OTML TransWorld*). It offers its services via the *i-o-Log marketplace*. *Dresden Spare Parts (DSP)*, a manufacturer of spare parts for cars, received an order from a customer in Philadelphia, US. In order to ship goods to the customer, DSP searches for a provider of logistics services, who can realize the shipment. DSP selects the *OTML TransWorld* service.

After receiving the precise information for transportation from *DSP*, *OTML* selects suitable services and creates a logistics process realizing the transport. Next to its own services, e.g. *Warehouse HH* and *Truck DE*, it also subcontracts other services. This is due to the fact that certain services are not available from *OTML* (e.g. goods transport via ship). At the same time it does not have enough local capacity to support the transportation process, e.g. for the transportation within the cities of Dresden and Hamburg. OTML subcontracts services via the same *i-o-Log marketplace*, where it also offers its own services to customers. When the setup of the transportation process is finished, the goods are shipped.

The transportation process starts with the pickup of goods at the site of *Dresden SpareParts*. The two services *AutoTrans DD* and *Truck DD* pick up the goods and deliver them to a warehouse in Dresden (*Warehouse Dresden*) where they are repackaged and made available for further transport with a larger truck. Service *Truck DE* is responsible for taking the goods from Dresden to a warehouse in Hamburg, where they are distributed to smaller trucks for further transport to the port of Hamburg.

The transport of the goods between *Warehouse HH* and *OTML Warhouse Port HH* is realized by three local transportation services (*Truck HH*, *HH Star Truck*, *HH Local*). In the port the goods are received by *OTML Warehouse Port HH* and prepared for further shipment. The goods have to undergo an export handling procedure and a security check. If all checks are passed successfully the goods can be transported to New York City via ship (*Sea Shipping DE-US*). They are further processed by *OTML Warehouse NYC*. The last part of the transport is realized by truck between NYC and Philadelphia (*Eastern Truck US*).

2.4.2 Use Case: Healthcare

The second use case describes a composite healthcare service (see Figure 2.6), which was inspired by a healthcare workflow presented in [RBFD01].

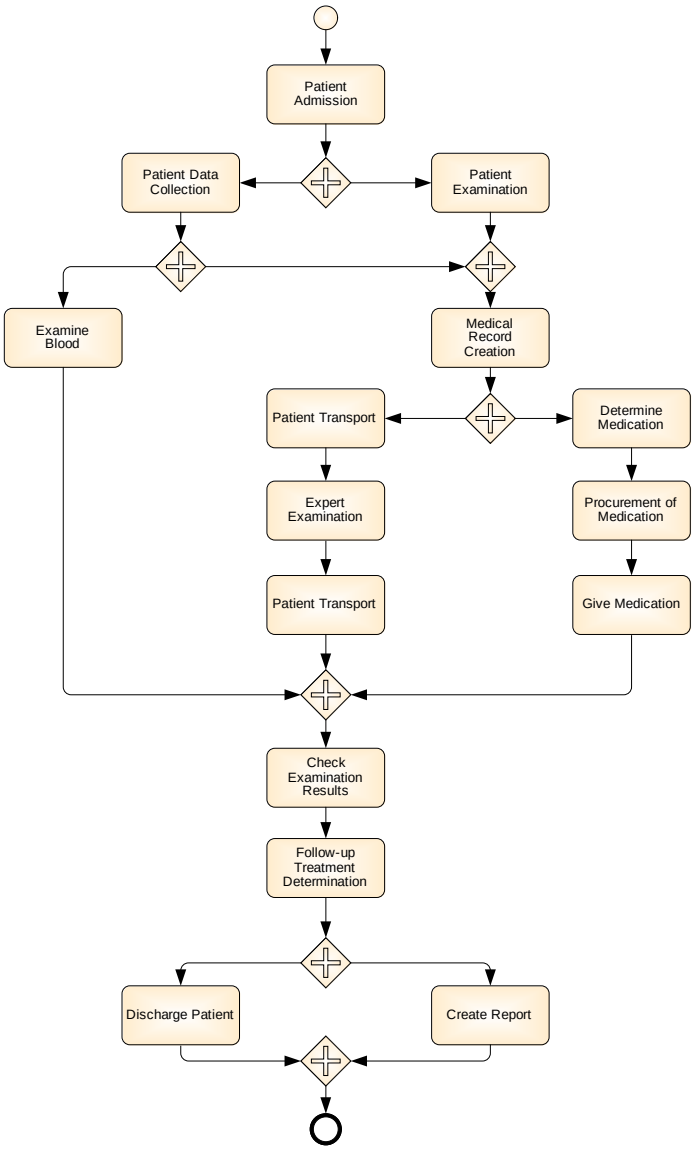

Figure 2.6: Healthcare scenario

The *Vita Health Care Center* offers complex medical services for patients. A variety of treatments and related medical services are realized by different health care providers. They are subcontracted by the *Vita Health Care Center* in order to realize complex treatments for patients.

A patient undergoes a series of stationary checkups organized as a composite service (*Stationary Patient Checkup*) by the Vita Health Care Center. After the admission of the patient to his room (*Patient Admission*), data about the patient is collected (*Patient Data Collection*). This includes information such as weight, height, age, and blood pressure. Furthermore, a blood sample is taken. The medical examination including auscultation and a first talk with a physician takes place (*Patient Examination*). Later the data collected during *Patient Data Collection* and *Patient Examination* is documented in a formal medical record document (*Medical Record Creation*). Based on the medical record a group of physicians determines the first medication for the patient (*Determine Medication*). This medication is ordered from a local medical provider, who delivers the medicine to the healthcare center (*Procurement of Medication*). The blood sample, which was taken during (*Patient Data Collection*) is sent to the laboratory where it is analyzed. The results are returned to the ward where the client is attended to (*Examine Blood*). After the medical record was created, the patient is transported to another location for a special examination (*Patient Transport*). A specialist executes an ordered examination and returns the examination results to the unit which takes care of the patient (*Expert examination*). After the expert examination the patient is transported back (*Patient Transport*) to his ward. After returning from the expert examination and as soon as the ordered medication is delivered, the medication is given to the patient by a nurse (*Give Medication*). Once the results from the blood check and the expert examination are available, a group of physicians checks the results of the different examinations and determines next steps (*Check Examination Results*). Based on the results of this evaluation the status of the patient is determined and a plan for follow-up treatments is created and integrated into the medical record of the patient (*Follow-up Treatment Determination*). Finally, the information captured during the different examinations as well as the plan for follow-up treatments are used as a basis to generate two reports for the patient's health insurance company as well as for his general practitioner (*Create Report*). The patient is released from the medical center and the billing process for the checkup is initiated (*Discharge Patient*).

2.5 Introduction of Service Dependencies

The previous section presented two use cases that apply the IoS vision of tradable services. Logistics and healthcare services are traded via service marketplaces and are composed to form service compositions. These service compositions are offered via a marketplace again.

When services are composed to compositions, they are implicitly collaborating to achieve a business goal. The collaboration of services can be con-

sidered as implicit because typically the single service providers are not aware of the complete business purpose of the service composition and the different services involved. Only the composite service provider (e.g. *OTML* or *Vita Health Care Center*) has information about the overall process. The provisioning of the different atomic services contributes to the overall goal of the composition. Due to the collaboration of services and the fact that atomic services realize the composite service functionality, dependencies exist between these services. In this thesis dependencies, which occur between atomic services, are called horizontal dependencies. Dependencies between atomic services and the composite service are called vertical dependencies. Please refer to Section 4.2.2 for a definition of horizontal and vertical dependencies.

As described before, in the IoS the provisioning of services is regulated by SLAs. It is the responsibility of composite service providers to ensure that the negotiated ASLAs and the CSLA enable the smooth collaboration between the different service providers as well as the successful provisioning of the composite service.

2.5.1 Examples of Dependencies in Service Compositions

In the logistics and healthcare use cases dependencies occur regarding the goods or data being handled by the different services, start and end times, start and end locations, service price, and the quality of service. These aspects are regulated by SLAs. The following examples briefly illustrate different types of dependencies that occur in the two use cases.

- **Resource dependency:** If one of the logistics service providers loses a part of the goods being transported, subsequent service providers will be affected. Furthermore, the goods will not reach the destination, i.e. the composite service is affected as well. Resource dependencies also exist in the healthcare scenario. If the Blood Examination service fails to provide examination results, this information cannot be used during the evaluation of all examination results.

- **Time dependency:** The late delivery of goods by AutoTrans DD affects the start time of the Warehouse Dresden service. Another example of time dependency would be the renegotiation of the start time of the composite service Stationary Patient Checkup by the customer. This would affect the start time of the Patient Admission service.

- **Location dependency:** In the logistics use case a location dependency exists, for example, between the delivery location of the composite service OTML Trans World and the Eastern Truck US service delivery

location. If the customer changes the delivery location of the composite service, the delivery location of Eastern Truck US also needs to be adapted. In the healthcare use case a similar location dependency exists between the Patient Transport end location and the location of the Expert Examination service.

- **Price dependency:** The subcontracted logistics provider Truck DD wants to renegotiate the price of its service. This affects the price of the composite service offered by OTML.

- **Quality of service dependency:** An example of a QoS dependency is the maximum temperature during the transportation process. The composite service maximum temperature depends on the maximum temperature of the different atomic services.

2.5.2 Requirements for a Solution

In this thesis an approach will be presented that supports the management of dependencies between services in service compositions. Dependencies between services are analyzed at design time. They are captured in a dependency model. The dependency model contains explicit dependency information. This information is used during service execution to evaluate the effects of renegotiation requests and SLO violations on other services. Thus, it supports the composite service provider to manage the service composition. The approach realizes the management of dependencies between atomic services as well as between atomic services and the composite service.

As a base for the development of the dependency management approach a number of requirements were identified. They were mostly derived from the described use cases and the dependency examples as well as from the four research questions. The requirements concern the representation of dependencies (dependency requirements - DR), the overall dependency management approach (management approach requirements - MAR), and general requirements (GR).

Requirements Regarding Dependency Representation

A number of requirements were identified with respect to the representation of dependency information in a dependency model.

DR1: Dependencies need to be expressed with a granularity of single service properties.

Dependencies between services occur with respect to different service properties such as QoS parameters or resource, time, and location constraints. While it is sufficient for some application areas of service dependencies to state that two services are dependent on each other (please compare [ZBH08]), this is not the case in the problem domain handled by this thesis. In order to be able to evaluate the effects of violations of service provisioning and renegotiation requests, it is necessary to express the dependencies between services at the level of service properties. This is due to the fact that violations and renegotiation requests occur with respect to single service properties. The violation or renegotiation of a single service property does not necessarily affect the dependent service as a whole but its corresponding service property.

DR2: The representation of dependencies needs to support the expression of multiple dependencies between two services.

The dependencies between services are captured with regard to single service properties (compare DR1). Two services can depend on each other with regard to multiple service properties. Each dependency needs to be captured in the dependency model.

DR3: The representation of dependencies needs to support the expression of multiple dependencies of the same type.

One service can depend on different services with regard to the same service property. Each dependency needs to be captured in the dependency model.

DR4: The representation of dependencies needs to support the expression of type-specific dependency descriptions.

Dependencies may require a more detailed description which helps to automate the handling of related events at runtime. The description of dependencies is type-specific.

DR5: The representation of dependencies needs to support the expression of dependencies between services with 1:1 and 1:n relationships.

A single service may depend on one service (1:1) or multiple different services (1:n) with regard to a specific property.

DR6: The representation of dependencies should support the automatic evaluation of dependencies.

The dependencies between services are captured for the purpose of supporting the determination of effects of SLO violation and SLA renegotiation events. In order to support the automatic evaluation of these events, the dependencies need to be expressed in a suitable way, which supports the automatic evaluation by a software component.

Requirements Regarding the Dependency Management Approach

This section describes requirements which concern the overall approach for managing service dependencies.

MAR1: Dependencies between atomic services need to be managed.

Different services within a service composition collaborate to achieve the functionality of the composite service. Dependencies occur between these services with regard to time and location constraints as well as handled resources.

MAR2: Dependencies between atomic services and the composite service need to be managed. Each service within a service composition contributes to achieving the objective of the composition. Thus, dependencies occur between the atomic services and the composition.

MAR3: Dependencies between services, which are directly or indirectly connected, need to be managed.

Dependencies between atomic services do not only occur between services which are directly connected via a control flow in a process, but between services distributed over the entire process.

MAR4: The approach should support application domain experts in their work.

The goal of managing service dependencies is to support the creators of composite services during the management of their service compositions. The creators of service compositions are experts from a certain application domain (e.g. from the field of logistics). The approach needs to take into consideration that these domain experts may not have a profound background in any field (e.g. computer science) other than their application domain.

General Requirements

A number of general requirements for the approach were identified:

GR1: The work necessary to model service dependencies at design time should be minimized.

At design time, when the service composition is created, the information regarding the dependencies of services within that composition needs to be analyzed. A good solution needs to minimize the work overhead for the user in order to maximize its benefit. If a solution is too complex or too time consuming users will disregard it.

GR2: The completeness of the recognized dependencies needs to be assured.

The analysis of service dependencies must deliver a complete list of the discoverable dependencies. That means, if a certain type of dependency can be discovered, then all of these specific dependencies, which exist in a composite service, should be discovered. If the service dependencies were to be discovered only partially, the system would be more difficult to use.

GR3: The effects of SLO violations as well as SLA renegotiation on other services need to be determined automatically.

The evaluation of whether the occurrence of SLO violations and SLA renegotiation have effects on other services needs to be executed upon the occurrence of such events. A solution should support the user in determining effects on other services more efficiently. Thus, an automatic approach is necessary, which is triggered by the occurrence of the respective events.

GR4: The approach should be flexible enough to be applicable for services from different application domains.

On internet service marketplaces services from different domains are traded. The creation of service compositions independent of the domain as well as the combination of services from different application domains will enable innovative solutions. The management of such composite services is highly relevant for the providers of service compositions.

GR5: The approach should be integrated with the methodology and toolset for developing and trading services in the Internet of Services.

In order to make the vision of tradable services come true a methodology and tools for creating services and service compositions as well as runtime support for service provisioning and monitoring are developed. The service engineering methodology and the tool chain enable the creation and description of services, their deployment to and offering via a service marketplace, the composition of services of different providers to form business processes, and the distributed provisioning of services and their monitoring. The management of dependencies between services is a part of this overall approach. Thus, the proposed approach needs to be integrated with the IoS methodology and infrastructure.

GR6: The approach should be independent of a specific SLA management approach.

Different approaches supporting the management of SLAs exist (e.g. [LDK04, BPSMS07, AF08, LF08]). It should be possible to apply the service dependency management approach to enhance the functionality of different SLA management approaches instead of designing it to be applied with one specific approach.

2.6 Summary

This chapter introduced the vision of the Internet of Services where services are seen as tradable goods offered and purchased via internet service marketplaces. To enable this vision there is a need for service descriptions to express what a service does and under which legal conditions and quality of service it can be provided.

Service level agreements were introduced as a means for regulating service provisioning. They represent a formal contract negotiated between a service provider and a consumer. During service provisioning the monitoring of SLAs is important. It makes violations of SLAs visible to service providers and consumers.

This chapter also introduced two use cases from the logistics and healthcare domains. These composite business services are examples of tradable services. Based on these two use cases the notion of a service dependency was introduced and illustrated by different examples. In the context of the IoS services are composed to form business processes, which fulfill complex tasks. To do so, they implicitly collaborate. As part of this collaboration services may depend on other services, i.e. failure or change of one service leads to failure or change of other services. The active management of such dependencies is a task of the composite service provider. Dependency management is important to ensure the provisioning of the composition.

This thesis describes a management approach for service dependencies in complex business processes. It supports the work of composite service providers. The approach was developed with the goal to fulfill the requirements that were derived from the two use cases. These requirements serve as a basis for evaluating the service dependency management approach in Chapter 5.

3
Related Work

In this chapter various technologies and approaches related to the core concepts of this thesis are discussed. Section 3.1 describes different technologies enabling the formalization of service level agreements. Related to that, different approaches to negotiating and monitoring SLAs are presented in Section 3.2. In Section 3.3 existing work about the analysis, representation, and management of dependencies is discussed. Each approach is discussed with regard to the dependency management concepts presented in this thesis. Finally, the related work discussion is summarized in Section 3.4.

3.1 Formalizing Service Level Agreements

Service level agreements are formal contracts negotiated between a service provider and a service consumer (see Section 2.2). Traditionally, service level agreements were specified using natural language [BLM08, LSE03]. In order to support the automation of SLA handling, a formal representation of service level agreements is needed, which can be evaluated by computers. During the last years a number of technologies for the formal specification of service level agreements were developed.

3.1.1 First Research on SLA Specification

The Web Service Level Agreement (WSLA) language and SLAng were two early approaches to specifying SLAs. The WSLA language was developed

by IBM [LKD+03] as a SLA specification language covering the creation, negotiation, and monitoring of SLAs. A WSLA document contains three main sections:

- **Parties:** Information regarding the signatory parties (i.e. provider and consumer) as well as supporting parties (e.g. third party monitoring service) is listed. The interfaces provided by the different parties are described.
- **Service Definition:** Service level parameters are described for the operations provided by the different parties. This includes information regarding how the respective information is measured or computed at runtime.
- **Obligations:** Service level objectives present guarantees of SLA parameters, which were specified in the service definition section. Action guarantees describe actions that need to be performed, e.g. when an SLO has been violated.

The further development of WSLA was stopped in favor of the WS-Agreement specification for which WSLA served as a basis.

SLAng is a formal XML-based language enabling service level specification with regard to the quality of service attributes. It was developed as part of the TAPAS project [LSE03]. SLAng covers the specification of agreements on different layers ranging from the network and hosting layers to the application layer. In the SLAng approach one or more SLAs are embedded in a SLA contract, which provides information about the (legal) parties, an optional trusted third party and the digital signatures of the parties. Thus, the SLA contract provides a legally binding frame for service level agreements. The SLA contains three sections:

- **End-point description:** Provides a description of the location and facilities of contracting parties.
- **Contractual statements:** Presents information like time of validity of a SLA, pricing and penalty information.
- **Service level specifications:** Description of quality of service attributes and values.

The SLAng language is not being developed any further.

3.1.2 Web Service Offerings Language

The Web Service Offerings Language (WSOL) was developed as an extension of WSDL by Tosic et al. [TPP02]. It allows the creation of different

service offers by a service provider for its customers. A service offering is described as "...a formal representation of one class of service of one Web Service ...". A service class is created by varying different properties (e.g. QoS, price and payment method, consumer rights) of a service.

WSOL allows the description of consumer and third party information, pre- and post conditions of service execution, constraints for different service parameters, and related service offers. However, it does not support negotiating an offer. Instead, a service consumer can select an available offer [TMPE04].

3.1.3 WS-Policy

The Web Services Policy framework is a W3C Recommendation currently available in version 1.5 [VOH+06]. It allows the specification of requirements or capabilities of e.g. Web service endpoints, messages, or operations. A policy may contain a set of such requirements or capabilities. They are concerned with aspects such as security policies or reliable messaging. The explicit description of policies is not part of the specification. Specific service description languages need to be applied.

The goal of WS-Policy is not exactly the same as that of different SLA approaches. A major difference is that SLAs are typically negotiated between two parties. Policies, as described by WS-Policy, are not negotiated, but the specified requirements and capabilities of two parties are compared. If a match is found between the requirements and capabilities of the two parties, the service consumer may utilize the service. However, since the functionality provided by the WS-Policy framework is still similar to that provided by different SLA approaches, it is listed here to provide a better overview of related approaches.

3.1.4 Rule-Based Service Level Agreements

In [Pas05] the Rule-Based Service Level Agreements (RBSLA) language is presented. It is an approach to expressing service level agreements as rules based on formal logics. It provides a formal basis for expressing SLAs, enabling machine readability and SLA monitoring by a rule engine.

RBSLA is based on the RuleML language, which is an XML based rule language driven by the Rule Markup Initiative [Ini09]. The approach extends RuleML with constructs, which are needed for the formal description of service level agreements. An example are event-condition-action rules enabling the automatic reaction to occurring events.

In order to support SLA specific vocabulary (terms such as provider, consumer, metric, ...), an ontology was developed. The respective types can be referenced from within the RBSLA document.

Due to RBSLA's formal and logic base these SLAs can be monitored automatically. RBSLA instances are not directly executable but are translated into an executable form in order to be monitored by rule engines. This is an advantage over other approaches, where it is necessary to write specific code, which extracts SLA information from negotiated SLAs and evaluates measured information based on the extracted SLA information during the monitoring process.

While RBSLA defines a language for expressing service level agreements it does not provide a protocol for SLA negotiation.

3.1.5 WS-Agreement

WS-Agreement is a specification from the Open Grid Forum [ACD+07]. It defines a language and protocol for the offering of capabilities by service providers, the negotiation of agreements between service consumers and providers, and for monitoring the compliance with these agreements. While the WS-Agreement specification provides a structure for SLA documents, it does not specify which aspects of a service are described and how. This needs to be handled by a specific language for service description. This is a major difference compared to WSLA, SLAng, and RBSLA. WS-Agreement enables the use of a domain specific service description language for specifying the single aspects of the service. The WS-Agreement specification defines three different types of documents involved in the negotiation process. In the following sections the different documents and their structure are explained. Following that the negotiation procedure will be outlined, thus also illustrating the purpose and usage of the different documents.

Agreement Structure

There are three different documents that are involved in the negotiation process: the agreement template, the agreement offer, and the agreement. The general structure of these documents is very similar. All of them consist of a context and terms section. The terms section contains the service description terms and guarantee terms subsections. In addition the agreement template contains the agreement creation constraints section. During the negotiation process the agreement creation constraints section is dropped. Information is added to or changed in the other sections. Figure 3.1 shows the structure of the WS-Agreement template.

- **Agreement Context:** Specification of information about the involved parties and their roles (agreement initiator or responder) as well as the time period validity of the agreement.
- **Terms:** Description of what the service will provide and a number of guarantees regarding the service quality specified in the form of service level objectives.
- **Agreement Creation Constraints:** Specification of rules for the creation of a valid agreement offer from the agreement template.

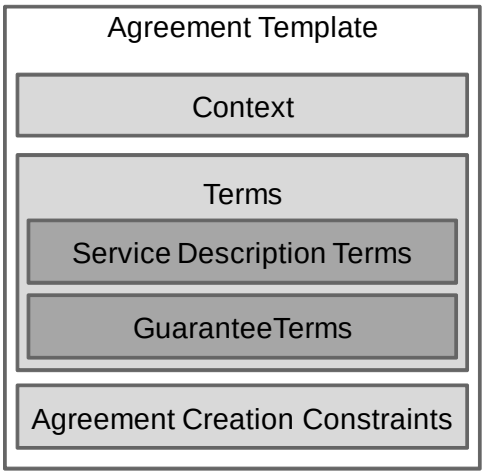

Figure 3.1: SLA template structure

Agreement Negotiation Process

A SLA agreement is created through negotiation between an agreement initiator and a responder. The starting point of the negotiation procedure is a request for an agreement template, which contains information regarding the service level available for a service. There are two options regarding who can offer agreement templates.

1. Agreement templates may be made available by an agreement provider in order to offer services. In that case the agreement provider acts as the service provider. The service consumer (acting as agreement initiator) requests the template in order to find out under which service levels a service is offered.
2. Alternatively, a template may be used by a service consumer to describe which service it would need. In this case the agreement initia-

tor (the service provider) requests from the agreement responder (the service consumer) which services it needs.

It becomes clear that the two roles involved in the process of establishing an agreement, the agreement initiator and the agreement provider, should not be confused with the service provider and the service consumer roles. Both, service provider and consumer can take either role in the course of creating an agreement. Depending on which role the agreement initiator has, the negotiation process proceeds as follows:

- Case 1 - service consumer acts as agreement initiator: The service consumer requests a template from the service provider, which acts as agreement provider in this case, to find out which services it offers. The service provider returns an agreement template. Based on the agreement template the service consumer creates an agreement offer by adapting the template according to its needs and the rules specified in the template. The agreement offer is then sent to the agreement provider which will either accept or reject it. In case of the agreement offer being accepted an agreement is created based on it.

- Case 2 - service provider acts as agreement initiator: The service provider may initiate the negotiation process by requesting from the service consumer what type of service it needs. The service consumer returns an agreement template stating its needs. From this agreement template the service provider will create an agreement offer suggesting what kind of service it could offer and will send it to the consumer. Finally, the service consumer will either agree to the proposal or reject it.

The SLA negotiation procedure between the agreement initiator and responder is illustrated in Figure 3.2.

3.1.6 Discussion

The previous section presented a short overview of approaches to specifying service level agreements and policies for services. WSLA and SLAng are two early approaches to SLA handling, which are not being developed any further. WS-Agreement is being actively developed by the Open Grid Forum. Compared to WSLA and SLAng it has the advantage of flexibility with regard to the language used to describe the service and its properties. It thus enables the application in different service domains. Two alternative approaches, RBSLA and WSOL, offer capabilities to express SLAs or offers,

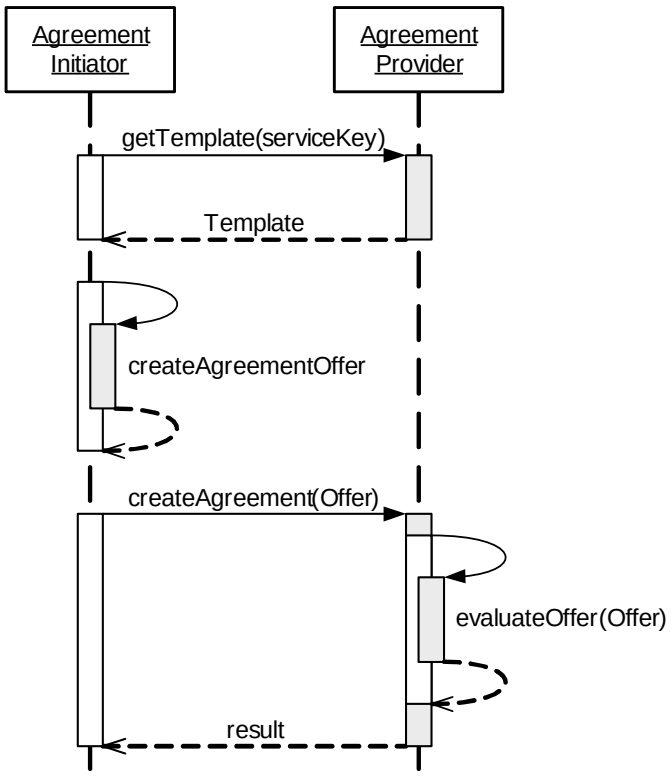

Figure 3.2: WS-Agreement negotiation

but do not provide support for SLA negotiation. Thus, for the work presented in this thesis, WS-Agreement is applied as a language for specifying and negotiating SLAs.

The establishment of SLAs has its focus on the relationship of two or sometimes three parties. All parties are involved in the same single interaction. In complex business processes where the creator of the service composition maintains contracts with the providers of the single services, dependencies do not only occur between the composite service and the atomic services, but also between the atomic services of the composition. Means for describing these relationships are not supported by SLA languages. This is due to the fact that contracts are supposed to regulate the interaction between two parties, not the complex construct of contracts in a service composition. An overview of the different approaches is presented in Table 3.1.

Table 3.1: Approaches to formalizing SLA information

Approach	Description	Negotiation	Dependencies
WSLA [LKD+03]	Formalizing and negotiating SLAs	+	-
SLAng [LSE03]	Formalizing SLAs	-	-
WSOL [TPP02]	Formalizing SLAs	-	-
WS-Policy [VOH+06]	Specification of service capabilities and requirements	-	-
RBSLA [Pas05]	Formalizing SLAs	-	-
WS-Agreement [ACD+07]	Formalizing and negotiation of SLAs	+	-

Legend: Yes (+), No (-)

3.2 Approaches to SLA (Re-)Negotiation and Monitoring

In Section 3.1 an overview of languages for expressing service level agreements was provided. SLAs are negotiated prior to and monitored during the provisioning and consumption of services. The processes of SLA negotiation and monitoring, as well as related management tasks are supported by a variety of SLA management frameworks. In this section different frameworks providing functionality for negotiating, renegotiating, and monitoring service level agreements are described. For each approach a brief discussion is provided regarding its relation to the handling of SLAs and the management of dependencies between services.

3.2.1 WSAG4J

WSAG4J is an implementation of the WS-Agreement specification. It is being developed by Fraunhofer SCAI [Wae08]. WSAG4J provides infrastructure components for agreement providers and consumers supporting the negotiation of SLAs as well as the retrieval of information about negotiated SLAs. The different infrastructure components are freely available for usage in SLA projects. However, the extension of WSAG4J with other functionality is a major effort. One example of an extension would be the support for SLA template deployment from a tool for designing SLA tem-

plates. SLA templates can be created during the design process of a service (compare Section 4.5.1). The deployment of a SLA template from the design tool would allow a better integration with the SLA infrastructure.

3.2.2 Cremona

Cremona [LDK04] is a middleware, which supports the negotiation, monitoring, and management of WS-Agreement-based service level agreements. It was developed by IBM. The provided functionality supports both parties involved in the service provisioning and consumption process, i.e. the service provider and the service consumer. The Cremona system architecture consists of three different layers providing different functionality, as well as interfaces, which can be implemented by custom implementations.

The Agreement Protocol Role Management (APRM) provides functionality for the agreement initiator and the agreement responder focusing on agreement related aspects. On the agreement provider side the Agreement Factory acts as an access point for the creation of new agreements from templates. Agreement templates, as well as negotiated agreements, are held in respective repositories. Information about the negotiated terms and the agreement fulfillment status at runtime is provided. Also available are interfaces for agreement status monitoring, decision making regarding the acceptance or rejection of agreement offers, and information propagation regarding new agreements. On the agreement initiator side, the Agreement Initiator component handles all necessary interaction for the negotiation process. Information is kept regarding agreement factories that can be contacted for requesting agreement templates and negotiating new agreements. Also, a set of all negotiated agreements is available. The Template Processor component supports the creation of agreement documents based on templates. Finally, the Agreement Implementer interface serves for propagating information regarding newly negotiated agreements.

The Agreement Service Role Management (ASRM) provides functionality for the service provider and consumer roles. It is based on the APRM, which provides agreement-related functionality to this service provisioning- and consumption- related layer. The functionality supports the provisioning of services based on agreements, i.e. different components help to monitor or enforce that the service level objectives are met. The process of evaluating whether or not new agreements can be accepted based on the available resources of a system is also supported. For the service consumer the ASRM provides information (e.g. endpoint references) regarding how to access a suitable service based on the current service requirements of the consumer.

The Strategic Agreement Management (SAM) layer supports the process of selecting agreement templates for negotiation, adapting agreement templates according to available resources, and enforcement of negotiated agreements during service provisioning. The functionality of this layer is very domain specific. In many cases it may not be possible to automate the functionality of this layer, but instead human involvement may be required.

The Cremona middleware provides support for SLA handling at different layers including the agreement negotiation handling, SLA based service execution, and strategic SLA selection and adaptation support. However, it does not provide support for managing SLAs in service compositions. No work was done with regard to handling the dependencies between services.

3.2.3 The GRIA SLA Management Service

Within the SIMDAT project the GRIA SLA Management Service was developed [BPSMS07]. It provides functionality for SLA negotiation, the monitoring of agreed SLA, and the enforcement of SLA at runtime. The system supports the handling of SLA at a technical as well as at the business level. Technical SLOs cover aspects like CPU time and disk space consumed by a service consumer while consuming a service. These are the resources needed to provide a service to the consumer. In most cases service consumers are not interested in these facts. They are rather interested in business level aspects, which describe the actual task of the consumer. Business level SLOs cover for example the number of rendered video frames. In order for the GRIA system to support business level SLOs the concept of private constraints was introduced. These are constraints, which are not visible to the consumer and allow the SLA designer to map the business level SLOs to technical SLOs, which can then be monitored by the system. The GRIA system enables the service provider to determine thresholds for the early detection of upcoming problems. Threshold violations can then be handled by predefined management actions, such as the deployment of further services for handling consumer requests or for the controlled violation of SLAs based on priorities of different SLAs. The GRIA SLA management approach does not consider dependencies in service compositions. If a SLO violation occurs, it is handled only with regard to the violating service.

3.2.4 SALMon

In [AF08] the SALMon architecture for monitoring SLAs is presented. It consists of three main components: Monitors, Analyzers, and Decision Makers. Monitors use Measure Instruments to measure different attributes like availability and response time. This information is stored in a data base. The Monitor manages this process. The Analyzer is responsible for determining SLA violations. In case of a violation the Decision Maker is informed. It is responsible for choosing a suitable strategy for reacting to the SLA violation, e.g. by replacing the service which violated the SLA. This approach is limited in so far as that it does not handle effects on other services, but instead tries to apply strategies to fix problems only where they occur, namely the service violating its SLA. This is not sufficient for business services in many cases, as they affect other services as well.

3.2.5 ASG Project

Within the Advanced Services Grid (ASG) project a framework for the negotiation and monitoring of SLAs was created. Service level agreements regulate technical parameters on different software stack levels including the operating system, middleware, application server, or the application itself. Relevant attributes considered for negotiation and monitoring include technical aspects such as response time, availability, and database connectivity monitoring [FTS06]. In [Mom06] an architecture for adaptive service management is presented, which incorporates the components SLA Manager, Service Negotiation, and Service Monitor. They are responsible for the negotiation and monitoring of SLAs. The presented work does not involve the handling of service dependencies.

3.2.6 Renegotiation of Service Level Agreements

Earlier in this chapter the WS-Agreement specification was described (see Section 3.1.5). While WS-Agreement allows for the negotiation of SLAs, it does not provide a mechanism for the renegotiation of SLAs. However, this would be useful in many situations.

In [FMA06] the authors present a formal description of the semantics of different WS-Agreement elements and propose a number of extensions to the WS-Agreement specification with regard to the valid states of agreements (visible to the service consumer), services (e.g. running, finished), and the single guarantees of an agreement, as well as transitions between these states. The goal of the authors was to enable the renegotiation of WS-Agreement-based SLAs. The WS-Agreement specification assumes that a

SLA is violated as soon as a single guarantee is violated. The authors argue that the possibility to renegotiate the single violated guarantee of an agreement would be beneficial, because it would enable the continuation of the service provisioning. The states introduced by this extension enable the specification that a guarantee is close to being violated and thus a reaction is required. In the violated state a guarantee can be renegotiated while this is not possible in the non_recoverable_violated state.

In [MTV09] the authors describe a framework for managing service level agreements in service compositions. The focus is on the renegotiation of SLAs. The renegotiation of SLAs is important to ensure the functioning of service compositions, even in cases where a single service has problems to fulfill its SLA. The authors emphasize that the violation of a single agreement within a service composition may affect the other services in the composition. Thus, it is important that the SLAs in a composition are not necessarily fixed, but instead should enable the dynamic renegotiation upon changes in the context of the composition. Another reason for renegotiation can be that one of the parties of a service composition (provider or consumer) would like to change its contract, for example to receive a better service quality [MTV09].

To enable renegotiation the authors present an extension to the WS-Agreement framework. The proposed extension consists of two parts. On the one hand the structure of an agreement is slightly adopted. A new type of service level objective is introduced, which states that this SLO can be renegotiated. It allows for the specification of time constraints regarding when and how many times renegotiation is allowed. On the other hand the negotiation protocol is extended. Means for creating a renegotiable SLA are added, as well as functions to ask for the modification or the conditional modification of a service level objective. The conditional modification request is necessary for cases where the modification of one contract can depend on multiple other contracts. The presented renegotiation process for simple scenarios (no dependency on other SLA) consists of two messages exchanged between the two involved parties: the request to modify a SLO answered by an accept or a reject message. For more complex cases where multiple SLAs are affected, a conditional modification request is sent to all parties. Only after a conditional acceptance by all parties will the initiator of the renegotiation send a confirmation to all parties.

The work presented in [MTV09] provides a basis for the renegotiation of SLAs in composite services. The authors emphasize the need to consider dependencies between services in the composition. The presented protocol for renegotiation considers this aspect. No work was presented with regard to determining the dependencies between services. This is the task of the dependency management approach presented in this thesis. It could help

to automatically determine which effects the changes of one SLO in one SLA have on other SLAs. The described work does not support that.

3.2.7 Discussion

There are a number of existing approaches supporting the negotiation, renegotiation, and monitoring of SLAs. The different approaches are mainly concerned with single SLAs instead of SLAs in service compositions. An exception is the work by [MTV09], which explicitly considers dependencies between services for the renegotiation of SLAs. However, neither the work presented by Modica et al. nor any of the other approaches explicitly handles dependency information. Table 3.2 presents an overview of the results of this section.

Table 3.2: Approaches to SLA (re-)negotiation and monitoring

Approach	(Re-)Negotiation	Monitoring	Dependencies
WSAG4J [Wae08]	+	-	-
Cremona [LDK04]	+	+	-
GRIA [BPSMS07]	+	+	-
SALMon [AF08]	-	+	-
ASG [FTS06]	+	+	-
Frankova et al. [FMA06]	+	-	-
Modica et al. [MTV09]	+	-	-

Legend: Yes (+), No (-)

3.3 Foundations and Approaches of Dependency Management

In this section the foundations of dependencies are explored from a more general perspective. Different types of dependencies, which have been identified in earlier work, are described. Furthermore, the handling of dependency information for different tasks along the lifecycle of a composite service is outlined. This illustrates the range of tasks for which the handling of dependency information is relevant. Finally, different approaches for representing dependency information, as well as for the management of dependencies, are described.

3.3.1 Relevance of Dependencies

In the area of project management time management is an important aspect. A project consists of different activities. These activities depend on each other for various reasons, e.g. it may be necessary to complete one activity before another one can start. In [PMI08] three different types of dependencies (mandatory, discretionary, external) are distinguished. Mandatory dependencies are based on natural constraints (e.g. one cannot paint a house before it is built) or contractual requirements. Discretionary dependencies are based on specific domain knowledge or experiences. They express preferences that lead to a certain sequencing of activities, while other activity sequences would also be possible. External dependencies are requirements of project activities on activities, which are not part of the project and usually cannot be controlled by the project team. When managing a project it is important to consider these dependencies. The management of times of the different activities is required to ensure the completion of a project in a timely fashion. The management of times involves the definition of activities, the sequencing of activities based on dependencies among them, the estimation of resources and time needed for each activity, and the development and controlling of a project schedule. For the handling of time dependencies the sequencing of activities is of special importance. It is defined as "...the process of identifying and documenting relationships among the project activities" [PMI08]. These relationships can be seen as dependencies between activities. In order to illustrate project activities and the relationships between them, project schedule network diagrams are used.

In their interdisciplinary study on coordination [MC94] Malone and Crowston define coordination as "...managing dependencies between activities". They provide an overview of dependencies and coordination approaches in different scientific fields including e.g. computer science, economics, and organization theory. An important aspect, which they took from the latter discipline, is that in cases where complex activities are divided between different actors, there is a need for "...managing the interdependencies among the different activities ..." [MC94]. Possible ways to achieve this include the usage of a set of rules, the handling by a central actor, and the individual handling by the participants of the complex activity. The approach presented in this thesis assumes the existence of a central actor, namely the composite service provider, which is responsible for the management of dependencies. This decision is based on two aspects. First of all the individual actors of composite services do not have enough information about the overall activity they are involved in or the other actors with whom dependencies exist. Thus, they are not in the position of managing their dependencies with other entities. Secondly, the automatic de-

termination of the right actions for resolving dependencies based on rules is a complex task. It requires the handling of dependency information in order to determine which activities depend on each other. Furthermore, it requires the creation of a rule set for handling different situations. This thesis is concerned with the handling of dependency information. However, it does not cover the creation and handling of rules for resolving problems which occur due to dependencies. The presented approach supports composite service providers to capture dependency information and to utilize this information for the determination of the effects of events at the time of service provisioning. Resulting actions can then be determined either by human beings or by a rule set. The creation of suitable rules for the automatic determination of actions requires a detailed understanding of the application domain. This is not the focus of this thesis.

In the area of service-oriented architectures dependencies occur between services in service compositions. Information about dependencies is used to achieve a variety of tasks throughout the lifecycle of a service.

- **Creation of processes:** In [Tol03] an approach to model service dependencies is presented. The goal is to support a management system to automatically create service flows based on dependency descriptions. Other approaches to automatically create service compositions are presented in [OS08] and [ZPPN07].
- **Optimizing sequencing constraints:** In [ZBH08] and [WPSB07] dependencies between atomic services are used to optimize sequencing constraints between diffent services. The authors argue that business processes are often over-specified with regard to sequencing constraints. A proper understanding of the real dependencies (e.g. data or control dependencies) enables an improved handling of services during the analysis, modeling, and execution of business processes.
- **Root cause analysis:** In [BWRJ08] and [EK02] dependencies between atomic and composite services are used to support root cause analysis in the case of service failure. Dependencies between SLAs are monitored with the goal to support the creator of a composite application to analyze problems of the composite service.
- **Management of service level agreements in service compositions:** Ludwig and Franczyk provided a solution for the management of SLAs in service compositions. They handle dependencies between atomic and composite services [LF08], which occur regarding different quality of service attributes as well as pricing information. The approach supports the composite service provider in validating the different SLAs to assure a proper provisioning of the composite service.

- **SLO Violation Effects:** Dependencies between atomic services in service compositions may lead to situations where problems during the provisioning of one service influence or even render impossible the provisioning of other services. In many cases those dependencies cannot be avoided. Instead it is necessary to actively manage them. The approach described in [BCD08] deals with effects of service failure on other services. Also, the approach described in this thesis supports the determination of SLO violation effects.
- **SLA Renegotiation:** There are situations where a stakeholder of a service composition (atomic service provider, composite service provider, service consumer) needs to renegotiate an existing SLA. This is the case when some replanning takes place at the respective stakeholders side. Here it is important to identify the potential effects of renegotiation on other SLAs within the service composition before renegotiation is executed. The evaluation of renegotiation requests is supported by the approach presented in this thesis. To the best knowledge of the author no other approach handles this issue.

3.3.2 Types of Dependencies

There are different types of dependencies that are distinguished by different researchers. Malone and Crowston [MC94] describe different kinds of dependencies and respective coordination approaches, which are applied in different scientific fields (e.g. economics, social sciences, computer science) to manage these dependencies.

- **Shared resources:** This type of dependency occurs when different activities need to use the same limited resources such as processor time or storage space. The management of resource dependencies can be realized by mechanisms for allocating the limited resources to the different activities.
- **Producer / consumer relationship:** When an activity A1 has a need for physical goods or information, which are the outcome of a different activity A2, the consumer A1 is dependent on producer A2. Producer / consumer relationships can be coordinated e.g. by notification strategies, where the producer informs the consumer that it finished its task.
- **Simultaneity constraints:** Two activities can be dependent on each other with regard to the time when they take place. They can be required to happen at the same or at a different time. The coordination of simultaneity constraints can be realized e.g. by a suitable scheduling mechanism.

- **Task / subtask dependency:** This type of dependency occurs when the achievement of a complex activity is realized by the execution of different activities. According to Malone and Crowston, suitable coordination mechanisms for handling task / subtask dependencies are task decomposition or goal selection mechanisms.

Wu et al. categorize different dependency types that occur in business processes [WPSB07]:

- **Data dependencies:** Data dependencies occur when a consumer of data is dependent on the provisioning of data from a producer.

- **Control dependencies:** These dependencies occur when the execution of a part of a process is dependent on a flag (i.e. the value of a variable).

- **Service dependencies:** The interaction between a process and a remote service defines service dependencies. They may, for example, require the scheduling of a listener for an asynchronous service call or the sequential calling of different ports as part of the service interaction.

- **Cooperation dependencies:** In order to achieve a certain goal, different activities (A1 and A2) need to cooperate. In case there are constraints regarding their synchronization (e.g. A1 before A2), which are not covered through other dependencies, there is a cooperation constraint.

While the work of Malone and Crowston is rather general and concerns dependencies which occur in different scientific disciplines, the work by Wu et al. is specific to business processes. Nevertheless, there are certain types of dependencies which are very similar. Data dependencies form a subset of producer / consumer relationships. Also, cooperation dependencies cover similar aspects as simultaneity constraints.

The different types of dependencies, which are handled by the dependency management approach presented in this thesis (see Section 4.2), overlap partly with the dependencies discussed by Malone and Crowston. Resource dependencies are equivalent to the producer / consumer relationship. Dependencies regarding start and end times of services correspond to simultaneity constraints. Dependencies regarding the location, price, and quality of goods transport are not covered by the work of Wu et al. or Malone and Crowston.

3.3.3 Approaches to Handling Dependencies

Various approaches to handling service dependencies exist in scientific literature. A brief overview regarding the different purposes for handling dependency information was presented in Section 3.3.1. In the following sections different approaches, which are more relevant for this thesis, are presented and discussed with regard to the core contributions of this thesis. The different approaches vary regarding the dependencies they handle (i.e. horizontal or vertical) as well as the purpose of handling dependency information. A further difference is the approach for capturing dependency information, e.g. via modeling or discovery of dependency information.

HP Discovery and Dependency Mapping

The Discovery and Dependency Mapping (DDM) is a software tool developed by Hewlett-Packard [HP08]. It supports change management in IT infrastructures. Together with the HP Universal Configuration Management Database (UCMDB), it is a part of the Predictive Change Impact Analysis solution.

It enables the discovery of components in an IT infrastructure as well as dependencies between the different components. Dependencies occur between components on the same or on different levels of the network stack. Examples of inter-dependent components include business applications, J2EE and .NET components, databases, storage devices, and network ports.

Dependencies are discovered by running different types of queries against standard interfaces of the system to be analyzed [HP07]. This way, dependencies such as “infrastructure monitor relationship” or “DB Client” can be discovered. In collaboration with the UCMDB, impact analysis functionality for planned changes to the infrastructure is provided.

The DDM and UCMDB functionality differs from the work presented in this thesis regarding several aspects. DDM and UCMDB target at providing impact analysis functionality to manage changes in the infrastructure, whereas the dependency management approach aims at supporting composite SLA management. Thus, the basic goals of the two approaches are fundamentally different. Furthermore, there is a difference in the dependency information that is handled. DDM and UCMDB handle component dependencies such as “infrastructure monitor relationship” or “DB Client”. The dependency management approach introduced in this thesis handles detailed dependency information regarding services and their SLAs by describing time, location, resource, pricing and QoS related dependencies. Finally, the dependency discovery approach of DDM differs from the dependency management approach. Instead of exploring dependencies in

an IT infrastructure by sending queries to its components, the dependency management approach analyzes composite service workflow descriptions and SLA information.

Dependency Analysis Based on Message Logs

In [BCD08] the authors present an approach for the discovery of service dependencies for the management of a service-oriented architecture. They argue that knowledge about service dependencies is fundamental to enable the determination of effects of service failure on other services in a composition as well as for the analysis of the root-cause of problems. The work is presented in the context of the HP SOA Manager[1], which enables the handling of dependencies but requires the manual specification of dependencies.

The suggested approach enables the analysis of logged messages between services. A number of assumptions have been made regarding the messages being analyzed. One assumption is that there is no information inside the messages available which enables their correlation. Also, the analysis of information in the message body is disregarded due to the high workload for logging this information. Furthermore, for a message only limited information regarding its origin and destination is available (e.g. IP address info), but no detailed information regarding the services. Finally, it should be possible to consider dependencies with services, which are not running in a managed container, but are rather legacy systems wrapped by a Web service interface. Thus, a modification of the messages is not possible.

The discovery of dependencies consists of the following four steps:

- **Analysis of dependencies between messages:** For the analysis of causal dependencies between messages the authors present three different approaches. The first approach considers the probability that a message was sent based on the fact that another message was sent before, taking into consideration a certain time window. The second approach assumes a statistical distribution for the duration of service executions and analyzes in how far the distribution of time difference between two messages fits the service execution time distribution. The last approach computes a histogram for all message pairs with respect to the time difference between the two messages. The assumption is that if few time difference intervals occur with a high probability then these two messages are likely to be dependent.
- **Dependency graph construction:** A dependency graph is constructed from the analyzed data. Services are represented as nodes.

[1] http://h20229.www2.hp.com/products/soa/

Edges between the nodes represent the dependency. These dependencies are annotated with probabilities for the level of confidence. This is necessary due to the fact that dependencies are not calculated with absolute certainty.

- **Dependency graph reduction:** As a next step the dependency graph is reduced by applying a threshold to the confidence levels. All edges with a lower confidence level are removed.

- **Frequent path determination:** Based on the dependency graph and the original log information paths of dependencies are created and evaluated regarding the frequency of their occurrence. The determined paths represent patterns of identified Web service conversations, which can then be used to e.g. identify root-causes of problems.

The approach for analyzing dependencies based on message exchange information is very different from the approach presented in this thesis. It assumes the execution of services as the basis for the analysis. This is, however, not feasible for the dependency management approach presented in this thesis, since all dependencies should be captured prior to service execution. Also, a dependency graph should be able to distinguish dependencies based on different SLOs in order to be able to determine the effects of different SLO violations as well as renegotiation requests for certain SLOs. That means that the representation of dependencies of the approach is not powerful enough to fulfill all requirements discussed in Section 2.5. Finally, the overall approach presented by [BCD08] is not concerned with the negotiation and renegotiation of SLAs.

Managing Service Dependencies in Distributed Systems

Ensel and Keller [EK02] introduce an approach to manage dependencies between services (e.g. web application server, database, operating system) in distributed systems. The goal is to support root cause as well as impact analysis for service failure situations. Dependencies are represented in a RDF based dependency model, which captures services and dependencies along with their attributes. An architecture of a dependency management system for querying dependency information from distributed managed resources is also presented.

The approach to capture dependency information in a dependency model, as well as the specific design of the model, is similar to the dependency management approach presented in this thesis. But while the basic structure of the dependency model is similar to the metamodel presented in this thesis (i.e. explicit representation of services, dependencies, antecedents,

dependents), there are differences in the relevant attributes of the different entities being modeled. This is due to the different domains of use (managing services in distributed systems vs. composite service SLA management). Also, the formalization of the dependency model is different. Ensel and Keller use RDF (compare [BM04]) for representing the dependency model instead of a XML Schema-based metamodel. Furthermore, there are architectural differences. In the approach by Ensel and Keller the dependency model is not stored in a central place, but is instead created from dependency information available at the different managed resources, i.e. each resource keeps information about its dependencies. This is different from the approach presented in this thesis, where dependency information is managed centrally by the composite service provider and the individual services do not have detailed information about the context in which they are provided. The reason for this is the assumption that a composite service provider has no information about the inner functioning of services he consumes. A further difference is that Ensel and Keller make no assumptions with respect to the creation of the dependency model. The authors mention a number of different existing approaches including explicit modeling of dependencies or the discovery via monitoring the activity of service pairs over time. This is, however, very different from the dependency discovery approach presented as part of this thesis.

Scheduling Business Process Activities

Wu et al. present an approach for modeling and optimizing the synchronization dependencies of activities in business processes [WPSB07]. A synchronization model, which contains dependency information, is used to support activity scheduling in business processes. The authors emphasize that scheduling activities based on dependency information has the potential to parallelize the execution of activities and thus improve business process performance.

Dependency information, which is needed to create a dependency model, is assumed to be available from design time constructs such as UML activity diagrams or other models as well as from domain experts.

The authors introduce four types of dependencies: data, control, service, and cooperation dependencies. The different types of dependencies can be explicitly modeled based on the DAG Synchronization Constraint Language (DSCL). DSCL is a modeling language for the synchronization of activities. Dependency information for each type of dependency is captured in a separate model. The information about the different dependencies is then merged into a single model and reduced to create a minimal set of dependencies for a business process. When the different types of depen-

dencies are captured, some of these dependencies may cause the same constraints on synchronization. The reduction process removes such dependencies. The outcome is a minimal set of necessary dependencies which results in a better handling, i.e. less resources are needed for the evaluation of dependencies as well as for the management of the dependency set. The DSCL representation of the minimal set of dependencies can be transformed into a Petri Net representation for the validation of the dependencies. The validated model is then used to create a BPEL process description.

In the presented approach, the authors are concerned with dependencies between the single services of a business process. Dependencies between the atomic services and the composite service (i.e. the business process) are not considered, though. Also, in the approach by Wu et al., dependencies are explicitly modeled. The dependency management approach enables the semi-automatic creation of a dependency model. Several types of dependencies are discovered automatically. Another important aspect, which distinguishes the work of Wu et al. from the work presented in this thesis, is the reduction of the set of dependencies, i.e. removing dependencies leading to the same synchronization constraints. It is an important aspect when applying dependencies for activity scheduling, which helps to optimize their handling. On the other hand, this hides the original source of the dependency. While this information is not important for activity scheduling, it is essential for the management of service compositions with regard to service monitoring and SLA negotiation as well as renegotiation. Furthermore, the representation of dependencies via synchronization constraints is not applicable for the dependency management approach. This model can not hold all the information needed to manage composite services.

Optimization of Sequencing Constraints

In [ZBH08] the authors discuss control and data dependencies in business processes and argue that they form the basis for sequencing constraints in business processes. However, business processes, e.g. in the form of a BPEL process, are often overspecified with regard to sequencing constraints. Capturing the different dependencies in a suitable model and thus having a clear understanding of the real dependencies would help to handle these dependencies during modeling, analysis, and execution of business processes.

The authors present an approach for deriving control dependencies from semantically annotated business activities by evaluating their preconditions and effects. Data dependencies are separated into mandatory

and optional dependencies. Input and output parameters of business activities comprise mandatory dependencies while data used for the evaluation of conditions on business activities forms the basis for optional data dependencies. Control and data dependencies are recorded in separate dependency graphs. Direct dependencies occur when one business activity directly depends on another. Indirect dependencies occur when a number of direct dependencies lead from one business activity to another. As a next step the different graphs are merged and optimized by removing double dependencies (control and data dependency occurring between two business activities) as well as direct dependencies which can be expressed by indirect ones. The authors argue that control and data dependencies can be used to support process modeling, as well as for identifying that a process is overspecified (i.e. unnecessary sequencing constraints), and to help to relax those constraints at runtime.

The approach is related to the dependency management approach with regard to two aspects. First of all the approach captures the dependencies between atomic services in a service composition. Secondly, the dependency model creation occurs as an automated process at the design time of the composition. However, Zhou et al. have a completely different goal. They apply dependency information to relax sequencing constraints between services at runtime. The goal of the dependency management approach presented in this thesis supports SLA negotiation and the evaluation of SLO violations and renegotiation requests with the goal to support the management of the composition. The different goals bring along different requirements for the dependency model as well as for the approach of creating dependency model instances.

In order to capture the dependencies between services with the goal of supporting business process modeling, analysis, and execution with regard to relaxing sequencing constraints, it is necessary to capture dependency information on a service level. More detailed dependency information, e.g. with regard to which data or precondition one service depends on another one, is not necessary. However, in order to support e.g. SLA negotiation, detailed information is necessary regarding the aspect by which two services depend on each other. Only with this information is it possible to determine the effects of specialized events, such as violations or renegotiation requests of specific service attributes. Thus it is clear, that the dependency model requires more detailed information.

The requirement for more detailed dependency information does not only result in a more fine-grained dependency model, but also affects the dependency model creation process. Zhou et al. propose a minimal dependency model, which results from merging the independent control and data dependency models. If one service depends on another one with regard to

control and data dependencies, only the general information about an existing dependency is kept, while the detailed information for the sources of these dependencies is disregarded. Applying this merging process to the dependency management approach of this thesis would result in the loss of important dependency information. If a data dependency is removed, effects of violations or renegotiation requests with regard to this data dependency cannot be determined any more. It is thus necessary to keep all dependencies along with information about the dependency type for better evaluation at runtime.

Furthermore, the two approaches differ concerning the information regarding which dependencies exist, regarding the information which is represented in the dependency model, as well as the artifacts used for determining the dependencies. While Zhou et al. analyze control and data dependencies this thesis is concerned with resources handled by a service, time and location constraints, as well as the quality of service and pricing information. In order to analyze the dependencies between services a semi-automatic process was developed. It captures dependency information based on SLAs and the BPMN process description of a service composition. Zhou et al. base their work on semantic service descriptions and a BPEL process description.

Dependency Markup Language

The Dependency Markup Language (DML) is a language for specifying dependencies between Web services [Tol03]. It was developed to provide an alternative way of describing composite services. While a Web service interface provides a fairly coarse grained description of a service, a BPEL description specifying the control flow of a Web service is a very fine grained description. DML was developed to provide a solution with a granularity between these two approaches. It models different types of dependencies between services. The model of dependencies can then be used to support tasks such as service coordination, discovery, and classification.

The DML was specified as a metamodel in the form of an XML Schema. It has two main building blocks: the description of different dependency types and the specification of a process. The dependency type description enables the modeling of different types of dependencies, which can be used to describe the dependency between two services within a process, e.g. temporal constraints such as *strictSequence*, task-subtask relationships, and producer-consumer dependencies. A dependency type can be derived from another one, which enables the specialization of dependencies. A process allows the specification of different dependencies between the elements it consists of. Similar to dependencies, processes can specialize other

more abstract processes. Thus, hierarchies of processes with different levels of abstraction can be created.

A DML model can be used for the coordination of services by a management system. Its task is to create a suitable service flow based on the modeled dependencies. To enhance service discovery, the author suggests using the dependency model of a service to provide the user with information about the internal functioning of the underlying process and the effects of that on the user, e.g. which steps are to be executed at the beginning or later in the process. This information may have an influence on the decision of a user regarding whether or not to consume a service. Finally, a classification of processes can be created. This is supported by the fact that processes can be created as a specialization of other processes. This classification may be useful information for the consumers of services when searching for a certain kind of service.

The DML was developed to enable the modeling of dependencies between the different services within a composition. This is different from the approach presented in this thesis, where dependencies between services are captured in a semi-automatic approach. DML allows the capturing of different types of temporal dependencies, resource related dependencies, and dependencies concerning the abstraction and refinement of processes. The model is open in so far as that it allows the specification of new types of dependencies besides the supported ones. However, the description of the different dependency types is limited. Detailed specifications of dependency description information are not modeled. However, this is a requirement for the dependency management approach presented in this thesis. Finally, DML was developed with the goal of supporting service discovery and coordination. The goal of this thesis is, however, to manage dependencies in service compositions with respect to the negotiation, renegotiation, and monitoring of SLAs. DML served as a basis for the development of the metamodel, which is used in this thesis to model dependency information. However, modeling constructs for the creation of processes was left out while more detailed descriptions for dependencies were introduced.

Aggregation of QoS in Service Compositions

Cardoso et al. [CMSA04] present an approach for the calculation of QoS properties of workflows from task QoS properties. Using the Stochastic Workflow Reduction (SWR) algorithm, workflows can be reduced in a stepwise process leaving only one remaining task with the QoS of the workflow at the end of the process. Six different reduction rules are applied by the SWR algorithm. They are called *sequential*, *parallel*, *conditional*, *fault-tolerant*, *loop*, and *network*. The authors present formulas to calculate aggregated

cost, time, and reliability of tasks for each of these reduction rules. As a prerequisite the workflow under consideration needs to contain tasks that have QoS estimates assigned to them. Furthermore, workflow transitions between the different tasks have transition probabilities assigned to them. Atomic QoS values, as well as transition properties, are used by the QoS aggregation formulas during the workflow reduction process.

Jaeger et al. [JRGM04] present an approach to aggregating QoS properties (execution time, cost, encryption, throughput, uptime probability) of atomic services in service compositions, which is very similar to the work of Cardoso et al. [CMSA04]. The approach was developed to serve two major purposes. On the one hand the selection of services during the process of creating a service composition at design time requires the consideration of the QoS values of the selected atomic services with respect to the QoS requirements of the composite service. At runtime, on the other hand, a service composition may be modified by adding or replacing services. Such modifications also require the consideration of composite service QoS requirements. Based on workflow patterns described by [VDATHKB03], the authors define seven different composition patterns, including e.g. *AND-AND*, *OR-OR*, *Sequence*, which are used to represent a composite service. For each composition pattern a number of formulas are defined for aggregating the different QoS properties. The composition patterns and associated formulas are used to calculate the composite QoS properties of a composite service in a stepwise process. Services, which are connected via a certain composition pattern, are aggregated to a single node. The different QoS properties of the services are aggregated to a composite QoS value using the specific formulas associated with the composition pattern and the respective QoS properties. During this process the graph of the composite service is reduced to a single node, which has the composite QoS properties assigned to it. The approach is illustrated in Figure 3.3. In a first step two services connected via an *AND-AND* composition pattern are aggregated followed by the aggregation of a *Sequence* pattern.

In [CP04] the authors present an approach to creating service compositions under QoS constraints. The selection of atomic services occurs based on QoS constraints, which enable fulfilling composite service QoS requirements. The approach for the calculation of composite service QoS properties is based on the work of Cardoso [CMSA04], with minor modifications (e.g. replacing probability value for loop statements with concrete number of loop executions). In order to find a valid selection of atomic services implementing the composite service, the authors apply a Genetic Algorithm approach. For each abstract service in a workflow a possible candidate is selected from a set of available ones. The different composite QoS properties (e.g. cost, response time) are calculated. A fitness function is used to evaluate the overall quality of the created composition in the light of the

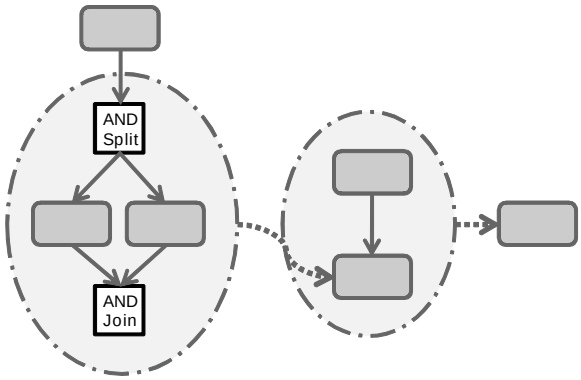

Figure 3.3: Aggregation of QoS (based on Jaeger et al.)

different QoS properties. Using the Genetic Algorithm different solutions are found and compared with each other based on the fitness function.

The approaches developed by Cardoso et al. [CMSA04], Jaeger et al. [JRGM04], and Canfora et al. [CP04] enable the calculation of different composite QoS properties. All approaches cover QoS relationships between the composite service and its atomic services. These QoS relationships (dependencies) play an important role for the management of dependencies between services in service compositions. Thus, these works form the basis for a part of the approach presented in this thesis. Dependencies between atomic services within a composition are not handled by these approaches.

MoDe4SLA

The MoDe4SLA (Monitoring Dependencies for SLAs) approach presented in [BWRJ08] was developed with the goal of supporting the creators of a composite services to analyze problems of the composite service. The approach does not only support the monitoring of composite service SLAs, which were established between the composite service provider and the service consumer, but enables the analysis of why violations occurred with regard to the atomic services forming the composition. In order to enable this analysis, Bodenstaff et al. suggest the analysis of SLAs at design time and the monitoring of dependencies based on event logs after the service execution.

The MoDe4SLA [BWRJ08] approach supports the handling of dependencies between a composite service and its atomic services. More precisely, dependencies of the type response time and price are supported. For each type of dependency and, thus, for the single service level objectives a sepa-

rate dependency model is created. These models express on which atomic services a composite service depends. They do not qualify how it depends on them. The modeling of dependencies is realized by two domain specific dependency modeling languages, which have only slight variations. Both languages allow the modeling of constructs, such as *service S1 depends on service S2*. The usage of the constructs AND as well as XOR enables the modeling of a composite service depending on two or more services, or on one service out of a set of services respectively.

In order to capture information regarding how a composite service depends on different atomic services, the dependency models are refined to an impact model. In order to achieve that, the different dependency modeling constructs of the dependency models are analyzed. An equation for calculating the impact of each node connected to the modeling construct is inserted into the impact model.

To enable the evaluation of dependencies, monitoring information measured for the composite service, as well as for all atomic services involved in the provisioning of the composite service, is applied. Based on the collected monitoring information, the dependency models, and the impact models, the approach allows the evaluation of how much each service contributed to the success or failure of the composition.

The authors target the analysis of composite service execution and management of compositions. This enables composite service providers to remove unreliable services from a composition or to offer better QoS values in SLAs. This is a different goal than the management of service compositions with a focus on SLA negotiation, renegotiation, and monitoring.

The overall approach of MoDe4SLA, which allows dependency modeling at design time and evaluation at runtime, is similar to the general approach presented in this thesis. Nevertheless, there are many differences. In the dependency management approach the dependency model is only partially modeled, as other parts are generated automatically based on SLAs and the process model of the composite service. Also, the evaluation of the dependency model at runtime is done at the moment when SLO violations or SLA renegotiation requests occur. In the MoDe4SLA approach the evaluation is done after a period of time regarding which SLOs were agreed upon (e.g. average response time=X within 1 month).

The types of dependencies supported by MoDe4SLA are also different. It does not support dependencies between atomic services (i.e. time, resource, location dependencies).

Adaptive SLA Management in Value Chains

In [KMK09] an architecture for the management of service level agreements is presented. The focus is on SLA-based resource management in hierarchies of agreements for services with individually negotiated SLAs. The authors present an agent-based architecture consisting of resource agents (resource providers), dispatcher agents (service providers), and customer agents (customers). Resource agents encapsulate resources, e.g. a bus for transporting people, which they offer to dispatcher agents. Dispatcher agents offer services to customers. These services are realized by resources. Dispatcher agents manage the available resource offers from resource agents and service requests from customer agents and match them. Customer agents request services from dispatcher agents.

Dispatcher agents cannot only offer resources which they manage themselves, but may subcontract other service providers to fulfill their contracts with customers. To achieve that, they ask other service providers for an offer, and evaluate the offers in order to find the most suitable offer according to their needs.

SLAs are created in order to regulate the relationship between service providers on the one side and customers (service provisioning), resource providers (usage of resources), and service providers (subcontracting) on the other side. A contracting component of the service provider handles the creation of SLAs from received offers upon the selection of resources and subcontracted services. Also the monitoring and billing of the contracted services is handled by the dispatcher agent.

The communication between the different agents is realized by means of web service standards. This enables the integration with SOA-based systems. The usage of agent technology allows the application of an agent-based coordination mechanism for the negotiation of SLAs.

Dependencies between SLAs on different levels of value chains, i.e. vertical dependencies, are handled. Dispatcher agents can react to changes in customer requirements (e.g. the need for more transportation resources) as well as to changes in resource provider's and subcontracted service providers capabilities (e.g. resource broken). The approach does not handle dependencies between the different resource providers and subcontracted services.

The authors do not present any work regarding the discovery of dependencies. This is different to the dependency management approach presented in this thesis. Also, Karänke et al. put the focus on different dependency aspects. They support the handling of the underlying resources needed for service execution (e.g. a bus to realize a transportation service) rather than the handled resources of services (input/output).

Composite SLA Management (COSMA)

One challenge which providers of composite services are facing is the management of SLAs negotiated between providers of atomic services and the consumers of the composite service. This problem is described by Ludwig and Franczyk [LF08]. The authors describe the COSMA (Composite SLA Management) approach, which provides support for automatic management of SLAs in service compositions.

The COSMA approach supports SLA management during the SLA lifecycle [Lud09]. It covers the dependencies of QoS attributes as well as pricing information between atomic and composite services within a service composition. The COSMA approach consists of three building blocks, namely COSMAdoc, COSMAframe, and COSMAlife.

COSMAdoc is a document which is bound to one composite service and used by different framework components during the SLA lifecycle. It contains all atomic and composite SLAs of the service as well as dependency information of different SLOs of these SLAs. The COSMAdoc contains the following sections:

- Header: expresses generic meta information such as the owner of the document instance, version information as well as information regarding languages and semantic models used within the document.
- ServiceComposition: presents the orchestration script of all services within the composition which "...governs the mapping and dependencies between SLA parameters of composite and atomic SLAs ...".
- SlaSetAssembly: contains all SLAs of the composite service
- SlaSetUsageValidation: contains constraints and requirements information regarding all SLA document elements (not regarding the specific data instances).
- SlaSetDataValidation: allows the expression of constraints regarding the data contained in the SLA (e.g. fixed value, boundaries for data values). Constraints may be expressed by formulas for the aggregation of SLA parameter values from atomic SLAs to composite SLAs. These formulas are expressed in a separate section and are referenced from the SlaSetDataValidation section.
- AggregationFormulas: formulas expressing the calculation of composite SLA parameters from atomic ones.

COSMAframe is a conceptual framework for managing the SLA lifecycle of composite services. It describes different components, which are involved in the handling of the SLA lifecycle:

- COSMA Manager: triggers different lifecycle steps of the COSMAdoc and provides validation functionality.
- COSMAdoc Creator: creates COSMAdoc instances based on a given composition.
- COSMAdoc Integrator: integrates relevant SLA contents (parameters) into COSMAdoc instances as well as restrictions based on contained parameters.
- COSMAdoc Repository: provides the means for storing COSMAdoc instances.
- COSMAdoc Validator and Violation Detector: provides validation functionality for COSMAdocs and handles the evaluation of SLO measurements. Violations of composite SLAs are determined based on SLO measurements and aggregation formulas. Also suggests actions when SLA violations are detected.

COSMAlife describes different practices for the management of SLAs based on COSMAdoc during its lifecycle. Three different lifecycle phases of the COSMAdoc are described. During the creation and integration phase a COSMAdoc instance is created from a template. Each COSMAdoc instance is specific to its related service composition. The COSMAdoc instance is then enriched with pre-set SLA data, information regarding validation and constraints, and service composition information. During the negotiation phase the concrete atomic services, which are to be used to implement the composition, are determined. The validity of the COSMAdoc instance is validated during the negotiation process to ensure that the negotiated parameters fulfill the constraints described in the COSMAdoc. Finally, during the validation phase, the COSMAdoc instance is validated based on monitoring data, which was collected during the process execution. In case of detecting violations actions are determined for reacting to the violations.

The COSMA approach allows the handling of vertical dependencies between services. Dependencies occur with respect to QoS parameters of atomic services and a service composition as well as price information. The dependencies between these services are expressed via functions, which form the basis for calculating QoS and price values of a composite service from its atomic building blocks. COSMA does not support the handling of horizontal dependencies between the atomic services of a service composition. In business service compositions horizontal dependencies play an important role. They occur with regard to handled resources as well as time and location constraints. Their handling is an important aspect, which is one of the key contributions of this thesis.

A further difference is the representation of dependency information. The COSMAdoc captures all information including negotiated SLAs, composition script (i.e. workflow description), and dependency information in one document. In contrast to that the dependency management approach presented in this thesis captures SLAs, composite service workflow description, and dependency information separately. This results in a better separation of concerns. It enables easier changes to the different parts of the composite SLA management and dependency handling, e.g. using a different format for expressing SLAs or extending the dependency model.

The COSMA approach is closely related to the work presented in this thesis. In fact, the automatic creation of calculation formulas was integrated into the dependency management approach as part of the solution.

3.4 Summary and Discussion

In this chapter different approaches to formalizing service level agreements were presented. None of the presented approaches offers functionality for capturing information about service dependencies in service compositions. Also most existing approaches to negotiating, renegotiating, and monitoring SLAs do not consider dependencies between services. Some work exists (e.g. [LF08, KMK09, BWRJ08]) which deals with dependencies between a composite service and its atomic services in the light of managing SLAs. However, none of these approaches handle dependencies between different atomic services in a service composition.

Several existing approaches support the discovery of dependencies. The calculation of composite QoS values ([CMSA04, JRGM04, CP04, LF08]) also serves as one building block of the dependency management approach. The work by Zhou et al. [ZBH08] has some similarities to the discovery of horizontal dependencies as presented in this thesis. However, the approach uses different artifacts for the discovery process. Furthermore, the discovery process creates a reduced dependency model with only limited dependency information. This is not sufficient to support the negotiation, renegotiation, and monitoring of SLAs. Finally, the discovery process presented by Basu et al. [BCD08] is based on service runtime message information. This is not feasible for the dependency management approach, since all dependency information is needed at design time.

Table 3.3 gives an overview of the different approaches in relation to the work presented in this thesis. It provides information regarding the coverage of horizontal and vertical dependencies, the support for discovering dependencies, and the types of dependencies which are supported.

Table 3.3: Approaches for handling dependency information

Author	Horizontal	Vertical	Discovery	Dependencies
SLA management				
This thesis	+	+	+	QoS, price, resource, time, location
Ludwig [Lud09]	-	+	+	QoS, price
Karänke et al. [KMK09]	-	+	-	Resources for service execution
QoS aggregation				
Cardoso et al. [CMSA04]	-	+	+	QoS, price
Jaeger et al. [JRGM04]	-	+	+	QoS, price
Canfora et al. [CP04]	-	+	+	QoS, price
Root cause and impact analysis				
Bodenstaff et al. [BWRJ08]	-	+	-	QoS, price
Ensel and Keller [EK02]	+	-	-	general dependency
Basu et al. [BCD08]	+	-	+	general dependency
HP (DDM) [HP08]	+	-	+	infrastructure monitor relationship, DB client, ...
Activity scheduling				
Tolksdorf [Tol03] classification	+	+	-	time, resource, refinement, ...
Zhou et al. [ZBH08]	+	-	+	data, control flow
Wu et al. [WPSB07]	+	-	-	data, control flow

Legend: Yes (+), No (-)

4

A Concept for Managing Dependencies of Business Services

In service compositions, dependencies exist between the different services. These dependencies originate from the fact that services, which are assigned to a service composition, are implicitly collaborating while contributing to the goal of the service composition. The providers of the collaborating services will not be aware of the different services and how they are collaborating. In fact they may not even be aware that service provisioning occurs as part of a composition. However, the composite service provider creates the composition and is responsible for managing the collaboration. This means, he creates the underlying process and selects and integrates relevant services. He also negotiates SLAs with the service providers, which regulate how each atomic service contributes to the composition. The SLA negotiated with the consumer of the composition guarantees which outcome can be expected from the service, the quality of service provisioning, and its price.

The fact that service dependencies exist implies that they need to be managed appropriately. The concepts presented in this thesis are concerned with the management of service dependencies in the context of the handling of SLAs within service compositions. Three different aspects are considered. First of all, service level agreements need to be negotiated in such a way that proper collaboration is possible between the different atomic services, the goal of the composition can be achieved, and the different SLAs will be fulfilled. A second aspect is that problems during the provision-

ing of one service may influence or even make impossible the provisioning of other services. It is the task of the composite service provider to manage such situations. Finally, there are situations where a stakeholder of the service composition (i.e. atomic service provider, composite service provider, service consumer) needs to renegotiate an existing SLA. This is the case when some replanning takes place at the respective stakeholders side. Here it is important to identify the potential effects of renegotiation on other SLAs before the renegotiation is executed.

This chapter describes the concepts for managing dependencies between services in service compositions. As a first step, Section 4.1 outlines the proposed dependency management solution. In Section 4.2 a formal definition for service dependencies is given. Furthermore, the properties of different types of dependencies are analyzed. As a third step, Section 4.3 discusses the implicit representation of service dependencies in service level agreements. A metamodel is presented for capturing service dependencies explicitly. The implicit and explicit representation of dependencies form an important basis for the management of dependencies. In Section 4.4 the dependency management approach is discussed in detail. Finally, Section 4.5 presents an architecture, which realizes the presented dependency management concepts.

4.1 Solution Overview

In order to manage the dependencies in service compositions an approach was developed, which supports composite service providers in their work. The dependency management approach is illustrated in Figure 4.1.

Information, which is relevant for the management of dependencies, is contained in the workflow description of the composite service as well as in the negotiated ASLAs and the CSLA. The workflow description describes the sequence flow and thus provides information regarding the order in which services are executed. The SLAs contain information regarding e.g. the negotiated QoS attributes. In order to enable the easier handling of this information it is captured in a dependency model. The *Dependency Analysis* components realize a semi-automatic approach for creating a dependency model. They analyze the underlying process of a composite service and the SLAs negotiated for each of the contained services as well as the composite service. Based on this information different dependencies (e.g. regarding time, resource, QoS, price) can be discovered automatically. However, there are other dependencies (e.g. regarding location) that need to be modeled explicitly. This is also supported by the *Dependency Analysis* components.

The creation of the dependency model happens at the design time of the service composition. As a result a formal dependency model is created.

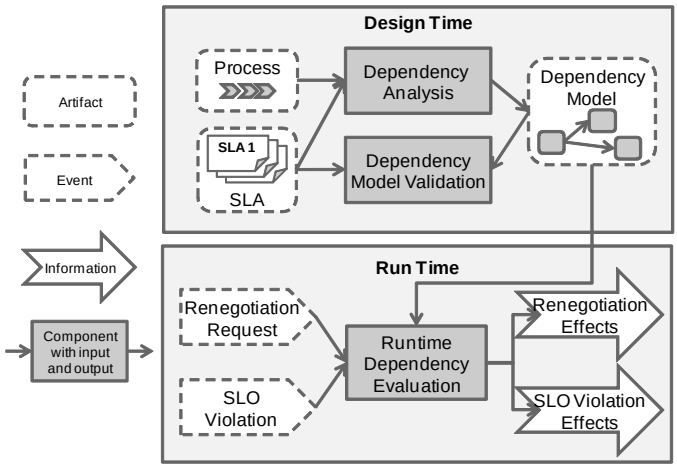

Figure 4.1: Overview of dependency handling

The dependency model captures information regarding the different services of the composition, the SLAs negotiated for these services, and details about the different dependencies.

During design time the dependency information is used by the *Dependency Model Validation* component to validate the negotiated SLAs, i.e. to ensure that the services will be able to work together properly and achieve the common goal. During the time of service provisioning the dependency information is used to evaluate the effects of SLO violations and SLA renegotiation requests. Upon the occurrence of SLO violations the *Runtime Dependency Evaluation* components evaluate the violation information with the goal to determine the services which are affected by the violation. Similarly, the information from SLA renegotiation requests is evaluated using the dependency model to determine which services are affected by this renegotiation.

In the following sections the service dependency model, the creation of the dependency model and its usage for managing service level agreements, as well as the architecture of the approach are described in more detail.

4.2 The Nature of Service Dependencies

Service dependencies are the underlying cause for the fact that events regarding single services influence other services in service compositions. In this section a formal definition for service dependencies is provided. The

nature of these service dependencies is analyzed with regard to their occurrence within a composition as well as symmetric and transitive properties. Furthermore, different classes of dependencies are described and used to classify the different types of service dependencies mentioned in Section 2.5.1. The different properties of service dependencies form the basis for the discovery and modeling of dependencies as well as their evaluation during service provisioning.

4.2.1 Defining Dependencies

Dependencies play an important role in several disciplines including not only computer science (e.g. [BCD08, EK02, WPSB07, ZBH08, Tol03, BWRJ08]) but also economics and social sciences [MC94] as well as project management [PMI08].

A general definition of the word *dependence* can be found in [MFT+09]. The notion of dependence is described as "the state of relying on or being controlled by someone or something else".

The following definition of dependencies between services forms the basis for the work presented in this thesis:

> A service dependency is a directed relation between services. It is expressed as a 1:n relationship where one service (dependent) depends on one or multiple services (antecedent). A service s_1 is dependent on a service s_2 if the provisioning of service s_1 is conditional to the provisioning of service s_2, i.e. if a property of service s_1 is affected by a property of s_2.

Let $S = (s_1, s_2, \ldots, s_n)$ be a set of services within a service composition including the composite service itself. A service dependency can be seen as a tuple Dep: $\{S_a, s_d, kpi_a, kpi_d, type, desc\}$, where

- $S_a \in S$, is a set of antecedent services from the set of all services of the composition;
- $s_d \in S$, is the dependent service from the set of all services of the composition; $s_d \notin S_a$;
- kpi_a, identifies the kpi of the antecedent services regarding which the dependency occurs;
- kpi_d, identifies the kpi of the dependent service regarding which the dependency occurs;

- $type \in (time, resource, location, QoS, price)$, identifies the type of the dependency;
- $desc$, is a type-specific dependency description. More details about how different dependencies are described can be found in Section 4.3.2.

Dependencies take effect when the properties of services, which are regulated by SLAs, are negotiated, renegotiated, or violations of service level objectives occur during service provisioning. In such situations it is necessary to manage these dependencies.

4.2.2 Horizontal and Vertical Dependencies

In service compositions dependencies occur either between the atomic services within the composition or between an atomic service and the composite service. The first class of dependencies is defined as horizontal dependencies, the latter one as vertical dependencies.

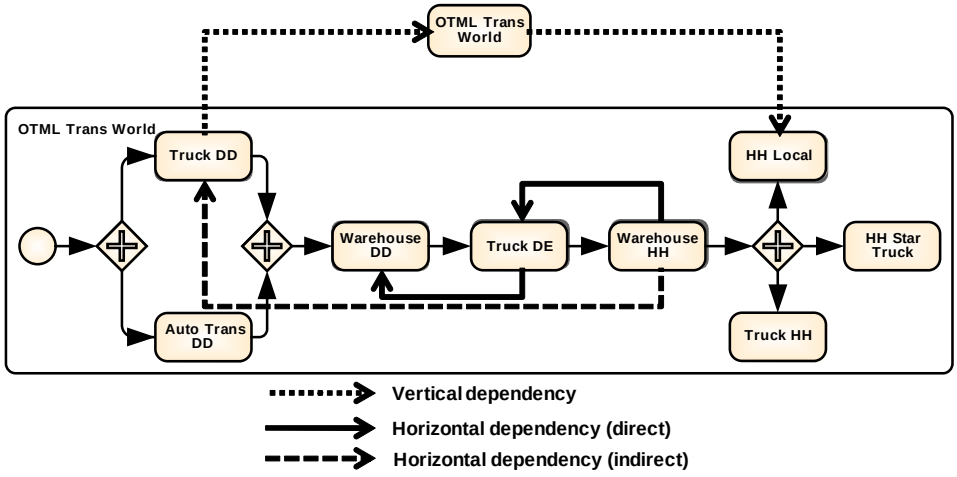

Figure 4.2: Horizontal and vertical dependencies

Horizontal Dependencies

Horizontal dependencies occur between the atomic services of a composition. They do not necessarily have an influence on the consumer of the overall composition. They rather affect the execution of services within compositions. The occurrence of problems during the execution of an atomic service may endanger or even render impossible the execution of

other atomic services. Similarly, the renegotiation of service properties will require the adaptation of the respective dependent properties of other atomic services.

> Dependencies that occur between the atomic services of a service composition are defined as **horizontal dependencies** because they affect services on the same hierarchical level of composition.

In cases where horizontal dependencies are managed properly, they will not show to consumers of the composition. This would be the case if some services could compensate for the problems caused by a SLO violation. Nevertheless, it might not be possible to completely compensate for occurring problems. In such cases the effects of the problem will be propagated throughout the whole composition and thus affect the consumer of the composite service.

Horizontal dependencies can be further classified into direct and indirect dependencies. Direct dependencies occur between two services. Indirect dependencies occur due to a transitive relationship between services regarding their dependency. In Figure 4.2 direct as well as indirect horizontal dependencies are depicted based on an extract from the logistics scenario workflow description.

Vertical Dependencies

Dependencies between services will not only occur within compositions. Problems of an atomic service can have direct effects on the overall composition. Similarly, the failure of a consumer to meet his obligations or the renegotiation of service level objectives can affect the atomic services within the composition.

> Dependencies that occur between the atomic services of a composition and the composition itself are defined as **vertical dependencies** because they affect services on different hierarchical levels of composition.

When SLO violations occur with regard to service properties having vertical dependencies, this may directly lead to SLO violations of services on a different hierarchical level. If an atomic service violates its SLA, this affects the composite service. If a service consumer violates his obligations, this affects the atomic services of the composition. The same is true for the renegotiation of service properties. Two vertical dependencies are depicted in Figure 4.2.

4.2.3 Dependency Classes

Section 3.3.2 briefly outlined different classes of dependencies based on the work of Malone and Crowston [MC94]. While Malone and Crowston discuss these dependency classes on a very general activity level, their work can be applied to services, which are also a type of activity. In the following sections the different dependencies are described in more detail in the light of services.

Task - Subtask Dependencies

Task-subtask dependencies occur when a process of several services (the subtasks) is created as a refinement of another service (task). In such a top-down approach the goal of a composite service is broken down into subgoals, which are achieved by other services. Similarly, several services can be composed in a bottom-up approach to create a more complex service.

The task-subtask dependency causes vertical dependencies between the composite service and its atomic services. Thus, the decomposition of a composite service into atomic services or the creation of a composite service from atomic services requires the regulation of dependencies between a composite service and its atomic services. Since a dependency is a directed relationship (see Section 4.2.1) it is important to consider the direction of this relationship. Depending on whether the dependency is a result of decomposing a service composition into atomic services or aggregating atomic services to create a new composition, the direction of the dependency relation varies. In the first case the atomic service attributes depend on the composite service values. In the second case the composite values depend on the atomic values.

When the composite service attributes such as QoS and price are dependent on the respective attributes of the atomic services, the composite service attribute values are determined via an aggregation of atomic service attribute values. Various approaches to the automatic aggregation of service attribute values with regard to workflow QoS have been investigated (e.g. [CMSA04, JRGM04]) by the scientific community. The decomposition of composite QoS and price values into atomic ones remains an open research issue according to [LF08].

A second issue is the fact that the composite service input and output interfaces need to be reflected by the respective subtasks input and output interfaces. The input and output interfaces concern different aspects, such as input and output resources, start and end time, as well as start and end location of a service. When the composite service is broken down into sev-

eral services it is not necessary for an atomic service to reflect the composite service interface. Instead, several services may reflect parts of it.

Producer - Consumer Relationships

Two services have a *producer-consumer* relationship (dependency) when the outcome of one service is required by another service in order for the second service to be able to start working. This is the case for the input resources of a service. The consumer of a resource depends on the provider of the resource. A similar relationship occurs when a service with pre-conditions depends on a service that fulfills these pre-conditions with its post-conditions. The handling of pre- and post-conditions is not the focus of this thesis.

Producer-consumer relationships occur as horizontal dependencies. The two services do not need to be directly connected in a workflow in order for this type of dependency to occur. It is sufficient if they are connected via a valid path in the workflow. This ensures that the dependent service can be reached from the service it depends on.

Simultaneity Constraints

The execution of services may be restricted by certain temporal constraints between services. These constraints are called *simultaneity constraints*. They express, for example, that two services need to occur at the same or at a different time, or that one service needs to be executed after another one has finished. Simultaneity constraints occur as horizontal dependencies between services. They are often a result of a producer-consumer relationship or of a shared resource dependency. As such, they often occur between services, which are connected via a path in the process workflow. However, this is not always the case. Also parallel activities may have requirements regarding their start or end time.

Shared Resources

If two services need access to the same resources, which are available for only one service at a time, there is a shared resource dependency between them. Dependencies regarding shared resources occur as horizontal dependencies. A shared resource dependency can result in simultaneity constraints, which were described above. The explicit handling of shared resource dependencies is not supported by this work.

4.2.4 Symmetry and Transitivity of Service Dependencies

The symmetric relation R is defined as a binary relation between two elements e_1 and e_2 from a set E. It is formally defined as follows:

$\forall e_1, e_2 \in E, e_1 R e_2 \Rightarrow e_2 R e_1$.

A dependency Dep between two services s_1, s_2 is symmetric if $\forall s_1, s_2 \in S, s_1 Deps_2 \Rightarrow s_2 Deps_1$. An example of a symmetric dependency in the logistics use case is a location dependency, where

$\{TruckDD, WarehouseDD, endLocation, startLocation, location, equals\}$

$\Rightarrow$

$\{WarehouseDD, TruckDD, startLocation, endLocation, location, equals\}$.

The transitive relation R is defined as a relation between the elements e_1, e_2, and e_3 from a set E. It is formally defined as follows:

$\forall e_1, e_2, e_3 \in E, e_1 R e_2 \wedge e_2 R e_3 \Rightarrow e_1 R e_3$.

A dependency Dep is transitive if $\forall s_1, s_2, s_3 \in S, s_1 Deps_2 \wedge s_2 Deps_3 \Rightarrow s_1 Deps_3$. An example of a transitive dependency is a resource dependency, where

$\{TruckDD, WarehouseDD, res_{out}, res_{in}, resource, (R1, R2, R3)\}$

$\wedge$

$\{WarehouseDD, TruckDE, res_{out}, res_{in}, resource, (R1, R2, R3)\}$

$\Rightarrow$

$\{TruckDD, TruckDE, res_{out}, res_{in}, resource, (R1, R2, R3)\}$.

4.2.5 A Classification of Service Dependencies

Dependencies between business services occur with regard to concrete service properties. They include quality of service (QoS) properties (e.g. response time, availability, throughput, encryption level), price information, start and end time as well as start and end location of service execution, and handled resources (e.g. data or goods). Dependencies regarding these properties can be classified with respect to the different classes of dependencies described above. Based on the dependency class, dependencies occur as horizontal dependencies, vertical dependencies, or both. Their properties with regard to symmetry and transitivity are also analyzed. The different types of dependencies are illustrated based on an extract of the logistics use case. Table 4.1 provides an overview of the different dependency types and their classification with regard to horizontal and vertical dependencies as well as symmetry and transitivity. Please note that the symbol

"o" is used to show that no general statement can be made regarding the dependency type and the relevant property. Symmetry and transitivity are both dependent on the relevant operator, which is used to describe the dependency. For example, the location operator *equals* is transitive while the operator *not_equals* is not transitive.

Table 4.1: Dependency classification

Dependency class	**Property**	**Horizontal**	**Vertical**	**Symmetric**	**Transitive**
Task-subtask	Response time	-	+	-	-
	Availability	-	+	-	-
	Throughput	-	+	-	-
	Encryption level	-	+	-	-
	Price	-	+	-	-
	Resource	-	+	-	-
	Start/end time	-	+	o	-
	Start/end location	-	+	+	-
Producer-consumer	Resource	+	-	-	+
	Start/end location	+	-	+	o
Simultaneity constraints	Start/end time	+	-	o	o

Legend: Yes (+), No (-), Operator specific (o)

Quality of Service:

The QoS attributes of services (e.g. response time, availability, throughput, encryption level) describe different quality aspects of service provisioning. QoS dependencies occur due to task-subtask dependencies. Within a service composition QoS dependencies occur as vertical dependencies. Violations or changes to the QoS values of atomic services affect the composite service. Changing a composite QoS value requires the modification of all the respective atomic QoS values. QoS dependencies between different atomic services in a composition do not occur unless there is an explicit statement of such a dependency, e.g. via a precondition of a service. Dependencies with regard to QoS attributes are not symmetric or transitive. Figure 4.3 presents an example of a QoS dependency based on the logis-

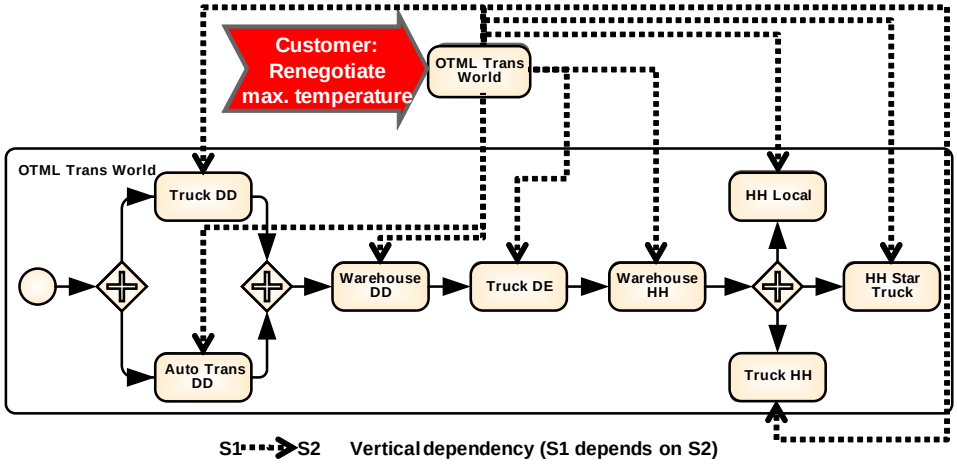

Figure 4.3: Dependencies for QoS renegotiation

tics use case. A relevant QoS attribute in logistics is e.g. the maximum temperature of goods transported.

Price Information:

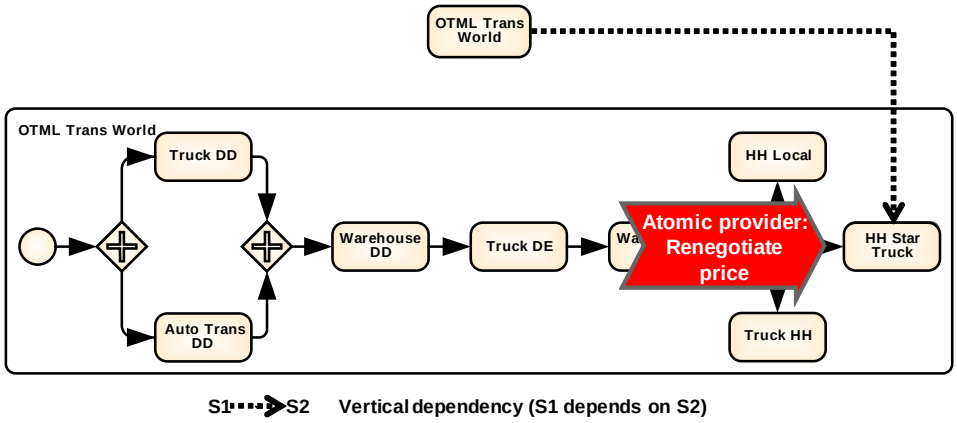

Figure 4.4: Dependencies for price renegotiation

A price dependency exists when the price of a service depends on the price of one or more other services. For example, the price of a composite service can be calculated based on the prices of its atomic building blocks (see Figure 4.4), or the prices of the atomic services are negotiated based on breaking down the price of the composite service. Price dependencies occur due to task-subtask dependencies and therefore only occur as vertical dependencies. While the price of a service is not a measurable attribute of

provisioning and thus cannot be violated, it is possible to renegotiate the price of a service. Therefore it is also important to consider this attribute. The price dependency is not considered to be symmetric or transitive.

Time Information:

Time dependencies express temporal relationships regarding the execution of services. They can occur as task-subtask relationships or simultaneity constraints. Time dependencies occur as horizontal as well as vertical dependencies.

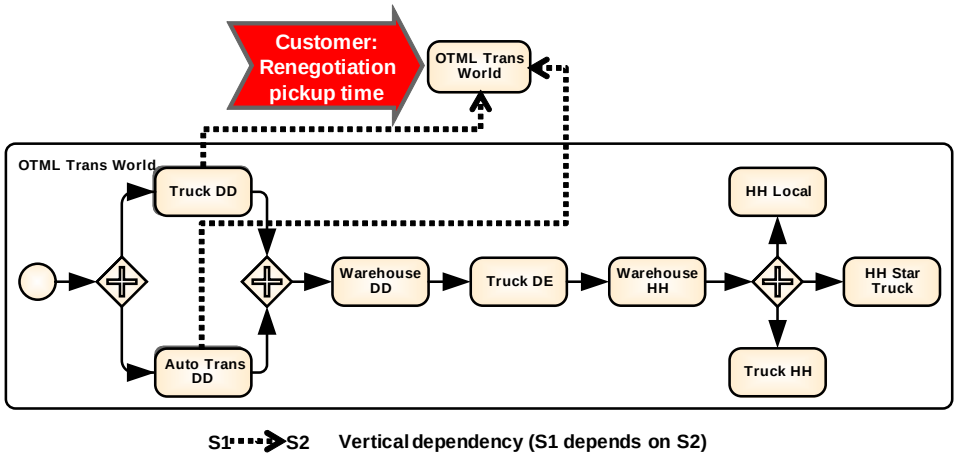

Figure 4.5: Dependencies for pickup time renegotiation

The example presented in Figure 4.5 illustrates two task-subtask-based time dependency. If the consumer of the composite service renegotiates the time when goods should be picked up, this affects the first atomic services in the composition.

Times are expressed as intervals that represent the time period of the service execution. They have a start time and an end time. The time relationship between the time periods of the execution of two services can be expressed via different operators (e.g. before, after) as defined by Allen [All83], or the ones used in the area of project management (e.g. finish-to-start) [PMI08]. The different time operators are described in more detail in Section 4.3.2.

A general categorization of time dependencies with regard to symmetry and transitivity cannot be made. It is rather necessary to distinguish the different time operators. The *equals* operator, for example, is symmetric, while the other time operators are not. Similarly, only some time operators have a transitive nature. This is the case for *before*, *starts*, *equals*, *during*,

finishes, *after*, *started-by*, *contains*, *finished-by* as well as *start-to-start*, *finish-to-start*, and *finish-to-finish*. The operators *meets*, *met-by*, *overlaps*, *overlapped-by*, and *start-to-finish* are not transitive.

Location:

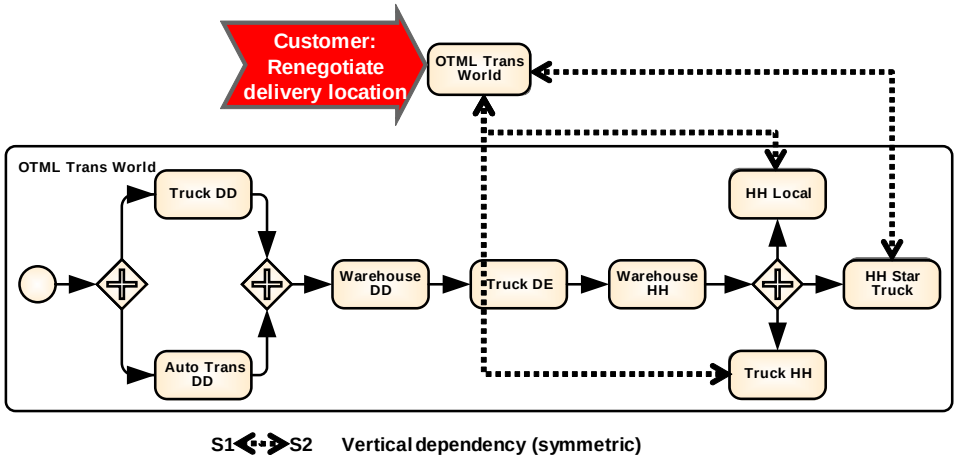

Figure 4.6: Dependencies for location renegotiation

Location constraints describe local dependencies between two services, meaning they need to be executed at the same or at a different location. Location constraints occur as a result of task-subtask relationships as well as a result of producer-consumer relationships. Thus, there are horizontal and vertical location dependencies.

In Figure 4.6 three location dependencies based on a task-subtask relationship are presented. The consumer of the composition decided to renegotiate the location of delivery. This affects several atomic providers of the composition.

Location dependencies can be expressed via the operators *equals* and *not_equals*. Both operators are symmetric. In case that a location dependency is expressed via the *equal* operator, it is transitive. This is not the case for the *not_equal* operator.

Resource:

Two services have a resource dependency when the availability of a resource needed by one service, is dependent on another service. Resources include electronic data and documents as well as material goods. Resource dependencies occur due to provider-consumer relationships as well

as task-subtask dependencies. Thus, they occur as horizontal and vertical dependencies.

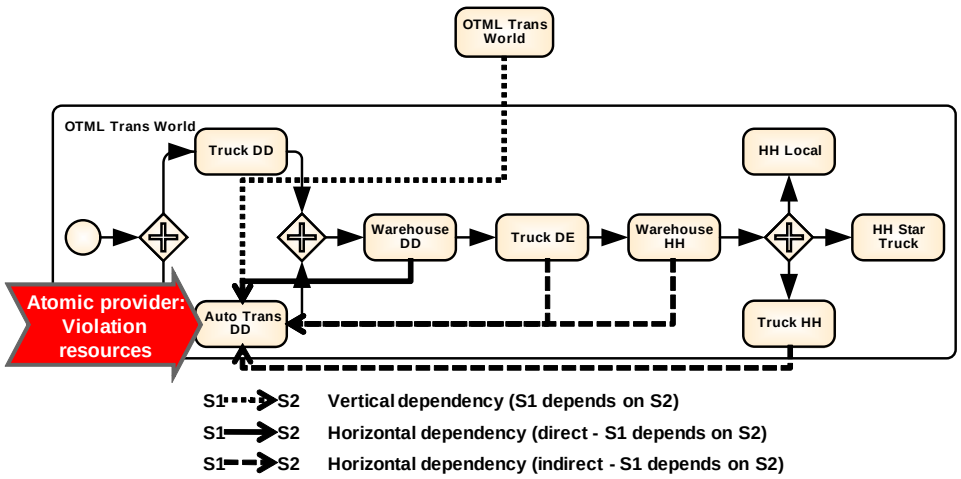

Figure 4.7: Dependencies regarding resource SLA violation

The example in Figure 4.7 shows the dependencies of a SLA violation with regard to the resources handled by service AutoTrans DD. If the goods being transported are damaged, the SLAs for all services, which should have handled the same resource later in the process, need to be adapted. This is an example of a provider-consumer based dependency.

A resource dependency between two services is expressed by listing the resources regarding which a dependency exists. Resource dependencies are not symmetric but have a transitive nature.

4.2.6 Relevance of Dependency Management

The types of dependencies, which occur in service compositions, depend on the nature of the involved services. They vary regarding different aspects, e.g. the time needed for executing a service, the degree of automation of service execution, and the handling of data or physical goods. There are services that are fully automatized and have a relatively short duration of execution. Single service requests are not managed with respect to the underlying resources, which support the execution in a way that the handling of a specific call is assigned to specific resources prior to a service call. This is rather done dynamically using e.g. load balancing mechanisms. There are also services which are longer-running, require distinct resources (e.g. machines, computers, and human personnel) for their execution, and often involve the handling of physical goods in addition to electronic data.

Different QoS aspects are of relevance for most services. The specific QoS attributes differ depending on the nature of the service as well as on its application domain. Similarly, resource dependencies are relevant for all types of services that involve data or physical goods (resources) being passed from one service to another.

Time dependencies are mainly important for services that specify a start and end time for service provisioning. For fully automatized services having a short execution time this is typically not the case because their service provisioning can start at any time. However, the execution of services having a long-running execution time is usually bound to specific resources, e.g. for transportation or production services. For such services the management of time dependencies is very important due to the fact that small violations of time constraints may have a large impact on other services.

Location dependencies mainly occur between services where the execution involves physical resources being passed from one service to another. Nevertheless they are also relevant for automatized services with location constraints on other services regarding the execution (e.g. a service cannot be composed with services which are hosted in a certain country).

The focus of this thesis is on services which are long running and where service provisioning involves manual steps, execution supported by machines, and physical goods being handled. However, other services (e.g. automatized and short running) are not excluded.

4.2.7 Summary

When business services are composed to service compositions, vertical and horizontal dependencies occur between the different services. Dependencies between services occur due to dependency classes, i.e. task-subtask and producer-consumer relationships, as well as due to simultaneity constraints.

Services are dependent on other services with regard to different service attributes such as QoS parameters, price information, start and end time as well as location of execution, and resources. Dependencies regarding these attributes are caused by the different dependency classes. Depending on the underlying cause of a dependency it occurs as a horizontal or vertical dependency and may have a symmetric or transitive nature. An overview of this classification is provided in Table 4.1.

4.3 Implicit and Explicit Representation of Service Dependencies

Services in compositions collaborate to achieve the goal of the composition. Due to this collaboration dependencies exist between the different services. Information about these dependencies is available, e.g. in the workflow structure as well as the SLAs of the different services. Further dependency information may exist as domain knowledge and is not captured in any service-related artifact. To enable the easier handling of dependencies, the dependency management approach captures implicit dependency information and represents it explicitly in a formal dependency model.

This section describes SLAs in more detail, since they are important service artifacts, which contain information relevant for dependency handling. Furthermore, the dependency model used for capturing dependency information in an explicit way is presented.

4.3.1 Dependency Related Information in SLAs

The information contained in SLAs forms the basis for the discovery of dependencies. Thus, the dependency management approach requires the formal representation of relevant information in SLAs. Necessary information includes:

- Required and provided resources of each service
- Start and end time of service provisioning
- Start and end location of service provisioning
- Quality of service description of service provisioning
- Price information

The structure of SLAs was explained in Section 2.2. In this section the expression of SLAs is based on the WS-Agreement specification (see Section 3.1.5). This sample implementation enables a more precise illustration of which information is necessary for the dependency management approach and how it is represented. However, other SLA languages could be used as well, as long as all necessary information is contained.

While a SLA contains a variety of important information, the following sections describe the information which is necessary for the dependency management approach.

Service Interface Description As part of the functional description of a service, the interface of the service to the consumer is described. The service interface description allows the specification of information on the start and end time, as well as the location of service provisioning. Furthermore, it provides information regarding resources that are required from the consumer for service provisioning, as well as resources provided as the result of service provisioning.

The XML code sample presented in Listing 4.1 shows a sample *inputInterface* element. It specifies one resource, which needs to be provided by the customer. Further information about start time and location of service provisioning is described.

```
<inputInterface>
 <ports>
  <port>
   <resourceType>Export document</resourceType>
   <resourceName>OTML export form</resourceName>
   <resourceID>Export-declaration-ED-316-VZ1</resourceID>
  </port>
  ...
 </ports>
 <time>2009-11-09T17:00:00+0100</time>
 <location>Teegasse 83, 01159 Dresden, Germany</location>
</inputInterface>
<outputInterface>
...
</outputInterface>
```

Listing 4.1: Sample service interface description

Service Description For each service a comprehensive service description is provided as part of the SLA. One important aspect is the description of price information as part of the *marketing* element. The XML code sample presented in Listing 4.2 illustrates the price description of the composite logistics service including information about the currency, taxes, and the specific price for service provisioning.

```
<marketing>
 <price>
  <payPerUse>
   <priceName>OTMLTransWorld Standard Price</priceName>
   <priceCurrency>EUR</priceCurrency>
   <priceVat>19.0</priceVat>
   <priceUnit>5000.0</priceUnit>
  </payPerUse>
 </price>
</marketing>
```

Listing 4.2: Sample price description

QoS Information in SLAs Information about the different quality of service parameters for a service is expressed as part of *GuaranteeTerm* sections in the SLA document. Each QoS parameter is identified by the *KPIName* element. The negotiated value is expressed by the *value* element, which is part of the *KPITarget*. This information is relevant for the calculation of QoS dependencies.

```
1 <GuaranteeTerm Obligated="ServiceProvider"
2  Name="OTMLTransWorld_H378G8329-J378-9U47-2ZU3-
       U378ZE8329EU_temperature_Guarantee">
3  <ServiceLevelObjective SLOID="Temperature-SLO-1">
4   <monitoringURI>...</monitoringURI>
5   <KPITarget>
6    <KPIName>maxTemperature</KPIName>
7    <Target xsi:type="SimpleTarget">
8     <operator>lte</operator>
9     <value xsi:type="xs:double">15.0</value>
10   </Target>
11  </KPITarget>
12 </ServiceLevelObjective>
13 </GuaranteeTerm>
```

Listing 4.3: Sample guarantee term description

4.3.2 Service Dependency Model Concept

One major building block of the approach to manage service dependencies is a dependency model. It captures the dependencies between the different services of the service composition in an explicit way in order to allow efficient handling of dependencies at runtime. The model is created at design time from the information negotiated in the SLAs for the different services in a composition, as well as from structural information of the workflow process underlying the composite service.

In this section the advantages and disadvantages of different approaches for expressing service dependencies are discussed. Based on this discussion a metamodel for a dependency model is presented and its features are explained.

Approaches to Modeling Dependencies

In Chapter 3.3 a number of approaches to modeling service dependencies were described as part of the related work. In this section four different approaches for representing dependencies are compared. A dependency graph, a dependency matrix, and special purpose dependency mod-

els based on an ontology or a metamodel are analyzed with respect to several requirements described in Section 2.5.2:

- DR1: Dependencies need to be expressed with a granularity of single service properties.
- DR2: The representation of dependencies needs to support the expression of multiple dependencies between two services.
- DR3: The representation of dependencies needs to support the expression of multiple dependencies of the same type.
- DR4: The representation of dependencies needs to support the expression of type-specific dependency descriptions.
- DR5: The representation of dependencies needs to support the expression of dependencies between services with 1:1 and 1:n relationships.
- MAR4: The approach should support application domain experts in their work.

Dependency Graphs A possible approach for representing dependencies is the application of a dependency graph. Graph theory is a field of mathematics where graphs express the relationship between a pair of entities from a collection of entities.

Clark and Holton [CH91] define a graph as follows: "A graph G = (V(G), E(G)) consists of two finite sets: V(G), the vertex set of the graph, [...], which is a nonempty set of elements called vertices, and E(G), the edge set of the graph, [...], which is a possibly empty set of elements called edges, such that each edge e in E is assigned an unordered pair of vertices (u,v), called the end vertices of e."

In a dependency graph entities having dependencies are expressed as vertices (nodes). Dependencies are expressed as edges. This approach is used e.g. by [ZBH08]. Edges can be annotated e.g. with weight or cost information. This is used by different algorithms for the calculation of specific paths within the graph.

The representation of dependencies as edges requires the annotation of edges to allow for typification of dependencies according to the different dependency types. A set of two vertices can be connected by several edges, which assures that different dependencies can be expressed between two services. At the same time a vertex can be connected with different vertices via edges of the same type. This way it is possible to express that one service has dependencies of the same type with different other services. The

expression of specific properties of each dependency is not possible, unless the edges are complex structures, which allow a sufficiently complex annotation with information. This is not the case for standard graphs and requires specific extensions of the notation. A further limitation of a graph, where services are expressed as nodes and dependencies as edges, is that n-ary dependencies, i.e. a single dependency, which expresses a relation between one service on one side and multiple services on the other side, cannot be represented. This is due to the fact that one edge can only connect two nodes and thus only express a relationship between two services. This issue can be overcome by applying a bipartite graph. In [Ore67] a bipartite graph is defined as a graph with two disjoint vertex sets, where an edge always connects a vertex from one set with a vertex from the other set. Services as well as dependencies can be represented as vertices. Service vertices and dependency vertices form two disjoint sets.

Dependency Matrix Another form of representing dependencies is a dependency matrix. A matrix is a system of m times n elements $M = (m, n)$, which are arranged in m rows and n columns [BSMM00]. A dependency matrix corresponds to an adjacency matrix where the number of rows and columns is equal, i.e. $M = (n, n)$, and where n is equal to the number of entities to be related to each other. The different entities label the vertical and horizontal axes of the matrix. The relationship between two entities is represented as an entry in the intersecting cell of the matrix. A matrix is a useful representation for recognizing e.g. cyclic dependencies between entities [SJSJ05].

The expression of different dependency types can be achieved by defining different values, which are used to fill the intersecting cells in the matrix. Using a matrix, it is possible to express dependencies between one entity and multiple others, i.e. one service has dependencies of the same or of a different type with other services.

However, there are some major drawbacks of a matrix with regard to the requirements described in Section 2.5.2. It is not possible to express complex details about a specific dependency between two entities. Furthermore, it is not possible to represent multiple dependencies of different types between two entities. Only one dependency can be expressed between two services. Also, it is not possible to express n-ary dependencies. Since these three aspects are important requirements for this work, a matrix representation is not suitable.

Special Purpose Dependency Model An alternative way of representing dependencies would be the creation of a model explicitly designed to capture dependencies. There are different ways of formalizing such a

dependency model. Two options, a metamodel for creating dependency model instances and an ontology for expressing dependency models, are discussed.

Metamodel: A metamodel is a specification of a language for describing models. It is thus a "...model of models ..." [MM03]. It specifies the classes, attributes, and relationships between classes, which are a part of such a language. Using a metamodel approach, a domain specific language for creating dependency models can be defined.

There are different languages, which allow the creation of metamodels, e.g. *UML* [OMG09], *XML Schema* [TBMM04], and *Ecore* [SBPM09]. The *Unified Modeling Language* is a specification from the Object Management Group (OMG). It defines a graphical notation for modeling different types of models (e.g. class and component diagrams). XML Schema is a W3C Recommendation, which allows the specification of XML document structures. Ecore was developed as an implementation of *Essential MOF*, which is a subset of the OMG *MOF* specification [OMG06] for metadata management. Ecore is a part of the Eclipse Modeling Framework (EMF) [SBPM09].

The different languages have the same expressivity, i.e. metamodels of the same complexity can be expressed. It is also possible to transform a metamodel represented in one language (e.g. XML Schema) into another representation (e.g. Ecore).

In order to capture information about service entities and dependencies, such entities can be represented as classes or elements, along with their respective properties and relations. A metamodel can be designed to enable the representation of dependency models of arbitrary complexity specifically tailored to the requirements of the domain.

An advantage of using a metamodel approach is its very good support for creating modeling tools (e.g. based on Eclipse EMF and GMF) that are directly targeted to the needs of domain experts, as well as the option to generate an API for handling models. Furthermore, specific constraint languages such as the Object Constraint Language (OCL) [OMG05] allow the expression of constraints on classes, attributes, and their relations. These constraints are specified with regard to the metamodel elements and are used to validate models.

Ontology: According to Gruber [Gru93] an ontology can be seen as "...an explicit specification of a conceptualization ...", i.e. the specification of abstracted knowledge about the world. It describes different objects of a domain, their properties, and relationships between them.

Different languages exist, which allow the specification of ontologies. Two ontology languages, which are W3C Recommendations, are the *Web Ontology Language (OWL)* [MH04], and the *Resource Description Framework Schema (RDFS)* [BM04].

The OWL language allows the modeling of classes, individuals, and properties. A *class* represents information about a set of individuals with common features. Class - sub-class relations can be used to describe a hierarchy of classes. The single instances of a class are called *individuals*. In order to specify relationships between two individuals *object properties* are used. These object properties can be further described by features such as *inverse*, *symmetric*, *asymmetric*, and *disjoint*. *Data properties* define properties of individuals with respect to data types.

One way of representing dependencies between services using an ontology would be the modeling of services as individuals and the dependencies as object properties. The usage of object property features would then allow the further specification of certain features of each property. However, this approach does not allow a detailed description of dependencies as required in Section 2.5.2. In order to specify more details about dependencies it would be possible to express dependencies as classes with specific properties.

While ontologies allow the specification of dependency models according to the requirements described in Section 2.5.2, they also have a major drawback. Available tools for creating ontologies, e.g. protégé [BIR] and OntoStudio [Gmb09], are only suited for ontology experts rather than application domain experts, who are interested in the management of service compositions and the respective service dependencies.

Discussion In this section four different approaches for realizing a dependency model were presented and their advantages and disadvantages discussed. An overview of the results is presented in Table 4.2.

The creation of a dependency model as a dependency graph or dependency matrix is suitable for expressing dependencies with limited information. However, they do not provide sufficient expressivity for dependency models needed for the management of service dependencies in service compositions.

The metamodel as well as the ontology approach to creating a special purpose dependency model are both expressive enough to realize the requirements for a dependency model. An important aspect is, however, the tool support for the creation of the dependency model. There is a need for tools, which support industry domain experts in their work of modeling the dependency model during development time of the composition. No knowl-

edge about specialized technologies (i.e. ontology or metamodel) should be required (see requirement MAR4). Instead, the complexity of underlying technologies must be hidden. This is easier to achieve using the metamodel approach, which supports the easy creation of tools based on Eclipse GMF. Besides that, the metamodel approach supports the generation of a Java API for managing dependency model instances based on Eclipse EMF. Thus, the metamodel approach was selected as the most suitable option for designing a language that allows the creation of dependency models.

Table 4.2: Comparison of dependency model approaches

Requirement	Graph	Matrix	Ontology	Metamodel
DR1	+	+	+	+
DR2	+	-	+	+
DR3	+	+	+	+
DR4	-	-	+	+
DR5	-	-	+	+
MAR4	-	-	-	+

Legend: Fulfilled (+), Not fulfilled (-)

The Dependency Model

A metamodel for creating dependency models was developed as part of the dependency management approach. The dependency model consists of a number of different elements and attributes, which describe the involved services and their dependencies. Dependencies occur with regard to services. They have a number of attributes, which specify the service level objectives regarding which the dependency occurs and a detailed description of the dependency. A dependency model consists of two core parts, which represent the services involved in a composition as well as the different dependencies that exist between them. An overview on the model is presented in Figure 4.8. In the following sections the elements of the model as well as their attributes and relations to other elements are described.

DependencyModel The *DependencyModel* element is the root element of the dependency model. It has a unique *id* and references the composite service and the SLA for the composite service underlying the created dependency model. The DependencyModel furthermore contains elements representing the services and dependencies in the model.

Attributes:

- **id:** A dependency model has 1 unique id.

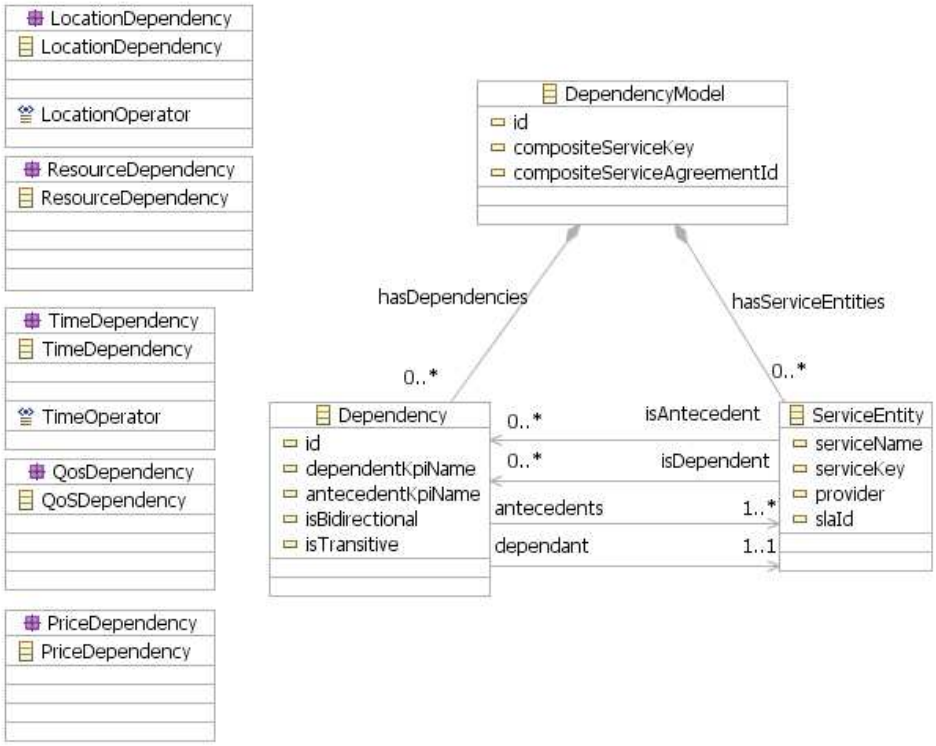

Figure 4.8: Dependency model class diagram

- **compositeServiceKey:** A dependency model references 1 composite service via its service key.
- **compositeServiceAgreementID:** A dependency model references the composite service SLA, regarding which the model was created.

Relations:

- **hasServiceEntities:** A dependency model has 0...* *ServiceEntities*.
- **hasDependencies:** A dependency model has 0...* *Dependencies*.

Service Entity The *ServiceEntity* element is used to represent the different services within a service composition as well as the composite service. Each service having a dependency with another service is represented. The different ServiceEntity elements are referenced by dependency elements.

Attributes:

- **serviceName:** The name of the service. This is not a unique identifier.

- **serviceKey:** The unique identifier of a service.
- **provider:** The provider attribute identifies the provider of the service.
- **slaID:** The unique identifier of the service level agreement, which specifies the service parameters of the service regarding which dependencies occur to different services.

Relations:

- **isAntecedent:** A service entity has the role of an antecedent in 0...* *Dependencies*.
- **isDependent:** A service entity has the role of a dependent in 0...* *Dependencies*.

Dependency The abstract *Dependency* element represents the different dependencies between services within a service composition as well as dependencies with the composite service. A dependency has a unique *id*. It may be uni- or bi-directional as well as of a transitive nature. A dependency references the respective service level objectives of the antecedent and dependent regarding which the dependency occurs. This reference is realized by naming the respective key performance indicator (KPI). Each dependency element references the services, which are the dependent or antecedents in the dependency.

Attributes:

- **id:** The unique identifier of the dependency.
- **isBidirectional:** This attribute expresses whether a dependency occurs as uni- or bidirectional.
- **isTransitive:** This attribute expresses whether a dependency has a transitive nature.
- **dependentKpiName:** A reference to the service level objective within the dependent service's SLA regarding which the dependency exists.
- **antecedentKpiName:** A reference to the service level objective within the SLA of the antecedent, regarding which the dependency exists.

Relations:

- **dependent:** A reference to a ServiceEntity element, which has the role of the dependent in this dependency, i.e. it depends on one or more other service entity(s).

- **antecedents:** A reference to one or more ServiceEntity elements in the role of the antecedent, i.e. the services the dependent depends on.

The elements described above represent the core of the dependency model. They are needed independently of the specific dependency to be modeled. Besides that, the dependency model contains five packages with model elements for specific dependency types. The goal was to achieve a modular design of the metamodel, which separates core aspects from very specific ones. Extensions to the metamodel with regard to supporting different types of dependencies would require the modeling of further packages with this specific information. The core of the metamodel would remain stable.

The current metamodel supports the modeling of five different types of dependencies: location, time, resource, QoS, and price. Each of them is represented in a different package.

LocationDependency Package The *LocationDependency* package contains the *LocationDependency* element and an enumeration of location operators.

LocationDependency The *LocationDependency* element represents a location dependency between two services. It extends *Dependency*. A location dependency is described by a *locationOperator* attribute.

Attributes:

- **locationOperator:** A *LocationDependency* has 1 location operator, which describes the location dependency. The location operator is of type *LocationOperator*.

LocationOperator The *LocationOperator* enumeration lists the different location operators, which are available for expressing a *LocationDependency*. It contains the following enumeration literals:

- **equals:** Two locations referenced by a *LocationDependency* need to be equal.
- **not_equals:** Two locations referenced by a *LocationDependency* cannot be equal.

ResourceDependency Package The *ResourceDependency* package contains the element *ResourceDependency*.

ResourceDependency The *ResourceDependency* element represents a resource dependency between two services. It extends *Dependency*. A resource dependency is described by a list of resource IDs, which lists all resources provided by the antecedent and needed by the dependent.

Attributes:

- **resourceIDList:** A *ResourceDependency* has a list of resource IDs regarding which the dependency exists, i.e. it describes the resource dependency.

TimeDependency Package The *TimeDependency* package contains the elements *TimeDependency* and *TimeOperator*.

TimeDependency The *TimeDependency* element represents a time dependency between two services. It extends *Dependency*. A time dependency is described by a *TimeOperator*, which describes the temporal relationship between the two services.

Attributes:

- **timeOperator:** A *TimeDependency* has 1 time operator of type *TimeOperator* that describes the time dependency.

TimeOperator The *TimeOperator* enumeration lists a number of time operators, which are used to describe the time dependency between two services. The times for service execution are expressed as time intervals, i.e. the execution of each service has a duration. A time interval is assumed to have a start time (t_s) and an end time (t_e). In order to express the dependencies between the execution times of two services (S_1) and (S_2) (e.g. the start and end time), the time operators defined in the field of project management [PMI08], as well as the ones defined by Allen [All83], are used. All of them are included in the enumeration. The following sections will explain the different operators and briefly discuss their differences.

In the area of project time management, four different time operators are defined to express temporal dependencies between two project activities (A1, A2) [PMI08]:

- **Finish-to-Start:** Activity A1 needs to be finished in order for activity A2 to start. Related time operators from Allen: *before*, *meets*
- **Finish-to-Finish:** Activity A1 needs to be finished in order for activity A2 to finish. Related time operators from Allen: *overlaps*, *starts*, *during*, *finishes*, and *equals*

- **Start-to-Start:** Activity A1 needs to be started before activity A2 can be started. Related time operators from Allen: *overlaps*, *starts*, *equals*

- **Start-to-Finish:** Activity A1 needs to be started in order for activity A2 to be able to finish. Related time operators from Allen: *overlaps*

The different operators do not specify precise relationships between the start and end times of two activities, but specify the earliest time one activity can be started with regard to another activity.

Allen [All83] defined a set of 13 time operators, which allow the expression of relationships between two time intervals:

- **Before:** Service (S_1) is said to be executed before service (S_2) if the execution of service S_1 finishes before the execution of service S_2 starts: $t_e(S_1) < t_s(S_2)$.

- **Meets:** Service (S_1) is said to meet service (S_2) when the execution of service S_1 finishes at the same time when the execution of service S_2 starts: $t_e(S_1) = t_s(S_2)$.

- **Overlaps:** Service (S_1) is said to overlap service (S_2) when the execution of service S_1 starts before the execution of S_2 and finishes only after the execution of service S_2 was started: $t_s(S_1) < t_s(S_2) \wedge t_e(S_1) > t_s(S_2)$.

- **Starts:** Service (S_1) is said to start service (S_2) when the execution of service S_1 starts at the same time the execution of S_2 starts and finishes before the execution of service S_2 ends: $t_s(S_1) = t_s(S_2) \wedge t_e(S_1) < t_e(S_2)$.

- **During:** Service (S_1) is said to be executed during service (S_2) when the execution of service S_1 starts after the start of the execution of S_2 and finishes before the execution of service S_2 ends: $t_s(S_1) > t_s(S_2) \wedge t_e(S_1) < t_e(S_2)$.

- **Finishes:** Service (S_1) is said to finish service (S_2) when the execution of service S_1 starts after the start of the execution of S_2 and finishes at the same time the execution of service S_2 ends: $t_s(S_1) > t_s(S_2) \wedge t_e(S_1) = t_e(S_2)$.

- **Equals:** The execution time of service (S_1) is said to be equal to the execution time of service (S_2) when the execution of service S_1 starts at the same time the execution of S_2 starts and finishes at the same time the execution of service S_2 ends: $t_s(S_1) = t_s(S_2) \wedge t_e(S_1) = t_e(S_2)$.

The six remaining time operators (*after*, *met-by*, *overlapped-by*, *started-by*, *contains*, *finished-by*) are the inverse operators of the first six operators in the list above and are not detailed any further.

While the goal of the two sets of time operators is very similar, namely the expression of temporal relationships between activities or time intervals, there are also some major differences.

The first difference is that, while Allen's time operators always involve the start and end time of an interval, the time operators from project management are only concerned with either the start or end time of one activity in relation to the start or end time of a second activity.

Another difference between the two approaches is the rigor with which the two types of operators express time relationships. The time operators used in project management allow the expression of more open time relationships. The specification of precise temporal relationships to the degree achieved by Allen is not possible. Allen's time operators allow a very precise expression of temporal relationships between activities. In order to express more open time relationships, as done by the project management time operators, several of Allen's time operators would have to be combined using the logical operator *OR*. An example is the *start-to-start* operator, which states that one service can start any time once another service has started. There are multiple options for how such a time dependency could look like. Allen's time operators *during*, *finishes*, *overlapped_by*, *ends*, and *after* all meet the *start-to-start* time constraint.

PriceDependency Package The *PriceDependency* package contains the *PriceDependency*.

PriceDependency The *PriceDependency* element represents the price dependency of the composite service on multiple atomic services. It extends *Dependency*. A price dependency is described by a formula for calculating the composite service price from the atomic service prices.

Attributes:

- **formula:** A *PriceDependency* has 1 formula that describes the price dependency, e.g. how the composite service price can be calculated based on the respective atomic service values.

QoSDependency Package The *QoSDependency* package contains the element *QoSDependency*.

QoSDependency The *QoSDependency* element represents a dependency of the composite service on one or more atomic services with regard to one specific QoS parameter. It extends *Dependency*. There are many different QoS parameters (e.g. response time, temperature, ...), which are of relevance in different application domains. A QoS dependency is described by an attribute identifying the respective QoS parameter, and a formula for calculating the respective composite service QoS value from the atomic service values.

Attributes:

- **qosParameter:** A *QoSDependency* occurs with respect to one specific QoS parameter, which is identified by this attribute.
- **formula:** A *QoSDependency* has 1 formula, which describes the QoS dependency, i.e. how the respective composite service QoS value can be calculated based on the atomic service values.

4.4 Managing Service Dependencies

The last section described a metamodel for a dependency model, which plays a key role for the management of service dependencies. This section describes a lifecycle for managing service dependencies using a dependency model. The different phases of the lifecycle are illustrated in detail.

4.4.1 Lifecycle of Dependency Models

The lifecycle of dependency models consists of four phases (see Figure 4.9). During the *Creation & Recomputation* phase the dependency model is created based on the process structure and SLA information. Information can be added or changed if conflicts are detected, SLAs change, or the process description is adapted.

The *Validation* phase is necessary to ensure that the created dependency model is valid. It is also necessary to validate the negotiated SLAs in order to check their compliance with the dependency model. If problems are detected either the dependency model or the negotiated SLAs need to be adapted.

During the *Usage* phase the dependency model supports runtime dependency evaluation tasks, such as the determination of SLO violation effects or the handling of SLA renegotiation requests. In the case of renegotiation the model may need to be adapted accordingly.

During the *Retirement* phase the dependency model is discarded.

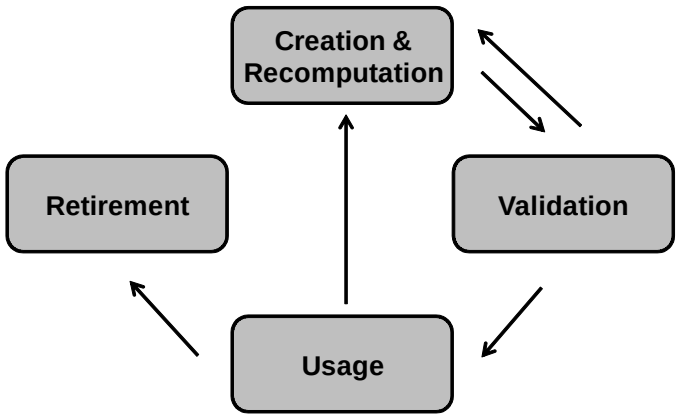

Figure 4.9: Dependency model lifecycle

4.4.2 Creation and Recomputation of Dependency Models

In order to evaluate dependencies between services at runtime (e.g. to evaluate SLA renegotiation requests), information about these dependencies needs to be available. While the availability of implicit dependency information would allow the evaluation of this information, it is advantageous to base the evaluation on an explicit description of dependency information. It allows easier handling at runtime than implicit information that is contained in different artifacts, such as the process description or SLAs. The dependency model creation process realizes the capturing of dependency information, which is available implicitly, and makes this information available explicitly in the form of a dependency model.

There are a number of challenges, which need to be considered for the process of creating a dependency model.

- **Implicit information from different sources:** A dependency model contains explicit dependency information. It is created from information, which is distributed between different artifacts created during the design time of the service composition. These different sources include the composite service workflow description as well as ASLAs and CSLAs negotiated for atomic services and the composite service.
- **Limited dependency information:** The dependency information contained in the different sources is limited. For example, no information is available with regard to temporal dependencies between services, which are not connected via a path in the workflow. Also, no information about location dependencies can be found in the workflow description or SLAs. Thus, a dependency model created based on these sources of information will only contain limited dependency

information. Further information, such as specialized domain knowledge, may need to be considered.

- **Different properties of different dependency types:** Different types of dependencies have different properties regarding their occurrence in a business process (compare Section 4.2.5). These different properties need to be considered during the process of creating the dependency model.

Taking these challenges into consideration, the dependency model creation process was realized as a semi-automatic approach. It consists of the automatic dependency discovery based on the composite service process description and SLA information as a first step, and the explicit modeling of dependencies as a second step. The separation of the process into two parts was necessary, because it is not possible to automatically discover all dependencies based on the process and SLA descriptions. Apart from the implicit dependency information available from these sources, further information is often available as application domain knowledge. Thus, the composite service creator is able to add this information to the dependency model. During the automatic dependency discovery process different algorithms are used to discover the different types of dependencies.

Figure 4.10 presents an overview of the process for dependency model creation and recomputation. At the beginning of the process a dependency model is created, unless it already exists. Following that, the discovery of different types of dependencies is executed. Finally, the dependency model is refined manually by adding information about dependencies which cannot be discovered automatically. The dependency discovery and dependency modeling are described in this section. An emphasis is placed on the strategies for discovering different types of dependencies. Furthermore, the integration of the dependency model creation approach into the service development process and the recomputation of the dependency model upon changes, are described.

Dependency Discovery

During the process of dependency discovery, the goal is to derive explicit resource, time, QoS, and price related dependency information from implicit dependency information contained in the composite service process description and the respective SLAs. The discovery process depends strongly on the specific properties of the different dependency types.

The discovery of time and resource dependencies consists of two phases. During the first phase the composite service workflow is decomposed into

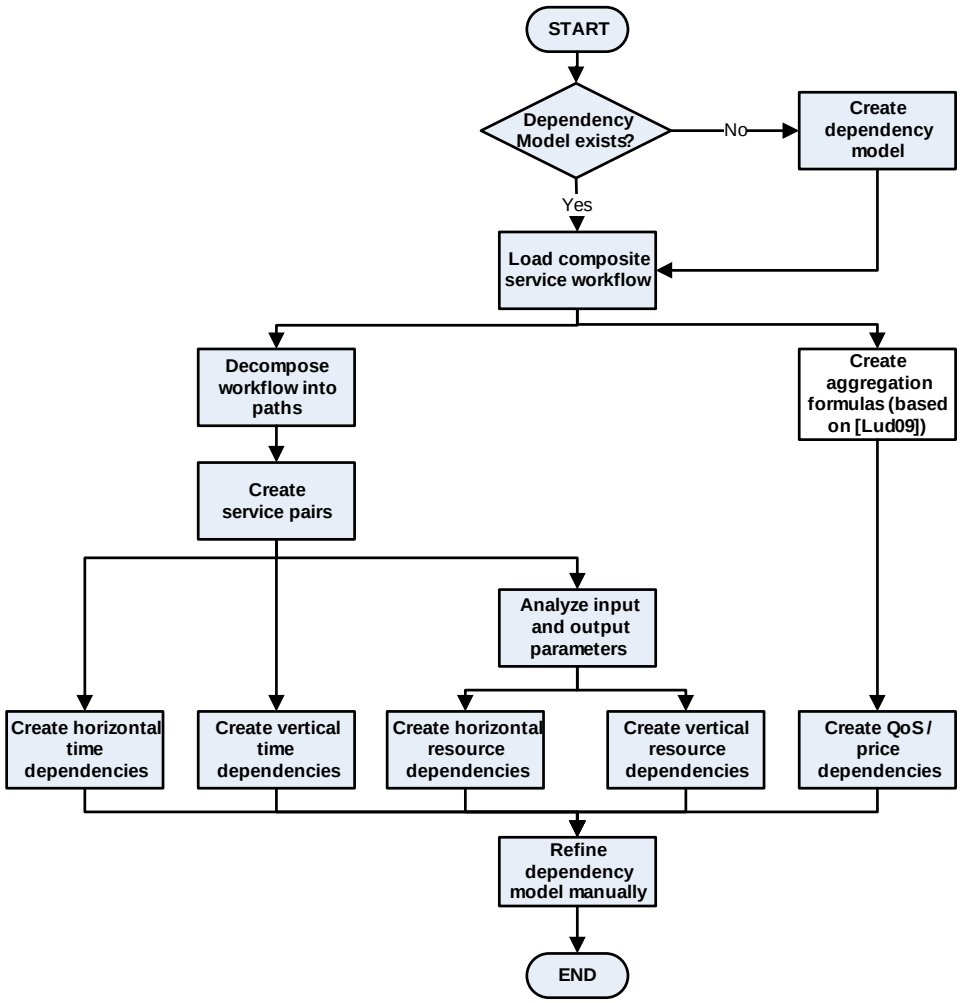

Figure 4.10: Creation and recomputation process for dependency models

linear paths. This phase is independent of the specific dependency. During the second phase the relevant services for dependency discovery are determined based on the created paths. Also the dependency discovery is executed and dependency information is added to the dependency model. Horizontal dependencies are found between different services within a path. Vertical dependencies are found between the composite service and the atomic services along the different paths. This phase is dependency-specific.

For QoS and price dependencies all atomic services are included in the dependency discovery. Instead of decomposing the workflow into paths, it is analyzed for specific workflow constructs such as *AND/OR/XOR* split or join. Based on the different constructs, formulas for the calculation of com-

posite QoS and price values are created. Initial work regarding the calculation of composite QoS values was done by Cardoso [CMSA04] and Jäger [JRGM04]. The approach of creating aggregation formulas was developed by Ludwig [LF08] as part of the COSMA approach and is integrated into the dependency management approach described in this thesis. In Section 3.3 more information regarding the three approaches is presented.

Path Creation As a preparation for the discovery of horizontal and vertical time and resource dependencies all linear paths, which lead from the start node to the end node of a process, are determined. Each element within a path has 0 or 1 predecessor and 0 or 1 successor.

Horizontal dependencies occur between two services, which are located on the same path. While e.g. time dependencies may exist between services on different paths, these dependencies cannot be discovered, but need to be modeled explicitly. This is due to the fact that the discovery approach for time dependencies is based on temporal information contained in workflow process descriptions (see Section 22). Resource dependencies exist due to provider-consumer relationships between services. Thus, they only appear when two services occur along a path.

Vertical dependencies occur between an atomic service from a path and the composite service under evaluation. Paths help to determine the right services for which vertical dependencies may occur. Since each path represents an ordered list of services within the process, the complexity for finding services is reduced.

Approaches to Path Creation: Different approaches exist in the area of workflows to analyze graphs with respect to finding certain paths. Many algorithms (e.g. Dijkstra's algorithm, Bellman-Ford algorithm [CLRS03], AO* [N.J80]) try to solve shortest path / lowest cost problems, i.e. finding the best path from one node to another node within the graph. However, the goal of the path creation step is finding all paths from a start node to an end node within a graph instead of finding one optimal path. Thus, these algorithms are not suitable to solve that problem. There are also other search algorithms, which traverse graphs following different strategies.

- **Breath first search:** The breath first search algorithm [CLRS03] is an uninformed search algorithm used to traverse through graphs. During the traversal process it always explores all neighbors of a node before continuing to one of the neighboring nodes and exploring its neighbors. Thus, the algorithm does not visit the nodes in the order in which they appear along a path, but instead continuously switches between the different paths. This way the path context of the nodes

(i.e. the services in a workflow) is lost. For this reason the algorithm is not applicable for the creation of the paths within a graph.

- **Depth first search:** The depth first search algorithm [CLRS03] is an uninformed search algorithm for graph traversal. During the traversal process, depth first search first explores the neighboring nodes of a node in an iterative approach until reaching an end node or a node that has been visited before. It then backtracks to the latest node (split node) with neighboring nodes, which have not been visited before, and continues to explore its neighbors. The algorithm visits nodes in the order in which they occur along a path in the graph. It finishes only after all nodes within a graph have been visited. It uses a node coloring approach to mark nodes which have not been seen (white), which have been discovered (grey), and which have been fully explored (black). Thus, it avoids exploring nodes and their neighbors twice.

None of the above mentioned algorithms is fully applicable to the problem space of this work. The *breath first search* algorithm does not provide the different paths of a graph but instead the nodes on the same level of depth inside the graph. The *depth first search* algorithm delivers linear paths. Some of these paths reach from the start node to the end node. At each split node several sub-paths are created reaching from the split node to an end node. Other subpaths may cover any linear segment within the graph. To create all full paths reaching from the start node to the end node, the algorithm would require some modifications. The created sub-paths would need to be merged so that full paths are created.

The Path Creation Algorithm: The algorithm for path creation (see Algorithm 1) is based on the depth first search algorithm. It traverses the workflow in a depth first manner. Each node, which is visited, is added to a path (*currentPath*). For each node its children are explored.

If a node has more than one child, these children are added to the node stack (*nodeStack*), which contains all nodes that have to be visited. Furthermore, it adds instances of the current element to the split node stack (*splitNodeStack*), which keeps all split nodes, i.e. nodes with more than 1 child node. If the current element (*currentElement*) has n children, it is added $n - 1$ times to the split node stack. If, for example, a node has two children, the node is added to the split node stack once, because after reaching the end node in the current traversal, it needs to come back to the last split node and continue traversing the graph for the second child. This way the algorithm keeps track of how many times the current path needs to be

Algorithm 1: createPaths

Input: Workflow $workflow$
Output: List $pathList$

```
1  WorkflowElement currentElement, splitNode = NIL
2  List currentPath, pathList, listOfChildElements = NIL
3  Stack splitNodeStack = NIL
4  Stack nodeStack = workflow.startNode
5  while nodeStack not empty do
6      currentElement = nodeStack.popFirstElement
7      currentPath.add(currentElement)
8      listOfChildElements = currentElement.childElements
9      if listOfChildElements not empty then
10         nodeStack.add(listOfChildElements.elems)
11         if listOfChildElements.size ≥ 2 then
12             for i = 1 to i = listOfChildElements.size − 1 do
13                 splitNodeStack.add(currentElement)
14             end
15         end
16     end
17     if listOfChildElements empty then
18         pathList.add(currentPath)
19         splitNode = splitNodeStack.popFirstElement
20         currentPath = currentPath.trim(splitNode)
21     end
22 end
```

trimmed up to the current split node before visiting the remaining children of the split node stored in the node stack.

When a node does not have any children, the end node is reached and the current path is added to the list of paths (*pathList*). Besides that, the path is trimmed to the last split node that was encountered during the traversal. This node is found on the split node stack.

The result of the path creation algorithm is a list of paths reaching from the start node to the end node. To illustrate the result of the algorithm, the complete list of created paths for the logistics and the healthcare use case are listed in Appendix C and D respectively.

Utilization of linear paths for the discovery of dependencies facilitates the analysis process because workflow constructs such as AND, OR, and XOR splits and joins are not treated separately. For the discovery of time dependencies information about the different workflow constructs is not relevant. Only information about the sequential order of activities is needed. How-

ever, not considering the different workflow constructs has some implications for the discovery of resource dependencies. During the dependency discovery process input and output resources are only matched for different services along a particular path. This can lead to situations where one service has a resource dependency regarding the same resource on two different services. This may be due to the fact that both services occur along different paths and only one path is executed during service provisioning. It may also be the case that one service occurs along two paths and the output of this service serves as input for the second service. If the two paths are created by an AND split and join an over-specification of resource dependencies will occur. However, this will not introduce false dependencies but only indirect dependencies.

An alternative approach would be the dependency analysis based on petri nets and the firing sequences between the start node and the end node of the petri net. This analysis would involve the following steps:

- Transformation of composite service workflow to a petri net (e.g. based on [DDO08, RPU+07, KDA06].
- Determination of all firing sequences covering the transitions between the start task and the end task
- Analysis of all tasks along each firing sequence for resource dependencies under the consideration of AND, OR, XOR split/joins

This approach is far more complicated than using simple paths for the analysis of dependencies.

Discovery of Horizontal Dependencies In order to discover dependencies between atomic services, the different paths created during the first step (see Section 4.4.2) are taken as the starting point. Each path is analyzed separately. Pairs of services are selected within each path and are analyzed for dependencies. The selection of services forming a pair is dependency-specific, i.e. different combinations of services are analyzed for time dependencies on the one hand, and resource dependencies on the other hand. A detailed explanation of the pair selection, as well as the dependency specific analysis procedure, is provided in the following paragraphs.

Horizontal Time Dependencies: Two atomic services $(s1_{atom}, s2_{atom})$, which are directly connected via an edge in a process, have a *finish-to-start time* dependency: $s1_{atom}.endTime$ finish-to-start $s2_{atom}.startTime$.

Algorithm 2 presents the creation of time dependencies. Time dependencies are created for any two neighboring services. Examples of horizontal

time dependencies for the logistics and the healthcare use case, which were discovered based on this algorithm, are presented in Tables 5.3 and 5.4 respectively.

Algorithm 2: analyzeHorizontalTimeDependencies

Input: List $pathList$

```
1  Service s1, s2 = NIL
2  foreach path ∈ pathList do
3      for i = 0 to i < path.size − 1 do
4          s1 = path.getElement(i)
5          s2 = path.getElement(i + 1)
6          addTimeDep(s1, s2, finish − to − start)
7      end
8  end
```

Algorithm 3: analyzeHorizontalResourceDependencies

Input: List $pathList$

```
1  Service s1, s2 = NIL
2  List resources, comResources = NIL
3  foreach path ∈ pathList do
4      for i = 0 to i < path.size − 1 do
5          s1 = path.getElement(i)
6          resources = s1.outResources
7          for j = i + 1 to j < path.size do
8              s2 = path.getElement(j)
9              comResources = match(resources, s2.inResources)
10             if comResources not empty then
11                 addResourceDep(s1, s2, comResources)
12                 resources.remove(match(resources, s2.outResources))
13             end
14             if resources empty then break
15         end
16     end
17 end
```

Horizontal Resource Dependencies: Two atomic services ($s1_{atom}, s2_{atom}$), which are directly or indirectly connected in a process, have a *resource* dependency, if a subset of the output resources of $s1_{atom}$ matches a subset of the input resources of $s2_{atom}$. In the case of matching resources between indirectly connected services, a dependency

only exists, if the input resources of $s2_{atom}$ are not provided by a different service $s3_{atom}$ located between $s1_{atom}$ and $s2_{atom}$: $s2_{atom}.inputResource$ resourceDependent $s1_{atom}.outputResource$.

Algorithm 3 presents the discovery of resource dependencies. Services from each path are analyzed. Output resources of earlier services are matched with input resources of later services. If common resources exist, a dependency is created. Resources, which are also provided as output by the latter service, are no more considered for further matching of the prior service. Examples of horizontal resource dependencies for the logistics and the healthcare use case, which were discovered based on Algorithm 3, are presented in Tables 5.3 and 5.4 respectively.

Discovery of Vertical Dependencies The discovery process for time and resource dependencies between the different atomic services and the composite service takes the paths created during the first step (see Section 4.4.2) as the starting point. From each path single atomic services are selected in order to be analyzed for dependencies with the composite service. As was the case for horizontal dependencies, the selection of services to be analyzed together with the composite service is dependency specific.

Vertical Time Dependencies: Each first atomic service in a path has a *start-to-start* time dependency on the composite service with $s_{comp}.startTime$ start-to-start $s_{atom}.startTime$. Each last atomic service in a path has a *finish-to-finish* time dependency on the composite service with $s_{atom}.endTime$ finish-to-finish $s_{comp}.endTime$. Algorithm 4 describes the creation of vertical time dependencies in pseudocode. Examples of vertical time dependencies for the logistics and the healthcare use case are presented in Tables 5.3 and 5.4 respectively.

Algorithm 4: analyzeVerticalTimeDependencies

Input: List $pathList$
Input: Service $compService$
1 **foreach** $path \in pathList$ **do**
2 Service $s1$ = $path.getFirstService$
3 Service $s2$ = $path.getLastService$
4 addTimeDep($compService$, $s1$, $start - to - start$)
5 addTimeDep($s2$, $compService$, $finish - to - finish$)
6 **end**

Vertical Resource Dependencies: An atomic service has a resource dependency on the composite service regarding its input resources, if (1) a subset of the composite service input resources matches a subset of the atomic service input resources and (2) the atomic service does not have a horizontal resource dependency regarding the matching resources: $s_{atom}.inputResource$ resourceDependent $s_{comp}.inputResource$.

The composite service has a resource dependency on an atomic service regarding its output resources if (1) a subset of the composite service output resources matches a subset of the atomic service output resources and (2) if the matching resources are not provided by another atomic service occurring after s_{atom} in the same path: $s_{comp}.outputResource$ resourceDependent $s_{atom}.outputResource$.

Algorithm 5: analyzeVerticalResourceDependencies

Input: List $pathList$
Input: Service $compService$

```
1  List resources, comResources = NIL
2  DependencyModel depModel = loadDependencyModel()
3  foreach path ∈ pathList do
4      resources = compService.inResources
5      foreach service ∈ path do
6          comResources = match(service.inResources, resources);
7          if comResources not empty then
8              if horizontalDep(service, comResources) ∉ depModel then
9                  addResourceDep(compService, service, comResources);
10             end
11             compResources.remove(matchResources(resources,
               service.outResources))
12         end
13         if resources empty then break
14     end
15     path = createReversePath(path)
16     resources = compService.outResources
17     foreach service ∈ reversePath do
18         comResources = match(service.outResources, resources)
19         if comResources not empty then
20             addResourceDep(service, compService, comResources);
21             resources.remove(comResources)
22         end
23         if resources empty then break
24     end
25 end
```

Algorithm 5 outlines the discovery of vertical resource dependencies in pseudocode notation. For each path the input resources of services are matched with the input resources of the composite service. If matching resources are found and no horizontal dependency exists regarding the same resources, a resource dependency is created. Resources, which are also provided as output of the atomic service, are not matched with later services. Once all input resource dependencies have been matched, the procedure is finished. Next, the path is reversed and each service is checked for output resource dependencies with the composite service. If matches are found, a resource dependency is created. The matched resources are not matched again with later services. Examples of vertical resource dependencies for the logistics and the healthcare use case, are presented in Tables 5.3 and 5.4 respectively.

Discovery of QoS and Price Dependencies In Section 3.3.3 different approaches to calculating composite service properties (e.g. QoS properties and price) were described. Cardoso et al. [CMSA04], Jaeger et al. [JRGM04], and Canfora et al. [CP04] developed approaches enabling the calculation of different composite QoS properties from atomic QoS properties using a graph reduction approach. Based on this work Ludwig [LF08] defined an approach for creating aggregation formulas for specific composite service properties. These aggregation formulas express the relationship (dependency) between a composite service QoS or price property and the respective atomic service properties, with regard to how the composite service QoS and price can be calculated from the atomic service values.

The usage of aggregation formulas, as defined by [LF08], enables the expression of vertical dependencies with respect to service QoS and price properties. For that reason it is used as one building block within the dependency management approach. The creation of aggregation formulas is part of the dependency discovery process as outlined in Figure 4.10. In a first step the workflow of a composite service is decomposed according to the composition patterns defined by Jaeger et al. [JRGM04]. Based on the respective patterns the aggregation formulas are created and stored as part of the dependency model.

Vertical Price Dependencies: A composite service s_{comp} has a *price dependency* on all atomic services $s1_{atom}$, .., sn_{atom} that have a price: $s_{comp}.price$ priceDependency $s1_{atom}.price$, .., $sn_{atom}.price$.

Vertical QoS Dependencies: A composite service s_{comp} has a *QoS dependency* on all atomic services $s1_{atom}$, .., sn_{atom} that have the same QoS

property. For the maximum temperature QoS attribute this looks as follows: $s_{comp}.maxTemperature$ qosDependency $s1_{atom}.maxTemperature$, .., $sn_{atom}.maxTemperature$.

An aggregation formula can be generated (for the price and QoS properties that were analyzed by Cardoso et al. [CMSA04], Jaeger et al. [JRGM04], and Canfora et al. [CP04]) and used to express dependencies of a composite service on a number of atomic services. The expression of the inverse dependency, i.e. how to calculate the QoS and price properties of atomic services based on composite service values, is not covered by the dependency discovery process. However, during the dependency modeling phase, such calculation formulas can be expressed based on the preferences of the composite service creator.

Due to the fact that the creation of aggregation formulas has been discussed by Ludwig [LF08], a more detailed discussion of this approach is not provided in this thesis.

Creation of the Dependency Model Information As part of the dependency analysis process new dependencies (see Section 4.3.2) are added to the dependency model. The process for adding a dependency is specific for each type of dependency. The differences are, however, only of minor nature, and are due to the different information represented by each dependency. As a first step the dependency model is loaded. Next, the services playing the role of dependent and antecedent are added to the model, if they are not already contained. They may be already contained due to the fact that each service can have different types of dependencies with different services. Finally, the dependency is added, if it is not already contained. Dependencies may already exist due to the fact that certain service pairs can occur in different service paths. Thus, some dependencies are discovered multiple times. The process for adding dependency information to the dependency model is presented in pseudocode notation in Algorithm 6 for the example of a time dependency.

Analyzing Subprocesses Service compositions do not only contain atomic services. The workflow representing the service composition may itself be structured using subprocesses. The structuring of the workflow is realized during its modeling. Atomic services are combined and included in subprocesses. Since these subprocesses represent a means for structuring the workflow and do not represent individual tradable services, no SLAs are negotiated for these subprocesses. Thus, subprocesses require special handling for the analysis of dependencies, i.e. for handling the introduced structural complexity.

Algorithm 6: addTimeDependency

Input: Service $dependentService$
Input: Service $antecedentService$
Input: String $timeOperator$

```
1  DependencyModel depModel = loadDependencyModel();
2  if dependentService ∉ depModel then
3  |  depModel.add(dependentService);
4  end
5  if antecedentService ∉ depModel then
6  |  depModel.add(antecedentService);
7  end
8  if dependency(dependentService, antecedentService, timeOperator)
   ∉ depModel then
9  |  depModel.createTimeDependency(dependentService,
   |  antecedentService, timeOperator);
10 end
```

The solution approach aims at resolving subprocesses and integrating respective subprocess activities into the composite service workflow. The process of integrating the subprocess activities into the process occurs before the creation of paths and ensures that the path creation algorithm is executed based on a flat workflow description (i.e. no contained subprocess). The approach assumes a representation of subprocesses according to the BPMN specification [Gro09]. A subprocess is represented by a special workflow element, e.g. the *Sub-Process* element of the BPMN specification. The subprocess consists of different services, which either form a workflow with a start and an end event, or which are grouped loosely without a start or end event and no sequence flow defined between the services. The latter case implies parallel execution of these services (compare [Gro09]).

To resolve the subprocesses of a workflow, the Sub-Process element is replaced by the services forming the subprocess and connecting them to the remaining workflow. This is done according to [NRA08], where the authors present an approach to redesigning processes. More specifically they show that parts of a SISO-net can be transformed to a different representation. A SISO-net is a generalized workflow-net with only a single input and a single output. Several transformations for redesign are presented including the *unfold transformation*, which replaces an aggregated task with the tasks it contains. In order to illustrate the replacement, two examples are shown here. A general description of the replacement approach is presented in [NRA08].

An example of a workflow containing a subprocess, which consists of loosely grouped services, is presented in Figure 4.11. It also shows the resulting workflow after resolving the subprocess. The following steps are taken:

1. Connecting all loosely grouped services within the subprocess with an *AND-Split* and *AND-Join*.
2. Connection of newly created *AND-Split* with incoming *Sequence-Edge* of *Sub-Process*.
3. Connection of newly created *AND-Join* with outgoing *Sequence-Edge* of *Sub-Process*.
4. Removal of *Sub-Process* element.

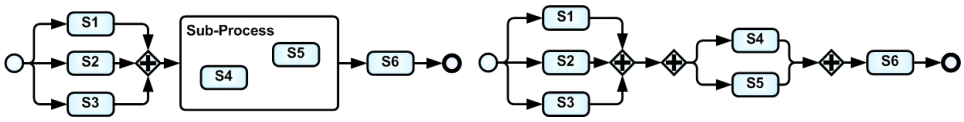

Figure 4.11: Resolving loosely grouped subprocess

Figure 4.12 shows an example of a subprocess containing a structured workflow and the resulting workflow after removing the subprocess element. If the subprocess consists of a structured workflow the following steps are taken in order to resolve the subprocess:

1. Removal of *Start-Event* and connection of incoming *Sequence-Edge* of *Sub-Process* to starting *Activity* or *Gateway*.
2. Removal of *End-Event* and connection of outgoing *Sequence-Edge* of *Sub-Process* to final *Activity* or *Gateway*.
3. Removal of *Sub-Process* element.

Figure 4.12: Resolving workflow subprocess

The steps for the removal of a subprocess are repeated for each subprocess found in a composite service workflow. There are no requirements with regard to the order of removing certain subprocesses prior to others. This is true for subprocesses at the same hierarchical level, as well as for subprocesses having a containment relationship.

Handling of Loops A loop within a workflow exists when an activity of the workflow is connected with an upstream activity within this workflow. An object is considered to be upstream if it "...has an outgoing Sequence Flow that leads to a series of other Sequence Flows, the last of which is an incoming Sequence Flow for the original object..." [Gro09]. The handling of loops with regard to dependencies is a challenging issue and is specific to the different dependencies.

Time dependencies express temporal relationships between the start and end times of two services. When loops occur, such temporal relationships cannot be expressed in an easy way. The downstream service would have to be executed after the first execution and before the second execution of the upstream service. This would require separate specifications of start and end times of each service execution, and thus separate SLAs. It would also require a predefined number of how many times a loop is executed. For services within loops without clear specification of how often they are executed, time dependencies cannot be described.

The handling of resource dependencies is a similar issue. For each loop iteration it is important to know exactly, which input and output resources of two services are matched.

The handling of loops as part of the determination of composite QoS and price attributes was partially solved by Cardoso et al. [CMSA04], Jäger et al. [JRGM04], and Canfora et al. [CP04]. All three approaches require additional information in order to solve the problem. The approach by Cardoso et al. requires the specification of the probability with which a loop is entered. However, it is difficult to determine the right number, unless sufficient historical data about a process is available. Jäger et al. as well as Canfora et al. require the specification of the number of loop iterations.

Loops introduce a level of complexity into workflows, which cannot be solved completely by this approach. Loops with a predefined number of executions can be represented equivalently by repeating the loop tasks in a linear fashion. At the time of SLA negotiation separate SLAs are negotiated for each task. The handling of such workflows is already supported by the approach. However, if loops do not specify the number of executions, but instead determine the execution of a loop based on runtime parameters, they cannot be handled by this approach.

Modeling of Non-detectable Service Dependencies

The dependency discovery algorithm produces a valid dependency model. It captures a number of different dependencies, which can be discovered based on the process description of the composite service, as well as the ne-

gotiated service level agreements. However, there are dependencies which cannot be discovered automatically. These dependencies have to be modeled explicitly. Information regarding such dependencies cannot be gained from the process description, but is available as domain knowledge of domain experts. To support the handling of these dependencies, a dependency model editor enables the manual extension and adaptation of the generated dependency model. The following list outlines optional modeling steps.

1. Modeling of complex time dependencies (e.g. *equals*, *overlaps*) between services in parallel flows.
2. Modeling of location dependencies (e.g. *equals*, *not_equals*)
3. Modeling of QoS dependencies for QoS properties, where no automatic detection is possible (compare [LF08, CMSA04]), but where the user wants to create a configuration. Also, the modeling of how to compute an atomic service QoS property from a composite service QoS property can be achieved.

Recomputation of the Dependency Model

The recomputation of a dependency model becomes necessary, when the process description of the service composition is changed. It may also become necessary upon SLA changes after renegotiation (e.g. changes of negotiated resources). In both cases the dependencies between services change and thus the model needs to be adapted. In case of renegotiating a service level agreement with regard to time, location, price, and QoS information, no changes in the dependency model are required.

The recomputation process repeats the steps of the dependency model creation process. More advanced approaches, e.g. the determination of changes and the execution of respective adaptations of the dependency model, are beyond the scope of this thesis.

Integration into the Service Lifecycle

In Section 2.1.2 the lifecycle of a service in the Internet of Services is described. The analysis of the dependencies between services is an important step in the *matchmaking phase*. It requires the composite service workflow description created during the *service offering phase*. Furthermore, it requires SLA offer documents for all atomic services and the composition. SLAs are negotiated during the *matchmaking phase*.

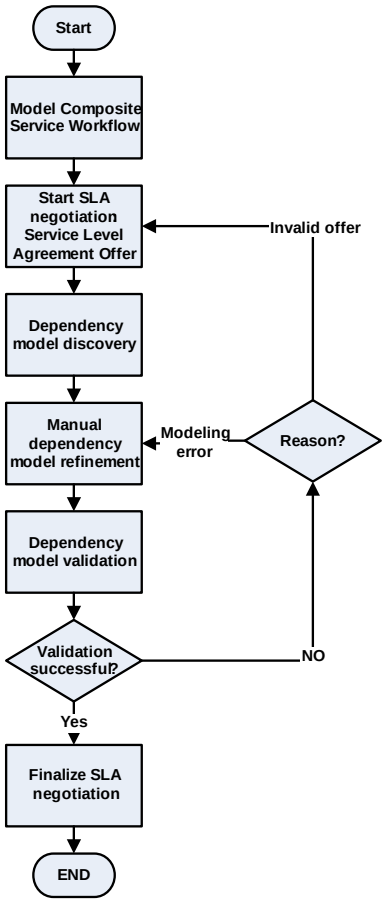

Figure 4.13: Dependency model creation and validation within process

Following the modeling of the composite service workflow and the creation of SLA offers for all services, the dependency analysis as well as the manual modeling of dependency information are executed. After that the dependency model is validated (see Section 4.4.3). If the validation of the model is successful, the negotiation of SLAs can be finalized and the model is available for further handling. If a validation error occurs, the user needs to investigate and fix the underlying cause. The integration of the dependency model creation and validation into the process of modeling the service and negotiating its SLAs is illustrated in Figure 4.13.

In the case that a dependency model needs to be recomputed, the process can also be executed using agreement documents instead of offer documents. If validation errors occur, the renegotiation of SLAs may be necessary.

4.4.3 Dependency Model Validation

After the creation of the dependency model it is necessary to validate the model based on the offered or negotiated SLAs. The validation is necessary to ensure that the dependency model and the respective SLAs are aligned. If this is not the case, i.e. there is a conflict between information in the dependency model and the SLAs, the underlying problem needs to be fixed before dependency handling and SLA negotiation can be continued. While the validation process is executed automatically, the handling of discovered problems is a manual process.

Validation Purpose

There are different errors, which can be found during the validation process. Some errors may occur due to false negotiation of SLAs. Other errors may occur due to false dependency modeling.

- **Time dependencies:** Time dependencies are discovered based on the workflow structure. Further time dependencies may have been modeled manually. Here it is necessary to verify that the different time dependencies are met by the negotiated start and end times in the referenced SLAs. Causes for errors include modeling errors of the composite service workflow or of time dependencies as well as falsely negotiated times.

- **Resource dependencies:** The discovery of resource dependencies is based on SLA descriptions of input and output resources in combination with the workflow structure. Thus, it is not possible to validate the SLAs based on the dependency model. However, it is necessary to check whether the resources provided by some services match the required resources of other services. According to [RM09] an important correctness issue of data flow in workflows is that if one activity ($A2$) needs to consume a data object ($d1$), another activity ($A1$) needs to provide $d1$ before $A2$ tries to consume the data object. This assumption is independent of the path taken. Typical mismatches include cases where a required resource fails to be provided, either by any other service in the workflow, or as the input of the composite service. Similarly, an output resource of the composite service may not have been provided by any other service or by the input of the composite service. Since the functionality of the composite service is realized by the atomic services, the composite service cannot create any output “by itself", i.e. besides the output created by the atomic services.

- **Location dependencies:** Location dependencies, which were modeled manually, need to be validated against the start and end locations negotiated in the respective SLAs. Errors include false dependency modeling as well as incorrectly negotiated SLAs.
- **QoS & price dependencies:** Aggregation formulas expressing QoS and price dependencies were discovered or modeled. These formulas are validated based on the respective SLO values of the different SLAs. Sources for errors include the false modeling of formulas or the false negotiation of SLO values.

Validation Approach

Algorithm 7 shows the validation algorithm using the example of time dependencies. Location, QoS, and price dependencies are implemented in a similar manner. However, location dependencies require a location operator while QoS as well as price dependencies require an aggregation formula. Furthermore, QoS and price dependencies involve more than one dependent service.

As a first step of the validation each service entity and dependency contained in the dependency model is checked if it contains all required data. As a second step each time, location, QoS, and price dependency is also validated with regard to its respective SLAs. SLA information is retrieved for the antecedent and all dependent services, and validated based on the time or location operator, or the calculation formula.

4.4.4 Dependency Model Usage

During service provisioning the provider of a composite service is responsible for managing the service composition. This includes two important tasks. When a SLO violation occurs, the effects on the atomic services and the composition need to be determined and handled. When a stakeholder of the composition tries to renegotiate its SLA, the effects of this renegotiation on other services need to be determined. Based on the outcome of the evaluation the renegotiation can be accepted or handled otherwise. Both tasks involve the determination of effects of an event on other services. In this section an approach is outlined, which describes how these effects can be determined using the dependency model. Figure 4.14 outlines the process.

Based on this evaluation further steps need to be determined. Possible steps for handling SLO violations include the raising of exceptions or the suggestion of actions, such as renegotiation of a SLA or the adaptation of

Algorithm 7: validateDependencyModel

Input: DependencyModel $depModel$
Output: Boolean $validationResult$

```
1  Service dependentService, antecedentService = NIL
2  SLA dependentSLA, antecedentSLA = NIL
3  Boolean result, validationResult = NIL
4  foreach serviceEntity ∈ depModel do
5  |  result = validateServiceEntityData(serviceEntity)
6  |  if result = FALSE then validationResult = FALSE;
7  end
8  foreach dependency ∈ depModel do
9  |  result = validateDependencyData(dependency)
10 |  if result = FALSE then validationResult = FALSE;
11 |  if dependency type TimeDependency then
12 |  |  antecedentService = dependency.getAntecedentService()
13 |  |  antecedentSLA = getSla(antecedentService.getSlaId)
14 |  |  dependentService = dependency.getDependentService()
15 |  |  dependentSLA = getSla(dependentService.getSlaId)
16 |  |  result = validateTimeDependency(antecedentSLA.timeValue,
   |  |  dependentSLA.timeValue, dependency.timeOperator)
17 |  |  if result = FALSE then validationResult = FALSE;
18 |  end
19 end
```

the overall process. Renegotiation requests can be accepted, rejected, or may need further considerations with regard to other SLAs. The determination of next steps may be realized by a special component or involve human activities. This is, however, not part of this thesis.

Evaluating SLA Renegotiation Requests

SLA renegotiation requests need to be evaluated whenever a provider of an atomic service or a consumer of a composite service wants to renegotiate one or more service level objectives of his contract. Algorithm 8 outlines the necessary steps for analyzing renegotiation information in pseudocode notation.

Based on the service key all relevant dependency models are retrieved, which involve the current service. For each dependency model the relevant dependencies are extracted. A dependency is relevant if two aspects are met: it expresses a dependency matching the KPI (1) and the current service has the antecedent role, or the dependency is a bidirectional depen-

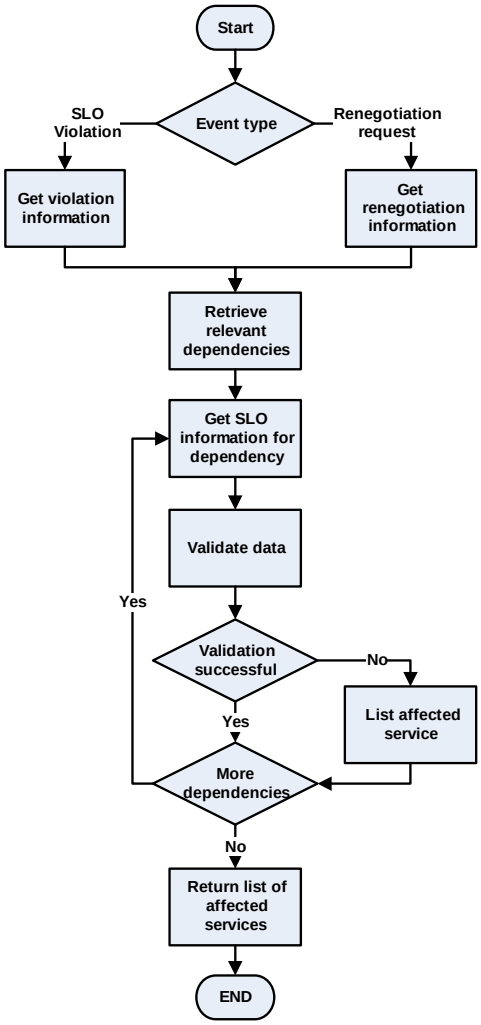

Figure 4.14: Dependency evaluation during service provisioning

dency (2). For each dependency the relevant service is extracted and its KPI information determined. Finally, the KPI information of both services is validated based on the dependency description. If a conflict is detected the respective service is added to a list of affected services, which is later displayed to the user. Furthermore, if the dependency is of transitive nature, further affected services are determined for the dependent service.

Evaluating SLO Violations

Service dependency evaluation is performed whenever SLO violations occur. SLO violations are detected during service monitoring. Information

Algorithm 8: getAffectedServices

Input: String $serviceKey$, String $kpiName$, KpiValue $kpiValue_1$
Output: List $affectedServices$

```
1  KpiValue kpiValue_2 = NIL
2  List dependencyModels = getModels(serviceKey)
3  List dependencyList = NIL
4  DependencyDescription depDescription = NIL
5  foreach model ∈ dependencyModels do
6      dependencyList = getDependenciesForServiceKPI(serviceKey,
       kpiName)
7      foreach dependency ∈ dependencyList do
8          kpiValue_2 = retrieveKPIValue(dependency.dependentService,
           kpiName)
9          depDescription = dependency.description
10         if validate(kpiValue_1, kpiValue_2, depDescription) ≠ true then
11             affectedServices.add(dependency.dependentService)
12             if dependency.isTransitive then
               getAffectedServices(dependency.dependentService.serviceKey,
               kpiName, kpiValue_2)
13         end
14     end
15 end
```

about SLO violations, i.e. SLO violation messages, are analyzed to determine affected services.

The functionality for determining affected services is very similar to the functionality for evaluating renegotiation requests. The difference is that bidirectional dependencies with services occurring earlier in the composition are not considered for evaluation.

4.4.5 Dependency Model Retirement

During the retirement phase the dependency model is taken out of service. This happens at the end of the lifecycle of the composite service for which it was created. At that time the SLA for the composite service has ended. The service will not be executed anymore. Thus, neither requests for renegotiation, nor SLO violation events are received. In case the customer wants to consume the same service again, a new SLA needs to be negotiated and a new dependency model will be created. While the dependency model is not needed any more for the management of service dependencies, it may be kept for later reference or documentation reasons.

4.5 A Platform for Service Dependency Management

The management of service dependencies is achieved by means of algorithms for dependency analysis at design time, dependency representation in a suitable model, and algorithms for dependency evaluation at runtime. This section describes the integration of the presented approach into a conceptual architecture supporting the lifecycle of services (see Section 2.1.2).

As a first step an overview of the tools and the platform for designing, trading, and providing services is presented. The tools and platform were developed jointly with different partners within the TEXO project. While the description presents the overall functionality of the tools and the platform, the focus is on the components for managing service level agreements. They provide the functionality for the SLA lifecycle presented earlier (see Section 2.2.2). As a next step a detailed description of the components for dependency management and their integration into the conceptual architecture is presented. These components implement the concepts described earlier in this chapter. Finally, the interplay of the different components to realize dependency management is illustrated.

4.5.1 An Architecture for Service and SLA Management

When services are traded on service marketplaces, different providers can offer their services to different consumers. While service development and provisioning is handled by service providers, this is not necessarily the case for tasks such as negotiating and monitoring SLAs and handling the billing for service usage. The architecture presented in this chapter and developed as part of the Theseus TEXO project assumes that a central marketplace offers many of these important functionalities.

The architecture supports the development and execution of services, the management of SLAs according to the SLA lifecycle described in Section 2.2.2, and the monitoring of services based on SLAs. Furthermore, a number of other important tasks, such as billing service usage and enforcing access policies, are handled by the architecture, but do not represent the main focus of this thesis.

The core structure of the architecture is composed of three building blocks: the design time tools (*ISE Development Environment*), a service marketplace (*Service Management Platform*), and a service runtime infrastructure (*Tradable Service Runtime*). Figure 4.15 presents an overview of the main architectural components (FMC notation[1]). Components for the handling of service de-

[1]Fundamental Modeling Concepts: http://www.fmc-modeling.org/

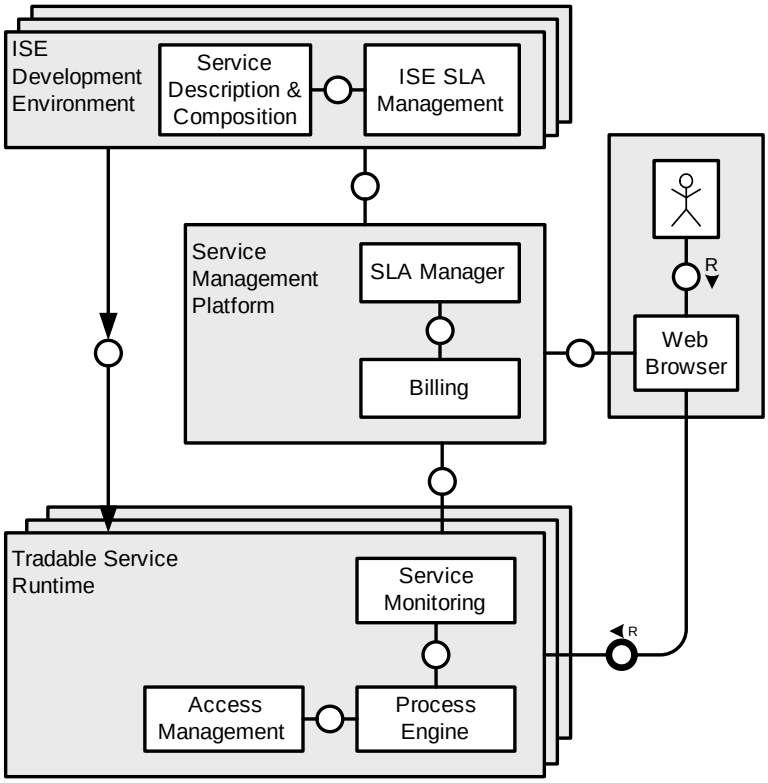

Figure 4.15: Abstract architecture for service infrastructure

pendencies are integrated into the architecture and will be described later in this chapter.

The ISE Development Environment consists of design time service engineering tools for designing and describing services as well as for creating service compositions. It also offers support for composite service providers to create and negotiate SLAs. The Service Management Platform (SMP) provides service marketplace functionality for offering and searching services, negotiating and monitoring SLAs, and billing, only to mention a few. The Tradable Services Runtime (TSR) supports service execution and monitoring at the provider's side. In a typical scenario there are multiple distributed service runtimes, which are interacting with the central service marketplace. The communication between the marketplace and the different service runtimes is realized via a message-oriented middleware, which supports the exchange of information regarding deployed services, newly negotiated SLAs, and monitoring events.

In the following sections an overview of the technical infrastructure of the design time service engineering tools, the service marketplace, and the service runtime is given. The SLA management and monitoring architecture

is part of this infrastructure. The main focus of this description is on the SLA management and monitoring aspects. Besides that, related functionality, such as service deployment and execution, as well as billing service usage, is only briefly presented to provide a more general understanding.

ISE Development Environment

The ISE Development Environment is a set of tools supporting the work of service providers to engineer services (see Figure 4.20).

Service Description and Composition The design time tools support the design of services and their description based on USDL (see Section 2.1.3). They also enable the creation of service compositions and their description as composite services. The tools realize the Integrated Enterprise Service Engineering (ISE) methodology developed as part of the TEXO project [CVW08]. The ISE methodology describes a model-based approach to service engineering. It separates different aspects of a service (e.g. its underlying workflow or its description, which is visible to marketplace users) and enables their modeling from a business as well as a technical perspective.

The service description provides a basis for searching and finding services as well as for formalizing contracts (i.e. SLAs) between service providers and consumers.

Service compositions are created from services, which are offered via the service marketplace or which are available in a local service repository of the composite service creator. As part of the creation process, service level agreements need to be negotiated between the composite service creator and the providers of the atomic services.

The ISE Development Environment supports the creation of service compositions based on the BPEL specification [JEA+07]. BPEL processes can be directly modeled using a BPEL editor. An alternative approach is the specification of a business process using the BPMN notation [Gro09], followed by a transformation to BPEL. This approach requires some additional modeling steps after the transformation.

The SLA Template Generation Component Service level agreements generally contain information such as a description of the service to be provided, the price of the service, and a number of guaranteed quality attributes. Much of this information is also available within the service description. Thus, service description information can be used as a basic building block for the formalization of service level agreements.

The negotiation of SLAs, which was implemented based on the WS-Agreement specification [ACD+07], requires a SLA template as its starting point. The template contains the relevant information about a service, which is needed for the negotiation process.

The SLA template is generated from the service description. The approach consists of two parts. On the one hand service description elements are utilized to formalize parts of the SLA template. USDL elements are included into the SLA template. On the other hand information regarding measurable attributes are mapped to the description of SLOs. This integration of the service description and the SLA template creation process helps to automatize parts of the overall service development process. The final agreement document contains language constructs, which are specific to the agreement notation (i.e. WS-Agreement) as well as the service description language (e.g. USDL). Listing 4.4 shows a simplified sample of a WS-Agreement document including USDL elements in the ServiceDescriptionTerm section. The name and metric of the variable in the ServiceProperties section also originate from the service description document. This information is mapped to elements in the agreement notation. The same is true for the SLO target value in the GuaranteeTerm section. The generated SLA template is then deployed together with the newly created service.

```
1 agreement {
2   Name Truck DD SLA
3   ServiceDescriptionTerm {
4     ServiceName Truck DD
5         usdl:providerName Dresden Logistics
6     usdl:description Pick–up and transport of goods
          within city area
7         usdl:price 170 EUR
8   }
9   ServiceProperties {
10    VariableSet {
11      Variable {
12        Name executionTime
13        Metric xsd:duration
14  }}}
15  GuaranteeTerm {
16    Name BasicService_GUARANTEE
17    ServiceScope Truck DD
18    ServiceLevelObjective {
19      KPIName executionTime
20      Target 2H
21  }}}
```

Listing 4.4: WS-Agreement SLA with USDL code

ISE SLA Wizard The *ISE SLA Wizard* supports composite service providers to create service composition. It provides the functionality to search for a SLA template based on a service key and to negotiate a SLA for the respective service. The wizard is integrated into the ISE Development Environment to enable better integration with the modeling work rather than having separate tools.

Service Management Platform

The Service Management Platform (SMP) enables service providers to make their services available to interested consumers, who can use the SMP to search for suitable services according to their needs. It provides the marketplace functionality for trading services and thus represents the central infrastructure for service providers and consumers to engage in business interactions.

In contrast to the service development tools, which are under the control of the different service providers, the SMP is a centralized component under the control of a marketplace provider, who makes the SMP functionality available for service providers and consumers.

Service Registration, Search, and Billing In order to make a service available on a marketplace, it needs to be registered. Service registration is one step of the service deployment process. During the registration process important information about the service is made available for different components of the marketplace. Different search engines are provided with information about the service functionality, promised quality of service parameters, and pricing information. Once the registration of the service is completed, it can be found and used by consumers. Different search engines support consumers in finding the right service for their needs.

Furthermore, functionality for billing and pricing is provided. This way the marketplace offers key functionality, without which business interactions would be difficult to establish.

SLA Management The provisioning and consumption of services is regulated by service level agreements (SLAs). The *SLA Manager* component provides functionality supporting the handling of service level agreements at the service marketplace (see Figure 4.20). This includes the deployment of new SLA templates, the negotiation of SLAs between service consumers and providers, the monitoring of negotiated contracts during service provisioning, and the renegotiation of contracts by service consumers and providers during the lifetime of a SLA.

SLA Manager Architecture and Context Figure 4.16 shows the main components of the SLA Manager. The *SLA Deployment* component supports the deployment of new SLA templates during the service deployment process as well as their removal, when a service should not be offered any more. These templates are received from the deployment component, which manages the process of deploying a new service and registering it at the marketplace. The received SLA template is stored in the *Template Repository*.

The *SLA Negotiation* component supports the SLA negotiation process, which can be triggered via a web interface by an end user or from the ISE SLA Wizard by a service engineer. Following the successful negotiation or renegotiation of a SLA, an event is sent to the message-oriented middleware informing the monitoring components about it.

The *SLA Monitoring* component receives events about detected SLO violations from the service runtime. This information is stored and made available for the billing component.

Furthermore, the SLA Manager makes contract information (e.g. consumer information) available for the billing component.

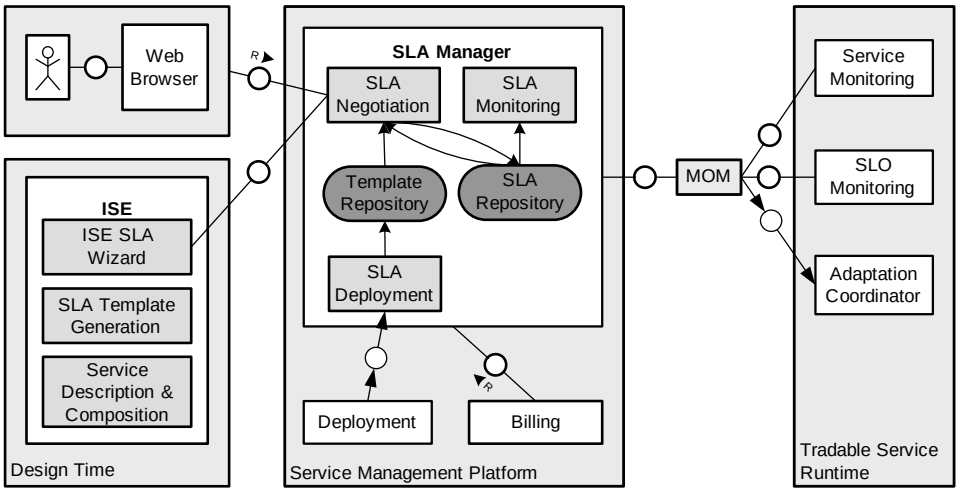

Figure 4.16: SLA Manager and interfaces to other components

SLA Template Deployment The SLA Manager provides interfaces for the deployment, update, and removal of SLA templates. The deployment of SLA templates occurs as part of the service deployment process. SLA templates, which form the basis for SLA negotiation, are created as part of the service engineering process. They are generated from the service descrip-

tion. When a SLA template is deployed it is stored in a template repository where it is available for the SLA negotiation.

During the lifetime of a service the SLA template of the service can be updated. This makes it possible to adapt to consumer requirements, e.g. if a service cannot be sold due to its current price model. It also enables the provider to change the contracting terms for the service if the capabilities for service provisioning change.

The removal of templates is necessary when a service ceases to be offered.

SLA Negotiation The SLA Manager supports the negotiation of service level agreements according to the WS-Agreement specification. It acts on behalf of the service provider. It provides two interfaces supporting the negotiation process presented in Listing 4.5 and 4.6.

```
1 public Template getTemplate(String serviceKey)
2 {
3 }
```

Listing 4.5: The interface getTemplate

```
1 public Agreement createAgreement(Offer offer)
2 {
3 }
```

Listing 4.6: The interface createAgreement

A consumer can request a SLA template for a service and refine it according to his needs, thus creating an agreement offer. Once all refinements are made, the agreement offer can be submitted to the SLA Manager in the form of a request for creating an agreement based on this agreement offer. If the agreement offer is successfully evaluated it is accepted, stored in the *SLA Repository*, and the consumer is informed. The negotiation procedure is presented in Figure 4.17. Furthermore, an event is sent to the message-oriented middleware informing interested components (e.g. different monitoring components) of the service provider about the new SLA. These components are then able to request the new agreement and use its information to prepare for service execution and monitoring.

A further functionality provided by the SLA Negotiation component is the renegotiation of SLAs. It allows to adapt single SLOs of a SLA if the service provider and the consumer agree. Renegotiation is only possible for SLOs which were marked as *renegotiatable* during the SLA negotiation procedure.

SLA Renegotiation The renegotiation of SLAs enables consumers and providers of services to adapt a SLA according to their needs even after

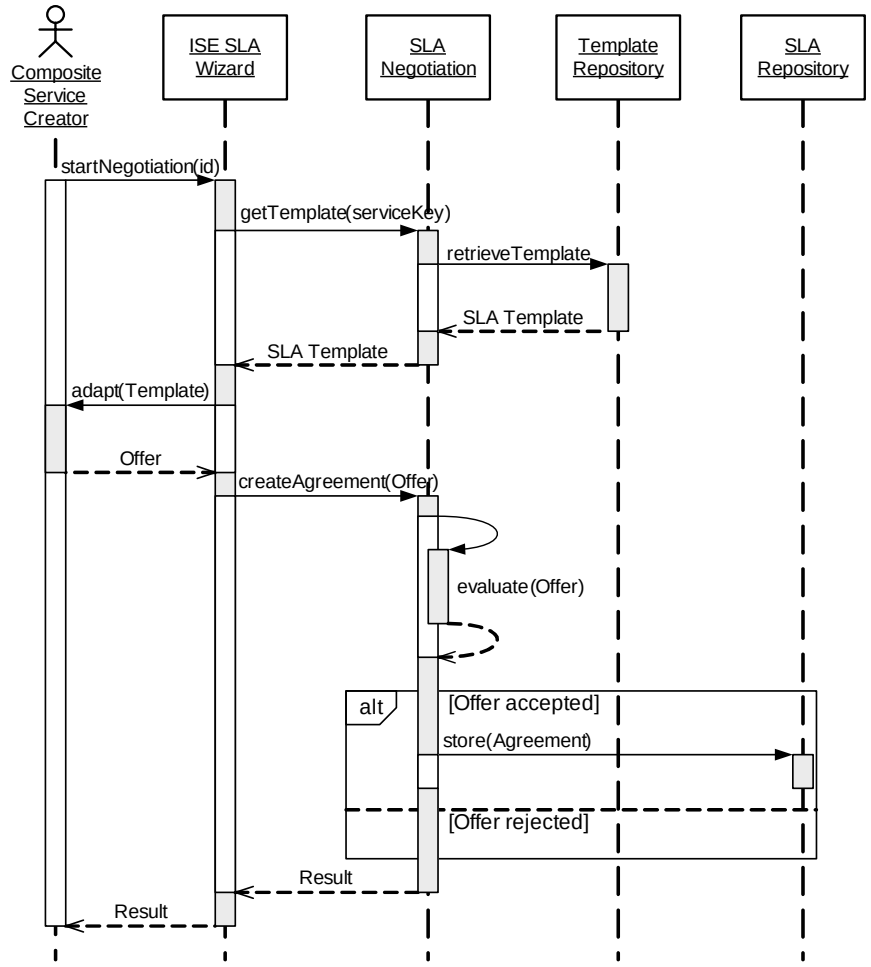

Figure 4.17: SLA negotiation procedure

the SLA has been agreed upon by both parties. This functionality is useful, because it allows the parties to adapt their SLAs to changing conditions (see Section 2.2.2 for more details).

The SLA Manager is the central component, which enables the renegotiation of SLAs. Requests for renegotiating a SLA need to be sent to the SLA Manager, which can either automatically evaluate the request or forward it to the respective service provider in case that some more advanced decision making process is involved (e.g. human involvement in the decision process is necessary). In Figure 4.18 the process for renegotiating a SLA is shown from the perspective of a composite service creator, who uses the ISE SLA Wizard for renegotiation. As a first step of the renegotiation process the agreement is requested by the SLA Manager. With the help of the ISE SLA Wizard the user can change the respective values of the SLA and

submit it to the SLA Manager for further handling. In case that the SLA Manager can automatically evaluate the renegotiation request, the evaluation is performed and a result is returned. Otherwise it forwards the renegotiation request to the responsible service provider for further handling. The renegotiation result is returned to the SLA Manager. If the proposed offer is accepted (either by the SLA Manager or by the service provider) it is stored in the *SLA Repository*. Finally, the result is sent to the requesting party.

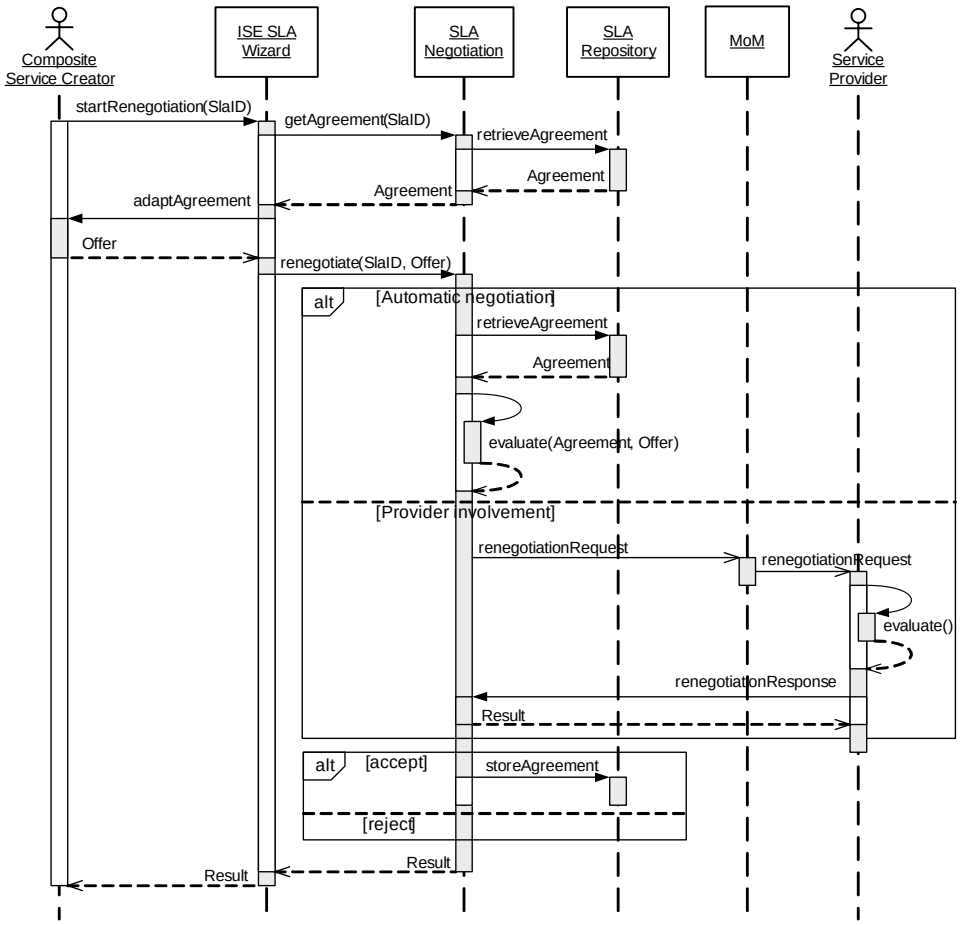

Figure 4.18: SLA renegotiation process

SLA Monitoring Monitoring information regarding the service execution is collected during runtime. The evaluation of the single measurements is not realized by the SLA Manager itself but instead by the *SLO Monitoring* component, which is a part of each service runtime infrastructure. The handling of all monitoring events by the SLA Manager would create a bot-

tleneck. The SLA Manager receives information regarding detected SLO violations. This information is stored and evaluated when other components or a service client request the status of a SLA.

```
1 public List<SLOMeasurementStatus> getSLAViolationDetails(
2                                   String serviceInstanceID) {
3         }
```

Listing 4.7: The interface getSLAViolationDetails

SLA status information is particularly important for the process of billing a service consumer for the usage of a service. The billing component uses the status information to determine the final price to be charged to the consumer for using the service. If the negotiated SLOs were violated, the consumer might be entitled to a discount. Listing 4.7 presents a function for determining the SLO violations of a SLA.

Tradable Service Runtime

The Tradable Service Runtime (TSR) provides support for executing services and their underlying processes as well as for monitoring the execution. A TSR may be under the control of a single service provider or managed by an external infrastructure provider. In the latter case multiple service providers may make use of the hosting service and share the same runtime infrastructure. Each service provider uses the functionality of a service runtime to support service execution. In a typical scenario there are multiple distributed service runtimes for the different service providers, which are interacting with the central service marketplace. Figure 4.20 outlines the most important components for service execution and monitoring.

Service execution and monitoring One major task of the TSR is to support the execution of services. The execution of services and processes is supported by respective engines, i.e. web service and BPEL engines. Before a service call from a user is executed, a check is performed whether or not consumers trying to access a service are entitled to do so. This is only the case if a SLA was negotiated, which allows access to the service. During service execution monitoring sensors measure a variety of different parameters and make this information available for further evaluation.

SLO Monitoring Component The *SLO Monitoring* component is responsible for evaluating monitoring data during service provisioning. It requests the negotiated SLAs from the SLA Manager upon receiving a notification that a new SLA has been negotiated.

The monitoring information is made available by monitoring sensors via a message-oriented middleware. When the SLO Monitoring component receives monitoring information, it evaluates it based on information in the negotiated SLA. As a first step the monitoring data is analyzed to find out under which SLA the service was executed. Based on the type of measurement data received, it retrieves the SLO information from the SLA and analyzes this information. If a SLO violation is discovered, an event is sent to the message-oriented middleware in order to inform other components about the occurrence of the problem. The process of handling monitoring data is illustrated in Figure 4.19.

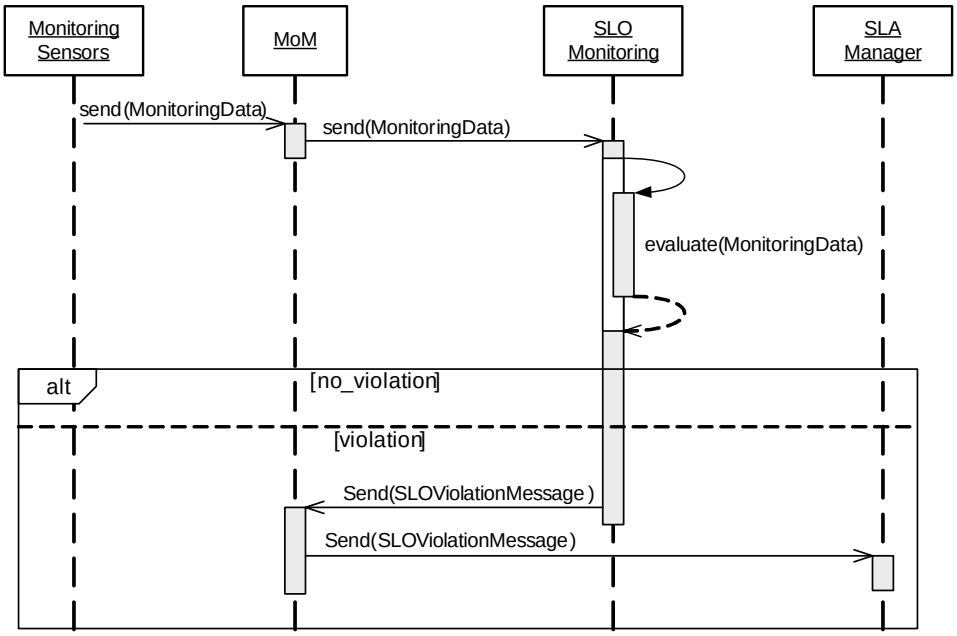

Figure 4.19: Evaluation of monitoring data

Integration of the Service Infrastructure Components

The service design time tools, marketplace, and runtime infrastructure provide their functionality in a distributed fashion. Service providers use the design time tools under their control to engineer their services. The service runtime infrastructure may be under the control of either the single service providers or a hosting company. The marketplace is a central infrastructure controlled by the marketplace provider.

The integration of the different components is necessary to enable the tasks of the different infrastructure components. It is realized in two different ways. Web service interfaces enable the direct communication between two

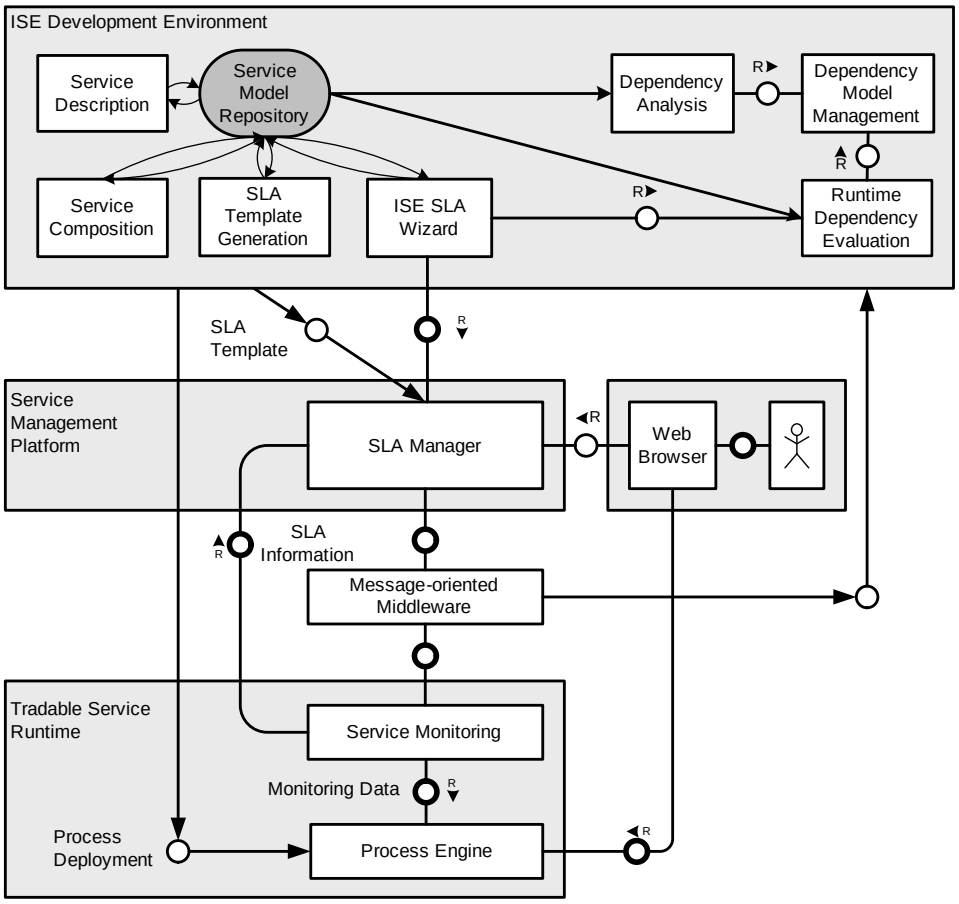

Figure 4.20: Architecture overview of service infrastructure components

components. A message-oriented middleware allows for communication between multiple components following the publish-subscribe pattern. It provides good scalability and flexibility with respect to who is interested in which information.

The deployment and registration of services from the development environment to a TSR and the marketplace, as well as the distribution of the service related artifacts (e.g. service description, process description), is realized by means of web service interfaces. The communication between the marketplace and the different service runtimes is realized via a message-oriented middleware, which supports the exchange of information regarding deployed services, newly negotiated SLAs, and monitoring information. The communication between the different components of the service runtime (e.g. to exchange monitoring information) is also realized via a message-oriented middleware. In Figure 4.20 the most important inter-

faces between the different components are illustrated. The interplay of the different components is described in the following section.

Exemplary Description of Component Interplay

At design time a composite service provider models a service composition using the *Service Composition* tools. For the different services that are integrated into the composition, SLAs are negotiated using the ISE SLA Wizard. It communicates with the SLA Manager via a web service interface to request a SLA template, submit the SLA offer, and receive the final agreement. During development the service is also described using the *Service Description* tools. Finally, a SLA template is generated for the service by the *SLA Template Generation* component. All service related artifacts are stored in the *Service Model Repository*.

Following the development, the service is deployed to make it available for consumers. The SLA template is deployed to the SLA Manager in order to serve as a basis for SLA negotiation. The process description is deployed to the *Process Engine*. The service can now be found by potential customers. Before using the service they need to negotiate a SLA for it using either a web browser or the ISE SLA Wizard, depending on whether they are end users or composite service providers. Once a new SLA is negotiated, the SLA Manager informs interested components such as *Service Monitoring* about this event using the message-oriented middleware. These components can then retrieve the new SLA via a web service call. The user can now utilize the service. Monitoring information about service provisioning is captured and evaluated with regard to the negotiated SLA. If the Service Monitoring components detect a violation, this information is forwarded to interested components via the message-oriented middleware.

Service consumers are able to requests the renegotiation of a SLA using the ISE SLA Wizard or the browser. If the SLA Manager cannot resolve the request by itself, it forwards the information to the respective service provider via the message-oriented middleware. From there it is available for further handling, e.g. by the dependency management components, which are described in the next sections.

4.5.2 Dependency Management Extensions

In this section the necessary extensions to enable the management of dependencies are presented. The extensions include components for the creation, validation, and storage of dependency models (*Dependency Model Management*), the discovery and modeling of dependencies (*Dependency Analysis*), and the evaluation of the dependency model with respect to dif-

ferent events at runtime (*Runtime Dependency Evaluation*). The respective components are presented in the architecture overview in Figure 4.21. They are described in the following sections. Furthermore the interaction between the different components as well as their integration into the overall design- and runtime architecture is explained.

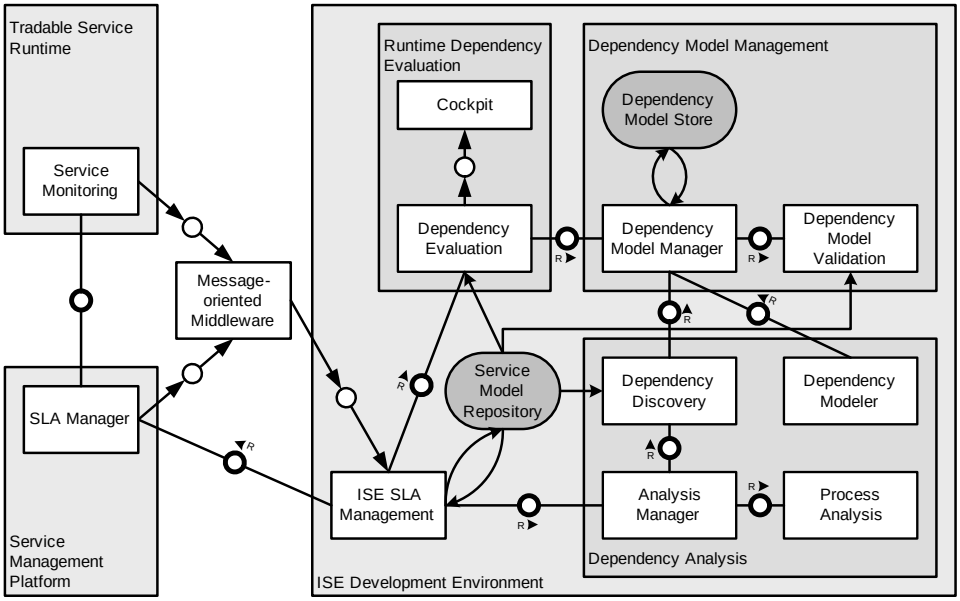

Figure 4.21: Conceptual architecture of dependency management

Dependency Model Management

The approach for managing service dependencies has at its core the dependency model, which is used to capture information about services and the dependencies that occur between them. The components, which are part of the dependency model management, are responsible for the creation, validation, and storage of dependency models and for making the respective models available to other components.

The *Dependency Model Manager* is the central component. It creates new dependency model instances for each new SLA negotiated for a composite service. The models are stored in the *Dependency Model Store*. It is also responsible for adding information to dependency models and making model information available to other components such as *Dependency Modeler* and *Dependency Evaluation*. The functionality for the modification of dependency models, as well as making model information available to other components, was assigned to a single component in order to avoid conflicting access to single dependency model instances. The *Dependency*

Model Validation component is responsible for validating the dependency model with respect to the defined constraints and the respective SLAs and SLOs. The dependency model instances are stored in the *Dependency Model Store*. From there they can be retrieved by the Dependency Model Manager and made available for runtime evaluation. Figure 4.22 shows the steps involved in the dependency model validation process.

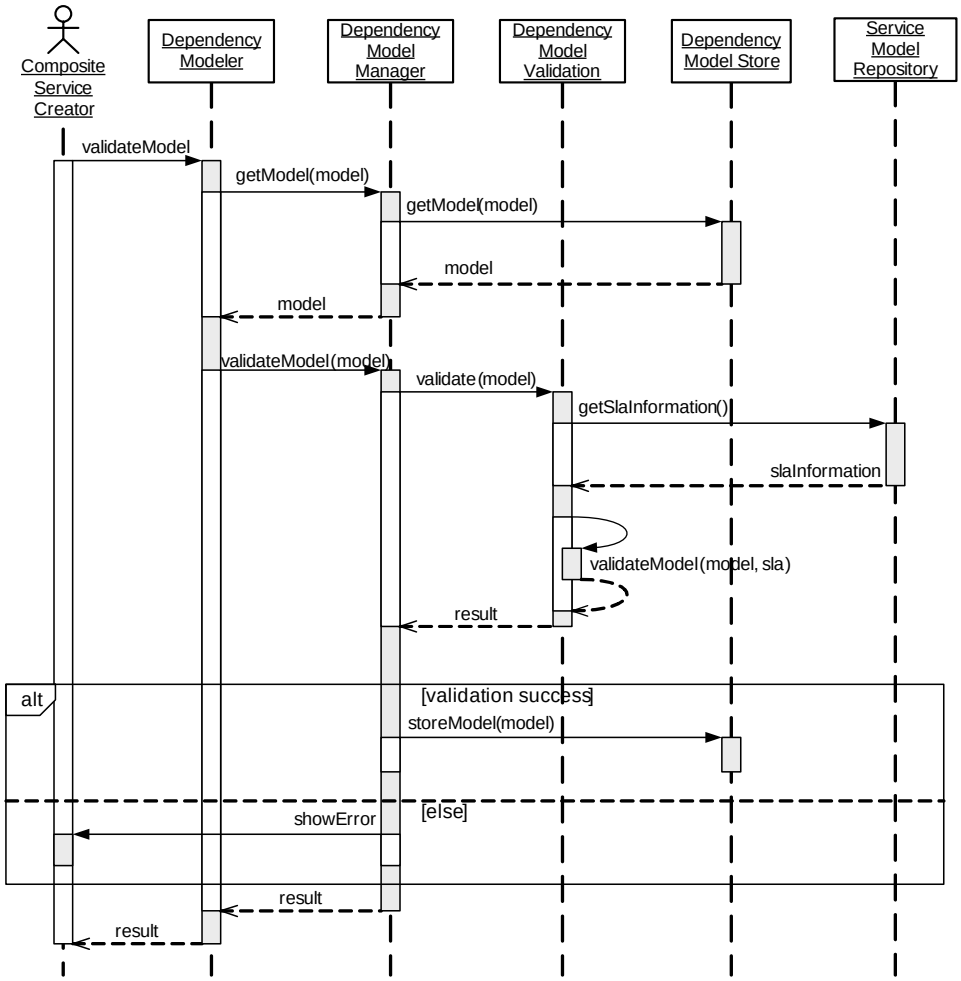

Figure 4.22: Validation of dependency model and SLAs

When the composite service creator wants to validate the dependency model using the Dependency Modeler tool, the dependency model is loaded from the Dependency Model Store via the Dependency Model Manager. As a next step the Dependency Model Manager is requested to validate the model. Thus, it orders the Dependency Model Validation component to validate the dependency model. To be able to do that, it requests the necessary SLA information for the different services from the Service

Model Repository. Based on the dependency model and SLA information the validation is executed and the result is returned to the Dependency Model Manager. In case the validation was successful, the dependency model is stored in the Dependency Model Store. In case of a problem occurring during the validation, the composite service creator is informed.

Service Dependency Analysis

The functionality for dependency analysis is distributed between components, which support the automatic dependency discovery as well as dependency modeling (see Section 4.4.2). The *Analysis Manager* manages the process of automatic dependency discovery. It initiates the creation of a dependency model, and invokes the functionality for creating the paths of the process and discovering the dependencies. The *Process Analysis* component is responsible for analyzing the underlying process of a composite service and preparing it for the dependency analysis. It decomposes the process into linear paths leading from the start node to the end node. These paths are used for the analysis of dependencies. The *Dependency Discovery* component realizes the different dependency discovery mechanisms of the dependency analysis process. The Dependency Modeler enables the modeling of dependency information, which cannot be discovered automatically.

The dependency analysis process, i.e. the dependency discovery, and the involved components are presented in Figure 4.23. The discovery process is initiated by the composite service creator during the process of negotiating the SLAs with the consumer of the composite service as well as the atomic service providers. Listing 4.8 presents the interface via which the dependency analysis is initiated.

```
1 public void analyzeDependencies(File process)
2 {
3 }
```

Listing 4.8: The interface analyzeDependencies

The analysis process is managed by the Analysis Manager component. It creates a dependency model and initiates the decomposition of the process underlying the composite service. This step is executed by the Process Analysis component. It decomposes the process and retrieves all linear paths of services leading from the start node to the end node of the process (see Section 4.4.2). Based on the path the discovery of dependencies is executed by the Dependency Discovery component. As part of the discovery process it retrieves SLA information for the different services. When it discovers a dependency it requests the *Dependency Model Manager* to add the

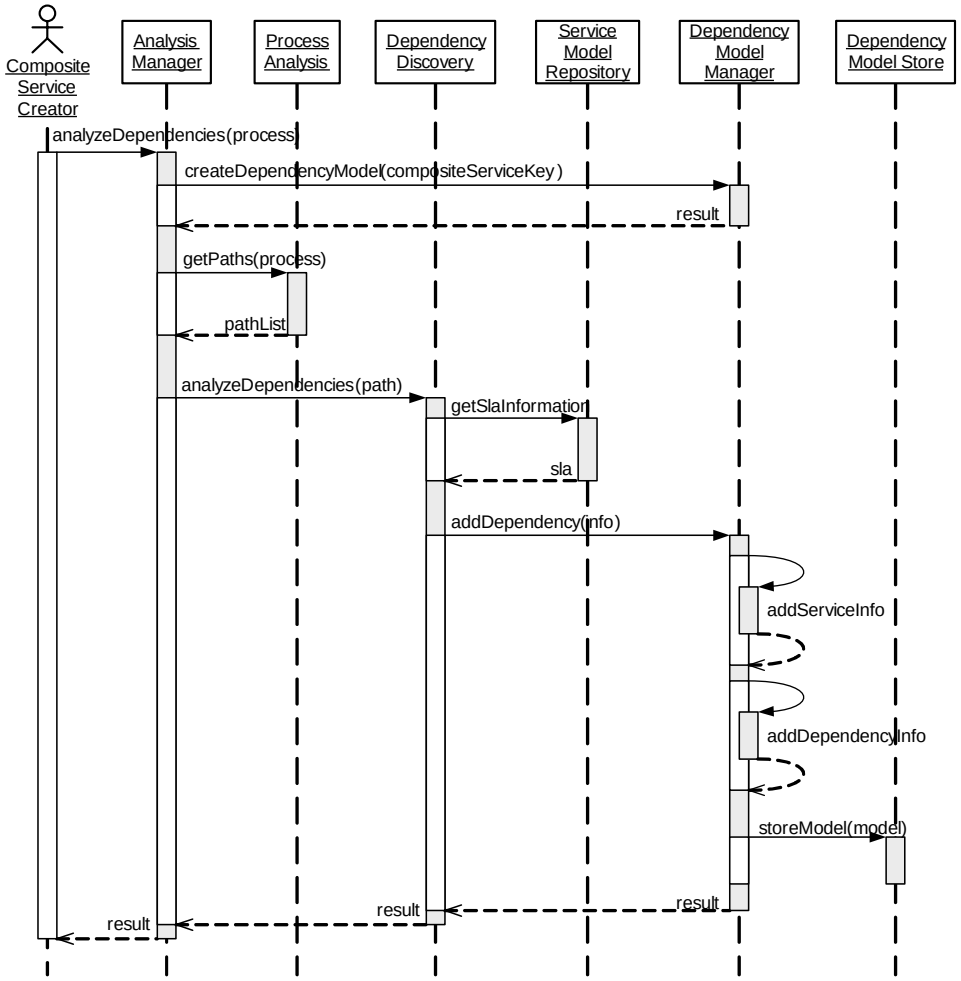

Figure 4.23: Dependency discovery steps

respective information to the dependency model. The new information, including service and dependency information, is added to the model. Once the information has been added to the model, it is stored in the Dependency Model Store. From there it can be accessed for further design time handling.

The second part of the dependency analysis is realized by the Dependency Modeler. It provides modeling functionality for the composite service creator, which enables the creation of new as well as the extension of existing dependency models. In Figure 4.24 the process and the involved components of the dependency modeling process are shown. The modeling process is initiated by the composite service creator. As a first step the Dependency Modeler requests a dependency model from the Dependency Model Manager. Once the model is available, the composite service creator uses

the editing functionality of the Dependency Modeler to add, remove, or modify dependency and service information in the model. Finally, the dependency model is stored again.

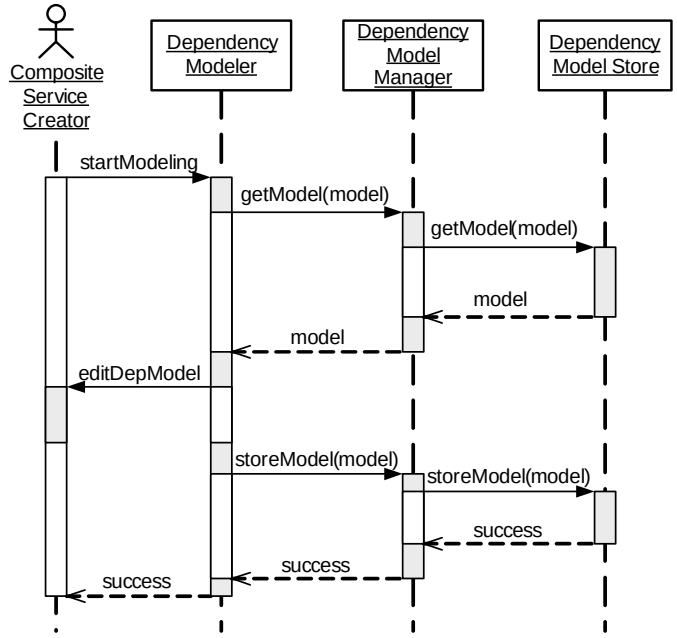

Figure 4.24: Modeling of dependencies at design time

Runtime Dependency Evaluation

The *Runtime Dependency Evaluation* components are responsible for the evaluation of dependency information at runtime. This thesis covers two major use cases that require the evaluation of dependency information. The occurrence of SLO violation information requires the determination of effects of this violation on other services (atomic or composite service). The occurrence of SLA renegotiation requests require an evaluation of the effects of this request on other services within the composition as well as the composition itself. Based on the result of this evaluation

- the renegotiation request is accepted, if it does not affect other services;
- effects on other services are detected and thus other SLAs need to be renegotiated before the current request is accepted or rejected;
- the renegotiation request is rejected, if effects on other services occur but cannot be resolved.

The process of dependency evaluation and the components involved in the process are presented in Figure 4.25. The process is initiated by the *ISE SLA Management* component in this example. It calls the Dependency Evaluation component, which executes the evaluation process. It first requests the dependency model from the Dependency Model Manager. Then it retrieves all dependencies involving the service under observation as well as relevant SLA information. The different dependencies are then evaluated with respect to the current event and affected services are determined. They are displayed to the user via the *Cockpit*.

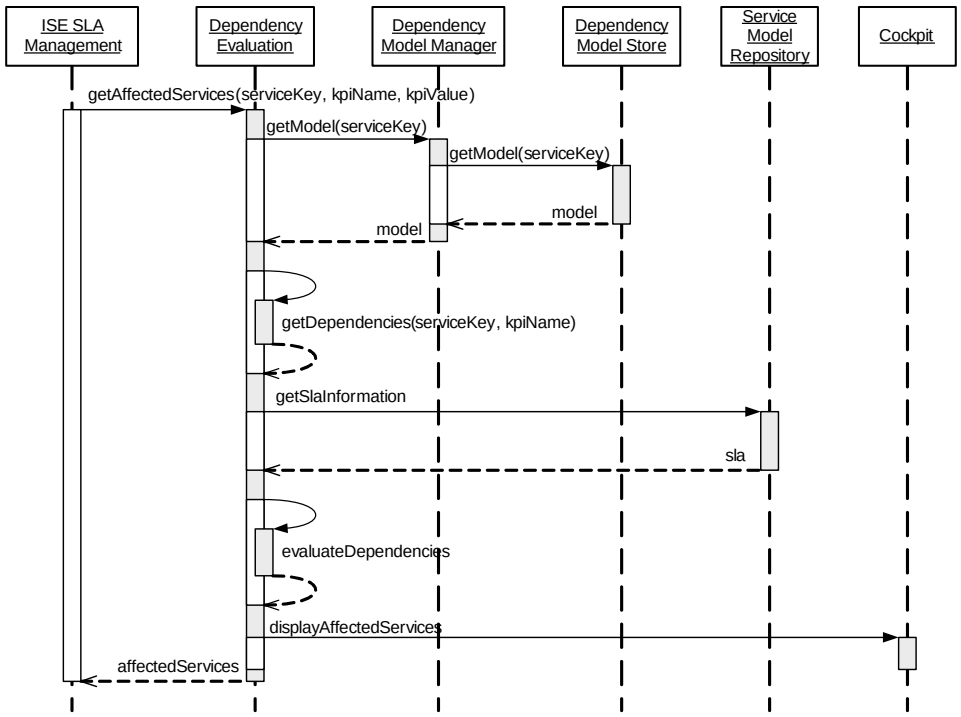

Figure 4.25: Evaluation of dependencies at runtime

```
1 public List<Service> getAffectedServices(String
      serviceKey, String kpiName, Object kpiValue)
2 {
3 }
```

Listing 4.9: The interface getAffectedServices

4.6 Summary

In this chapter an approach for the management of dependencies in service compositions was presented.

As a first step the nature of service dependencies was discussed. The notion of horizontal and vertical dependencies was introduced. Furthermore, a classification of different dependency types was presented.

As a second step different concepts for the representation of dependencies were analyzed. Based on this analysis a dependency model was introduced. This model consists of a generic core for expressing dependency information as well as additional packages enabling the description of dependency type-specific information. The dependency model is a core contribution of this thesis, answering the third research question stated in Section 1.4.

As a third step, the semi-automatic creation of a dependency model, dependency model validation, and evaluation were described. These concepts implement a lifecycle for the management of service dependencies. This lifecycle describes the overall approach of managing dependency information in the context of composite SLA management and thus answers the first research question stated in Section 1.4. The semi-automatic process of dependency model creation enables a more efficient handling of dependency information. It answers the second research question.

Finally, an architecture realizing the dependency management approach was presented. Components implementing the different concepts were integrated into an environment for service design, trading, and provisioning. This architecture shows how the different components interact with each other and with the remaining infrastructure. While the design of the architecture was inspired by the TEXO architecture, the components for service dependency management could be integrated into other related architectures. The presented architecture answers the fourth research question.

5 Validation

This chapter serves as the validation of the concepts developed as part of this thesis and described in Chapter 4. First the validation approach is described and its goals are outlined (Section 5.1). Furthermore, a prototypical implementation of the concepts is presented (Section 5.2) and evaluated based on a set of test cases (Section 5.3). The approach is also validated in Section 5.4 with regard to its performance. The chapter concludes with an evaluation of the requirements (Section 5.5) presented in Section 2.5 and a discussion of the validation results (Section 5.6).

5.1 Validation Approach

The purpose of the validation is twofold. On the one hand the general feasibility of the dependency management approach is shown. On the other hand the performance of the approach is analyzed to show its applicability in real-world application scenarios.

The validation consists of three major steps. First a proof of concept of the presented dependency management concepts is provided. The different concepts were prototypically implemented and integrated into the TEXO infrastructure. As part of the validation the prototype is described. The prototype was implemented in order to show the functioning of the different concepts. It also serves as a basis for further evaluation steps.

Two extended scenario descriptions and a set of test cases are used in the second step to demonstrate the dependency management approach based on the prototype. As part of this step, important artifacts, which are results of test cases, are presented. For brevity reasons, only small examples are presented in this chapter, but the full information is available in the appendix.

In a third step the performance of the approach is analyzed. The different algorithms of the approach are discussed with regard to their computational complexity. Furthermore, a number of performance measurements, which were executed based on the prototype, are presented and discussed.

Based on these three steps the requirements described in Section 2.5 are evaluated with regard to their fulfillment by the concepts and the prototype.

5.2 Prototypical Implementation of the System

In order to prove the feasibility of the presented dependency management approach, the system was prototypically implemented. The different parts of the prototype were integrated into the TEXO infrastructure.

The SLA Manager and the SLO Monitoring were implemented as standalone applications as part of the TEXO Service Management Platform and Tradable Service Runtime respectively. The other components were integrated into the Eclipse-based ISE Development Environment. The ISE Development Environment provides a variety of modeling tools for service design. Functionality for SLA negotiation and the creation of dependency models extend this design time functionality. Furthermore, functionality for dependency management during service provisioning was added. This adds a new facet to the ISE Development Environment.

In order to provide a better understanding of the implementation of ISE and the integration of the design and runtime components, a brief description of the Eclipse platform is provided. After that the implementation of the different components of the prototype is described.

5.2.1 Eclipse Architecture Overview

The Eclipse platform, which was developed under the umbrella of the Eclipse Foundation [Fouc], provides comprehensive functionality for application development. The functionality of the Eclipse platform is provided by different plugins. A plugin is a component which conforms to the OSGi Service Platform Core Specification [All09] by The OSGi Alliance

currently available as Release 4. Plugins are started and stopped by the platform core. Based on this plugin infrastructure, Eclipse can be adapted by creating new plugins or removing existing ones.

Existing plugins provide a wide range of functionality, e.g. for model development and code generation (Eclipse Modeling Framework - EMF [Foub]) and support for the creation of graphical editors based on developed models (Graphical Editing Framework - GEF [Foud]). The Eclipse plugin technology in general, and the EMF and GEF frameworks in particular form an important basis for the implementation presented in this chapter.

On the user interface (UI) level the Eclipse platform provides a number of different technologies.

- The Standard Widget Toolkit (SWT) provides basic UI elements such as button and scrollbar [Dau04]. Instead of implementing these UI elements, SWT provides a platform independent interface to the windowing system of the underlying operating system.
- The JFace-API [Dau04] provides higher level UI elements. It builds directly on SWT and uses SWT UI elements to realize e.g. dialogs and wizards.
- The Eclipse Forms [Glo05] provide web page like look and feel to Eclipse based applications. Eclipse forms are based on the SWT and JFace technologies.

In Figure 5.1 an overview of the Eclipse architecture is presented with respect to the prototype implementation described in this chapter. The main functionality provided by the prototype was implemented in the form of Eclipse plugins. Several plugins also provide a user interface based on SWT, JFace, and Eclipse Forms. More information is provided in the following section.

5.2.2 Prototype Implementation

The implementation of the prototype comprises infrastructure components for service description, SLA handling, and dependency management. Figure 5.2 presents an overview of the implemented components as well as their interaction.

In this section the most important components are briefly described with regard to their implementation and integration within the architecture. Furthermore, special technical requirements of each component with regard to other components are described, in order to better illustrate the

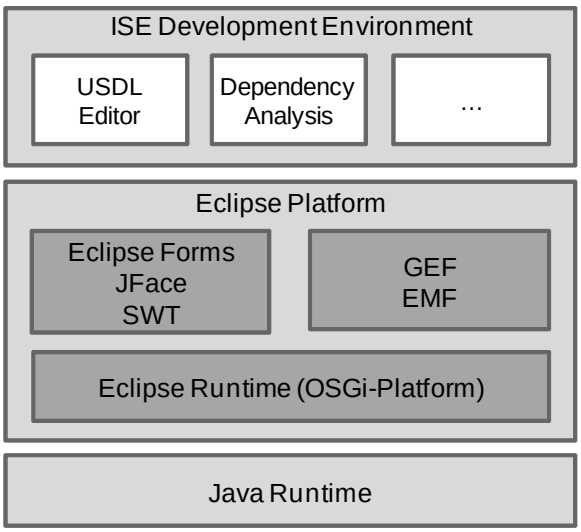

Figure 5.1: Implementation specific architecture

interplay of the different components. References are provided pointing to relevant sections of the concept chapter, which provides more detailed descriptions of the components functionality and the implemented concepts.

USDL Editor

The *USDL Editor* is one of several editors of the ISE Development Environment. It belongs to the service description components. Several extensions were made to the editor. First of all, functionality for the flexible modeling of SLOs was added. Furthermore, the input and output interface descriptions were extended to enable the specification of resource IDs, the start and end time of service provisioning, as well as its location. In order to support the handling of logistics services (i.e. one of the use cases), the USDL editor was extended with functionality to describe logistics services. Using the USDL editor, USDL documents are created by a service provider. They are either modeled manually, or partly generated by a model-to-model transformation from other existing models. USDL documents are needed for the generation of SLA templates. The USDL editor was implemented as an Eclipse plugin using the Eclipse Forms technology. Figure 5.3 shows the USDL editor during a USDL to WSAG transformation process.

Requirements: No requirements.

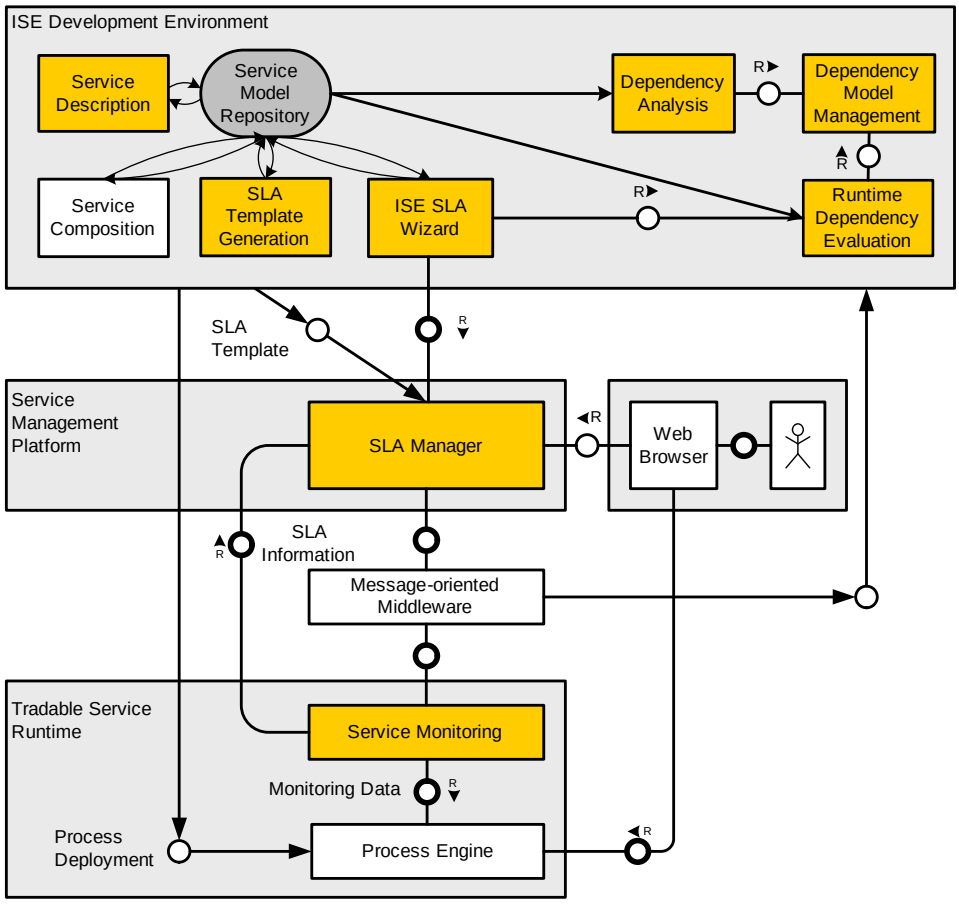

Figure 5.2: Overview of implemented components

SLA Template Generation

The SLA Template Generation plugin implements the SLA template generation as described in Section 4.5.1. The implementation uses openArchitectureWare [ope] to realize the transformation. It reads USDL service description documents and creates WSAG based SLA template documents. These SLA templates form the basis for SLA negotiation and are deployed to the service marketplace (i.e. Service Management Platform), together with the USDL service description. The transformation process is triggered manually by the service provider as soon as the modeling of the USDL service description is finished. Figure 5.3 shows a screenshot of the transformation wizard. The SLA Template Generation plugin is integrated into the ISE Development Environment.

Requirements: The SLA Template Generation plugin requires a valid USDL file as the starting point for a transformation from USDL to WSAG.

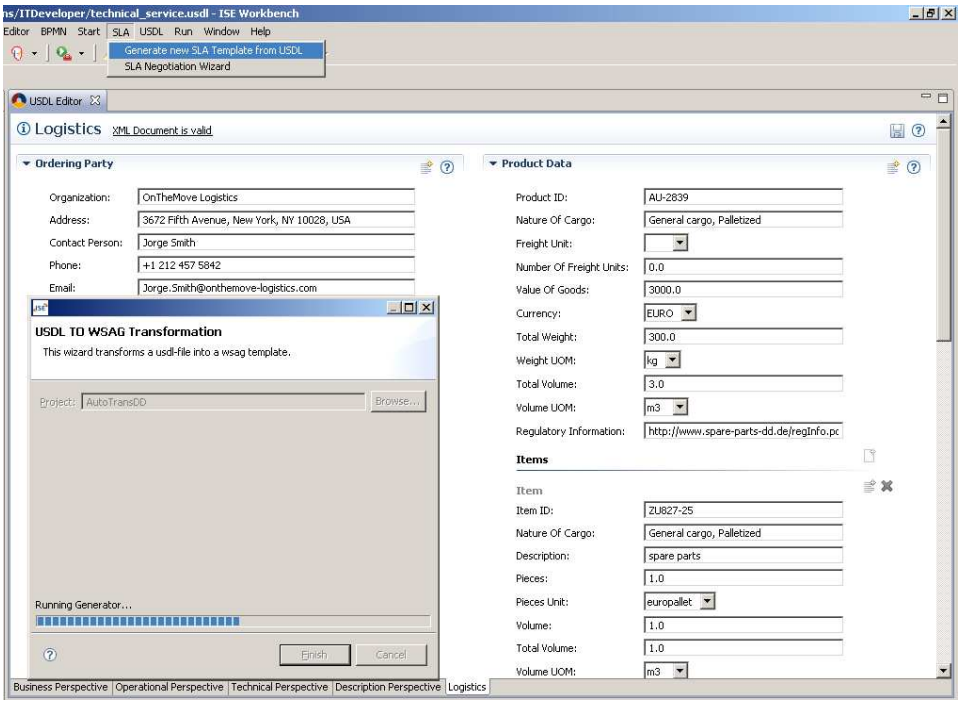

Figure 5.3: SLA Template Generation wizard and USDL editor

ISE SLA Wizard

In order to enable the negotiation of SLAs from within the ISE Development Environment, the ISE SLA Wizard plugin provides a suitable wizard. It was implemented using generic Eclipse wizard components. Figure 5.4 shows the ISE SLA Wizard. A more detailed description can be found in Section 4.5.1.

Requirements: The ISE SLA Wizard plugin requires a SLA Manager component to connect to during the SLA negotiation process.

SLA Manager

The SLA Manager is a standalone component, which provides functionality for SLA template deployment, SLA negotiation based on WSAG, SLA renegotiation, and SLA monitoring. It was implemented as a web service using JAX-WS (Java API for XML Web Services) technology. The SLA templates, offers, and agreements were realized based on JAXB (Java API for XML Binding). Both JAX-WS and JAXB are part of the Metro web service stack [SMCC]. The SLA Manager component is part of the Service Management Platform. It is integrated with other components (e.g. SLO Monitoring) via

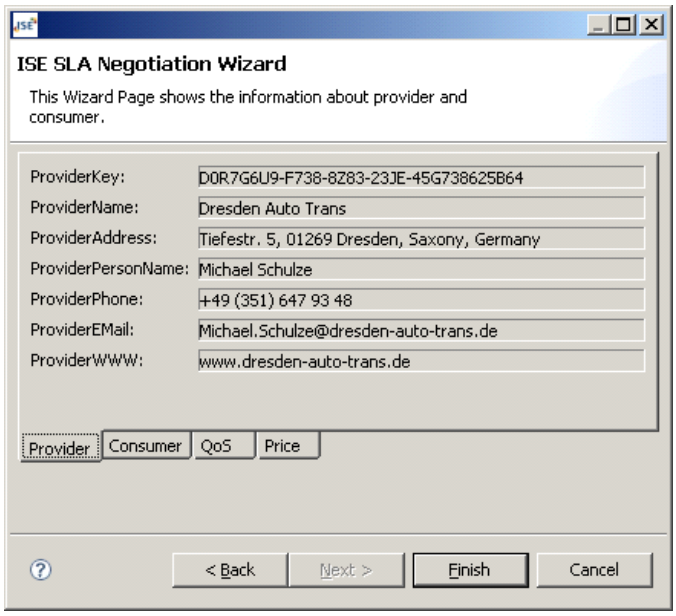

Figure 5.4: ISE SLA Wizard

a message-oriented middleware (MoM). A detailed description of the SLA Manager can be found in Section 4.5.1.

Requirements: No requirements.

SLO Monitoring

The SLO Monitoring component is a standalone Java component responsible for the monitoring of service level objectives based on provided monitoring information. It listens to a MoM for new monitoring information sent from monitoring sensors. In case it detects SLO violations it sends a violation message to interested components via another MoM. For this implementation the Apache ActiveMQ JMS implementation was selected as message-oriented middleware [Foua].

The SLO Monitoring component is part of Service Monitoring of the Tradable Service Runtime (see Section 4.5.1). It is loosely integrated with other components via a message-oriented middleware in order to receive monitoring information and to send SLO violation information to interested components.

Requirements: The SLO Monitoring component requires the SLA Manager in order to retrieve SLA information.

Dependency Analysis

In order to realize the functionality of the Dependency Analysis component, two Eclipse plugins were developed and integrated into the ISE Development Environment. The first plugin implements the dependency model discovery concept (see Section 4.4.2) including the analysis of workflows and the discovery of dependencies between services in service compositions. The second plugin implements a model editor (Dependency Modeler), which allows the manual creation of dependency models, service entities, and dependencies along with their properties. It also enables the modification of existing dependency models. In Section 4.4.2 the need to model non-detectable service dependencies was discussed. The editor was created based on the metamodel for expressing dependency models, which was described in Section 4.3.2. In Figure 5.5 the editor is depicted.

Requirements: The Dependency Analysis plugin requires the Dependency Model Management plugin in order to create and store dependency models.

Dependency Model Management

The Dependency Model Management plugin provides functionality for creating dependency models, storing them in the file system, and loading them when needed. This functionality is used during design time model creation as well as during runtime dependency model evaluation. Furthermore, this component provides functionality for the validation of dependency models (see Section 4.4.3). The Dependency Model Management plugin is integrated into the ISE Development Environment.

Requirements: No requirements.

Runtime Dependency Evaluation

The Runtime Dependency Evaluation component realizes the concepts described in Section 4.4.4. It was implemented as a plugin with the functionality for analyzing SLO violation events as well as renegotiation requests with regard to their effects on services. A cockpit view was integrated, which displays information about received SLO violations and renegotiation requests along with information about affected services. The cockpit was implemented based on the JFace technology. The plugin is integrated into the ISE Development Environment and receives messages via a message-oriented middleware.

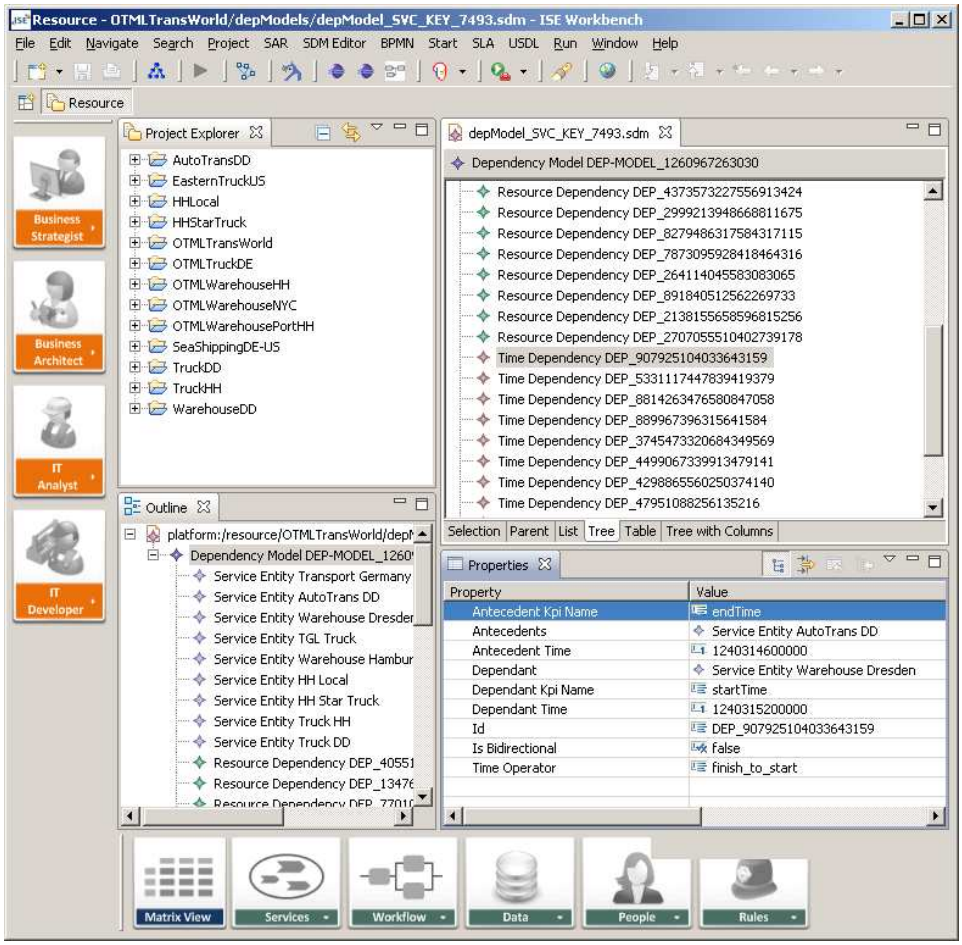

Figure 5.5: Tree-based editor for dependency modeling

Requirements: The plugin requires the Dependency Model Management plugin for access to the dependency models as well as for the validation functionality.

5.2.3 Discussion of Prototype Capabilities and Limitations

As a first part of the validation a prototypical implementation of the dependency management concepts was described. The prototype was integrated into the TEXO design and runtime infrastructure. Components for the creation, (re-)negotiation, and monitoring of SLAs provide the basic infrastructure for dependency management. A number of plugins realizing the design time and runtime management of dependencies were also implemented.

The prototype supports the full lifecycle of handling dependency models (see Section 4.4.1) and integrates dependency handling with the (re-) negotiation and monitoring of SLAs.

The prototype has a number of limitations:

- The **SLA management infrastructure** currently only supports a limited negotiation process. For use cases like logistics multiple roundtrips for negotiation would be necessary. Furthermore, the persistence of SLAs is currently realized by storing them into a folder within the SLA Manager as well as in the development environment. A suitable repository (e.g. a database) would be a useful extension of the prototype. Both limitations do not affect the concepts introduced by this thesis.

- The **storage of dependency models** is also realized by saving the respective XML files to a project folder. This does not limit the dependency management functionality.

- The **handling of QoS and price dependencies**, i.e. workflow reduction and discovery of a calculation formula, the validation of a calculation formula, and the evaluation at runtime were not implemented. This is due to the fact that these concepts were taken from existing work ([LF08, CMSA04, JRGM04]). The goal was to show that the functionality can be integrated with the work presented in this thesis.

- The **validation of resource** dependencies is limited to checking if resource dependencies are valid with regard to the negotiated SLAs. This allows finding errors introduced by manual dependency modeling. The validation of the correctness of the data flow in the composite service workflow was not implemented, since it is not in the main focus of this thesis.

5.3 Scenario-based Validation

In this section the dependency management approach will be validated based on two general use cases from the logistics and healthcare domains, which were introduced in Chapter 2. Fifteen test cases are used to analyze the functionality of the prototype, which was presented in the last section. The interaction between users and the dependency management components is illustrated.

The test cases follow the four lifecycle phases presented in Section 4.4.1:

- Creation and recomputation: TC-1 to TC-7
- Validation: TC-8 to TC-10
- Usage: TC-11 to TC-14
- Retirement: TC-15

They cover the different activities of dependency handling in relationship to the negotiation, renegotiation, and monitoring of SLAs, and describe them from the perspective of a composite service provider. All test cases cover both use cases. Several test cases include detailed examples for better illustration. These examples are specifically taken from the logistics and healthcare use cases where applicable (e.g. no QoS parameters are available for the healthcare use case). All test cases are described in Appendix A. Most test cases were implemented based on the prototype (the QoS dependency examples were not implemented due to the fact that the prototype does not support the discovery, validation, and evaluation of QoS dependencies).

A number of different artifacts (e.g. paths of activities or dependencies) are created throughout the process of handling dependencies. Examples of these artifacts are presented and discussed in this section. A detailed list of all artifacts can be found in the appendix.

5.3.1 Dependency Model Creation and Recomputation

During the first phase of the dependency handling lifecycle a dependency model is created. This happens during the SLA negotiation process of composite services. In Figure 5.6 the test cases TC-1 to TC-6 are illustrated, which cover the negotiation and dependency model creation process. Test case TC-7, i.e. the recalculation of the dependency model, follows the same steps as TC-2 to TC-6. In Table 5.1 the test cases of the dependency model creation process are summarized.

During the negotiation of SLAs (TC-1), which is supported by the SLA Manager and SLA Negotiation Wizard, SLAs are negotiated between the composite service provider on the one hand and the composite service consumer or the atomic service providers on the other hand. SLA documents in the form of SLA offers are stored in the Service Model Repository. Following that, the process for handling dependencies is initiated (TC-2).

TC-2 is detailed by test cases TC-3, TC-4, and TC-5, which cover the different steps of the dependency discovery process. During the creation of workflow paths (TC-3) a workflow description of the composite service is analyzed by the Dependency Analysis component. All linear paths leading

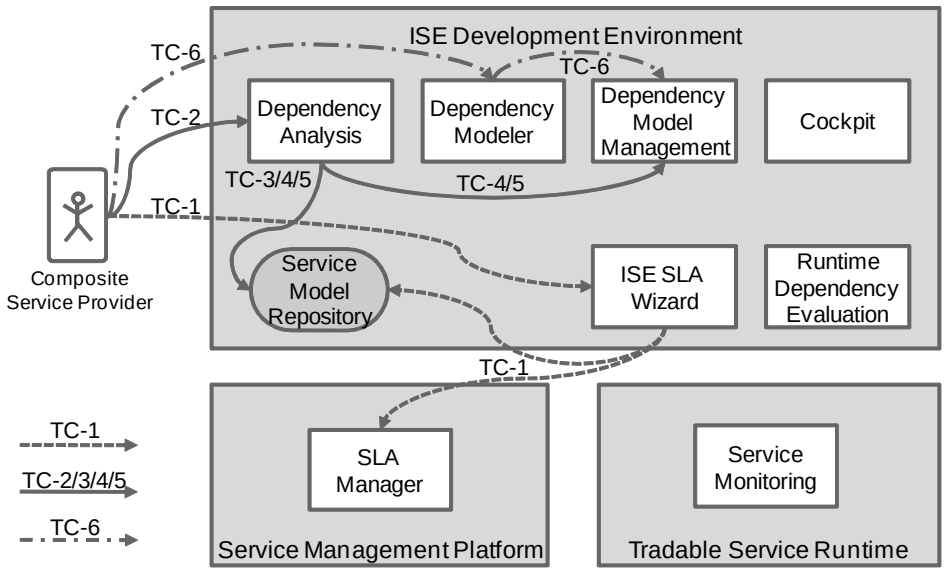

Figure 5.6: Illustration of test cases TC-1 to TC-6

from the start node to the end node are determined. In Table 5.2 two sample paths for the composite logistics and healthcare services are presented for illustration. The complete list of paths for both composite services (i.e. logistics and healthcare service) can be found in Appendix C and D.

During the creation of horizontal dependencies (TC-4 and examples), time and resource dependencies are discovered between the different services along a single path by the Dependency Analysis component. Duplicate dependencies, which occur due to different paths having equal sections, are detected and removed. The time and resource dependencies, which are presented in Table 5.3 and Table 5.4, are examples of discovered horizontal dependencies (first two dependencies) of the logistics and healthcare scenarios. A complete list of all dependencies for the logistics use case is shown in Appendix C. All dependencies from the healthcare scenario can be found in Appendix D. The necessary information for the analysis, i.e. the composite service workflow description and SLAs, are retrieved from the Service Model Repository. Finally, the dependencies are stored in a dependency model by the Dependency Model Management.

The creation of vertical dependencies (TC-5 and examples) supports the discovery of time, resource, price, and QoS dependencies. However, the prototype is limited with regard to supporting QoS and price dependencies. The implementation of vertical dependencies is realized by the same components of the prototype as the horizontal dependency discovery. Table 5.3 presents three vertical dependencies between *OTML TransWorld* and different atomic services from the logistics scenario. Appendix C shows

Table 5.1: Test cases for dependency model creation

Test case	Description	Realizing concept
TC-1	Negotiation of atomic and composite SLAs	SLA negotiation (4.5.1)
TC-2	Composite service provider initiates dependency model discovery	Dependency discovery (4.4.2)
TC-3	Creation of workflow paths	Path creation (4.4.2)
TC-4	Creation of horizontal dependencies	Discovery of horizontal dependencies (22)
TC-5	Creation of vertical dependencies	Discovery of vertical dependencies (17)
TC-6	Manual extension of dependency model	Modeling of non-detectable Service Dependencies (10)
TC-7	Recomputation of dependency model	Creation and recomputation of dependency models (4.4.2)

a complete list of all dependencies from the logistics use case. Table 5.4 shows similar dependencies from the healthcare scenario. The full dependency model is shown in Appendix D.

Since not all dependencies can be discovered automatically, it may be necessary (depending on the use case) to manually extend the dependency model (e.g. with location dependencies). This aspect is covered by test case TC-6. The last dependency shown in Table 5.3 is a location dependency, which was added to the dependency model manually using the Dependency Modeler.

Test case TC-7 covers the aspect of recalculating the dependency model upon changes. This is necessary e.g. when structural changes occur within the composite service workflow. Recalculation of the dependency model follows the same steps as the dependency model creation and is therefore not described in more detail.

5.3.2 Dependency Model Validation

After the successful creation of a dependency model, the model as well as the respective SLAs need to be validated. This validation functionality helps the composite service provider to only negotiate agreements, which enable the successful provisioning of the composite service. Table

Table 5.2: Linear path samples

Logistics	**Healthcare**
AutoTrans DD	Patient Admission
Warehouse DD	Patient Examination
Truck DE	Medical Record Creation
Warehouse HH	Determine Medication
HH Local	Procurement of Medication
Receive Goods	Give Medication
Prepare Shipment	Check Examination Results
Export Handling	Follow-up Treatment
Security Check	Discharge Patient
Ship Goods	
Sea Shipping DE-US	
OTML Warehouse NYC	
Eastern Truck US	

5.5 presents the different test cases for this task. They are also illustrated in Figure 5.7, along with the different components involved in each test case.

During conflict detection (TC-8) the created dependency model is analyzed with respect to the SLAs under negotiation. The validation is realized by the *Dependency Model Management* component. It retrieves the dependency model as well as the different SLA documents for the composite service from the *Service Model Repository*. Discovered problems are presented to the composite service provider. The following examples illustrate the conflict detection process for examples from the healthcare and logistics scenario:

Time dependencies (TC-8 EX1) were discovered based on the workflow structure of the composite service. Further dependencies may have been added manually. During conflict detection the negotiated start and end times for two depending services are compared with respect to the time operator of the dependency. In the logistics use case Eastern Truck US and OTML TransWorld have a finish-to-finish dependency, i.e. Eastern Truck US needs to finish before OTML TransWorld can finish. A selection of times where Eastern Truck US ends at 11:00 and OTML TransWorld ends at 12:00 is verified as valid.

Relevant dependencies: Eastern Truck US.endTime *finish-to-finish* OTML TransWorld.endTime

Evaluation: $11:00 \leq 12:00 \rightarrow$ OK

Result: The times of the two services are valid.

Table 5.3: Dependencies of logistics process

Antecedent - Dependent	Dependency	Description
Truck DD - Warehouse DD	time	endTime *finish-to-start* startTime
AutoTrans DD - Warehouse DD	resource	R4, R5, R6
OTML TransWorld - Truck DD	time	startTime *start-to-start* startTime
OTML TransWorld - AutoTrans DD	resource	R4, R5, R6
OTML TransWorld - all atomic services	QoS	max(TruckDD.maxTemperature, WarehouseDD.maxTemperature, ...)
Truck DD - Warehouse DD	location	endLocation *equals* startLocation

In the healthcare use case the services Patient Data Collection and Examine Blood have a finish-to-start dependency, i.e. Patient Data Collection needs to finish before Examine Blood can start. Assuming an initial selection of times where Patient Data Collection ends at 14:00 and Examine Blood starts at 13:00 an error occurs. The times need to be adapted.

Relevant dependencies: Patient Data Collection.endTime *finish-to-start* Examine Blood.startTime

Evaluation: $14:00 \leq 13:00 \rightarrow$ Error

Result: The times of the two services need to be adapted.

Table 5.4: Dependencies of healthcare process

Antecedent - Dependent	Dependency	Description
Patient Data Collection - Examine Blood	time	endTime *finish-to-start* startTime
Patient Admission - Examine Blood	resource	patientID
Stationary Patient Checkup - Patient Admission	time	startTime *start-to-start* startTime
Patient Transport - Expert Examination	location	endLocation *equals* startLocation

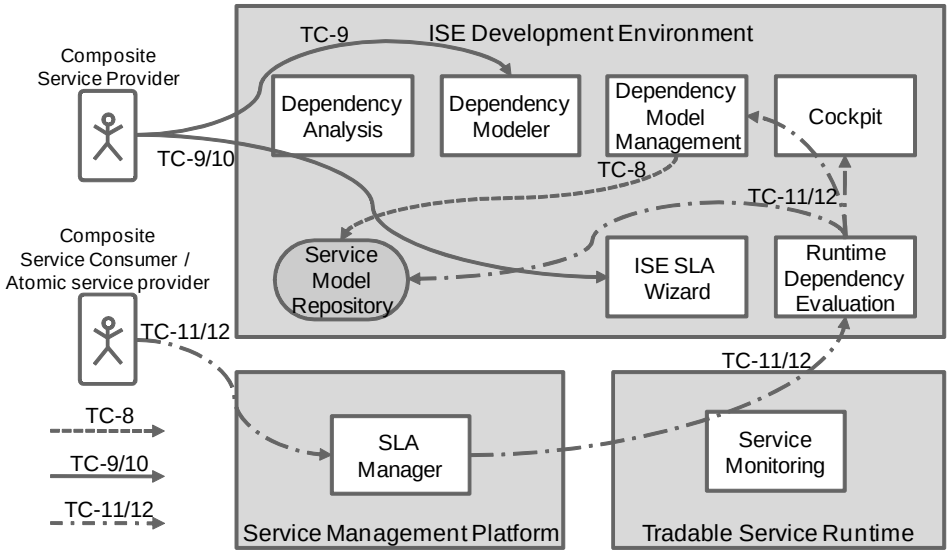

Figure 5.7: Illustration of test cases TC-8 to TC-12

Table 5.5: Test cases for dependency model validation

Test case	Description	Realizing concept
TC-8	Conflict detection / validation with regard to dependency model and negotiated SLAs	Dependency model validation (4.4.3)
TC-9	Conflict resolution between SLA offer documents	no support for automatic handling
TC-10	Finalization of SLA negotiation	SLA Negotiation Wizard (4.5.1)

Resource dependencies (TC-8 EX2) were discovered based on the description of input and output resources of SLA documents. Errors occur for example when a required input is not provided by any service. Similarly, there is a problem if the composite service provides an output, which is not provided by any atomic service and which was not an input to the composite service. Furthermore, any unmatched output represents a potential error. However, errors may have also been introduced through manual modification of the dependency model or changes to the SLAs. In the logistics use case the Export Handling at OTML Warehouse Port HH requires an export declaration as input. Since the export declaration is not provided by any service there is an error.

Evaluation: Analysis of output of other services as well as input of composite service.

Result: Error - export declaration not provided by any service.

In the healthcare use case the service Check Examination Results ha resource dependencies on several services. The validation shows that all required resources are provided.

Relevant dependencies:

- Check Examination Results.$resources_{in}$ *dependsOn* Examine Blood.$resources_{out}$
- Check Examination Results.$resources_{in}$ *dependsOn* Expert Examination.$resources_{out}$
- Check Examination Results.$resources_{in}$ *dependsOn* Medical Record Creation.$resources_{out}$

Evaluation: (laboratory test result, examination results, medical record) to be provided by:

- Examine Blood(laboratory test result)
- Expert Examination.$resources_{out}$(examination results)
- Medical Record Creation.$resources_{out}$(medical record)

$\rightarrow$ OK

Result: No problem detected.

Location dependencies (TC-8 EX3) were modeled by hand and thus require validation of the respective SLA information of two related services based on the defined location operator. In this example the dependency expresses that the start location of OTML TransWorld needs to be equal to the start location of service AutoTrans DD. Since both locations are equal the dependency is evaluated as correct.

Relevant dependencies: OTML TransWorld.startLocation *equals* AutoTrans DD.startLocation

Evaluation: Teegasse 83 *equals* Teegasse 83 $\rightarrow$ OK

Result: No problem detected.

In the healthcare scenario the Patient Transport end location needs to match the start location of the Expert Examination.

Relevant dependencies: Patient Transport.endLocation *equals* Expert Examination.startLocation

Evaluation: Unit for Internal Medicine *equals* Unit for Internal Medicine $\rightarrow$ OK

Result: No problem detected.

QoS dependencies (TC-8 EX4) are expressed as calculation formulas which are evaluated using the respective KPI values of the SLAs. In this example the composite service maxTemperature is greater or equal to the maxTemperature calculated for the different atomic services using the respective formula. This is verified as follows.

Relevant dependencies: OTML TransWorld.maxTemperature $\geq$ max(AutoTrans DD.maxTemperature, Truck DD.maxTemperature, Warehouse DD.maxTemperature, ...)

Evaluation: 15°C $\geq$ max(13°C, 14.5°C, 15°C, ...) $\rightarrow$ OK

Result: No problem detected.

For the healthcare use case no QoS properties were defined. Therefore, no QoS dependencies can be evaluated.

The evaluation and resolution of detected conflicts (TC-9) remains the responsibility of the composite service provider. No automatic support is provided to decide about the cause for the conflict. However, once the composite service provider has found the cause of the conflict, it needs to be resolved. This may include tasks such as making changes to SLA documents using the ISE SLA Wizard or refining the dependency model using the Dependency Modeler.

Once the dependency model has been validated and detected conflicts have been handled, the negotiation of SLAs can be finalized (TC-10). This task is driven by the composite service provider and supported by the SLA Manager and the SLA Negotiation Wizard components.

5.3.3 Dependency Model Usage

After the process of creating a dependency model and finalizing SLA negotiation, the dependency model is applied to fulfill two major tasks. The first task is the evaluation of renegotiation requests. Different stakeholders of a composite service, i.e. atomic service providers and the composite service consumer, have the chance to request the renegotiation of different SLOs of their SLAs. In Table 5.6 the different test cases for handling dependencies during renegotiation of SLAs of a composite service are summarized.

In test cases TC-11 and TC-12 (see also Figure 5.7) as well as in their respective examples the renegotiation of SLAs and the evaluation of these requests with respect to the dependency model are handled. The composite service consumer or an atomic service provider requests renegotiation via the SLA Manager, which forwards the request to the Runtime Dependency Evaluation component. Based on the dependency model and the

Table 5.6: Test cases for SLA renegotiation

Test case	Description	Realizing concept
TC-11	Request to renegotiate composite service SLO by consumer	SLA renegotiation (4.5.1) & Evaluating SLA renegotiation requests (4.4.4)
TC-12	Atomic service provider requests the renegotiation of a SLO	SLA renegotiation (4.5.1) & Evaluating SLA renegotiation requests (4.4.4)

negotiated SLAs the Dependency Model Management then determines the effects of the request.

The second task is the evaluation of SLO violation events with the goal of identifying other services, that are affected by the violation. SLO violations occur during service provisioning. The composite service consumer as well as the atomic service providers may be responsible for these violations. A SLO violation message is sent from the Service Monitoring component to the Runtime Dependency Evaluation component. The next steps are identical to the ones for evaluating renegotiation requests. Test cases 13 and 14 (see also Figure 5.8) as well as their respective examples (see Table 5.7) illustrate these violations and the resulting evaluation of dependencies.

For the evaluation of dependencies at runtime (upon the occurrence of renegotiation requests or SLO violations) three main steps are executed:

1. Determination of all dependencies involving the service under evaluation in the role of the antecedent (or dependent in case of bidirectional dependency). Only dependencies regarding the respective KPI are considered.

2. The detected dependencies are evaluated based on the data provided in the renegotiation request or the SLO violation info. Furthermore, the data of the negotiated SLAs is evaluated. In the case an affected service is found and the dependency is of transitive nature, the search process continues with the new service.

3. All affected services are displayed to the user.

Renegotiate location (TC-11 EX1): In this example the customer of OTML TransWorld wants to renegotiate the end location of the service. One dependency is found. The evaluation shows that Eastern Truck US will be affected by the change.

Renegotiation request: OTML TransWorld.endLocation = 3247 E River Road

Relevant dependencies: OTML TransWorld.endLocation *equals* Eastern Truck US.endLocation

Evaluation: 3247 E River Road *equals* 4878 N Broad Street → Error

Result: Eastern Truck US service will be affected.

In the healthcare use case a customer request to change the start or end location of the service is not useful. This would rather mean that the old contract needs to be canceled and a new contract with a different provider would be negotiated.

Renegotiate resource (TC-11 EX2): The customer wants to renegotiate its contract so that less goods are transported. The resource with ID R6 is removed. Two resource dependencies are found with regard to the input resources of OTML TransWorld. The evaluation shows that AutoTrans DD will be affected by this change. Furthermore, due to the transitive nature of resource dependencies, there is a need to check whether other services are affected when AutoTrans DD is affected.

Renegotiation request: Remove OTML TransWorld.$resources_{in}$ = R6

Relevant dependencies:

1. Truck DD.$resources_{in}$ *dependsOn* OTML TransWorld.$resources_{in}$
2. AutoTrans DD.$resources_{in}$ *dependsOn* OTML TransWorld.$resources_{in}$

Evaluation:

1. (R1, R2, R3) *contains* (R6) → Error
2. (R4, R5, R6) *contains* (R6) → OK

Result: AutoTrans DD service will be affected. Furthermore the following services are found to be affected: Warehouse DD, Truck DE, Warehouse HH, HH Local, Receive Goods, Prepare Shipment, Export Handling, Security Check, Ship Goods, Sea shipping DE-US, OTML Warehouse NYC, Eastern Truck US.

In the healthcare use case the customer wants to renegotiate the contract for the Stationary Patient Checkup service. The provisioning of the medical report is not needed any more.

Renegotiation request: Remove Stationary Patient Checkup.$resources_{out}$ = medical report

Relevant dependencies: Stationary Patient Checkup.$resources_{out}$ *dependsOn* Create Report.$resources_{out}$

Evaluation: (examination report, medical report) *contains* (medical report) $\rightarrow$ OK

Result: The service Create Report will be affected.

Renegotiate time (TC-11 EX3): In the logistics use case the service consumer wants to renegotiate the start time of the service. Two dependencies with regard to the composite service start time are found and evaluated. Only one of the services (Truck DD) will be affected by the change. Its start time is too early, so that an error occurs when evaluating the dependency.

Renegotiation request: OTML TransWorld.startTime = 17:15

Relevant dependencies:

1. OTML TransWorld.startTime *start-to-start* Truck DD.startTime
2. OTML TransWorld.startTime *start-to-start* AutoTrans DD.startTime

Evaluation:

1. 17:15 $\leq$ 17:00 $\rightarrow$ Error
2. 17:15 $\leq$ 17:30 $\rightarrow$ OK

Result: Truck DD will be affected

In the healthcare use case the service consumer wants to renegotiate the start time of the composite service. One dependency with regard to the composite service start time is found and evaluated. A conflict is detected.

Renegotiation request: Stationary Patient Checkup.startTime = 12:00

Relevant dependencies: Stationary Patient Checkup.startTime *start-to-start* Patient Admission.startTime

Evaluation: 12:00 $\leq$ 10:00 $\rightarrow$ Error

Result: Patient Admission will be affected

Renegotiate QoS (TC-11 EX4): The consumer wants to renegotiate the maxTemperature of the transport. Thus, the calculation formula is evaluated. One service is found to be affected by the change.

Renegotiation request: OTML TransWorld.maxTemperature = 14.5°C

Relevant dependencies: OTML TransWorld.maxTemperature *dependsOn* max(Truck DD.maxTemperature, AutoTrans DD.maxTemperature, Warehouse DD.maxTemperature, ...)

Evaluation: 14.5°C $\geq$ max(14.5°C, 13°C, 15°C, ...) $\rightarrow$ Error

Result: Warehouse DD service will be affected.

Renegotiate time (TC-12 EX1): Warehouse HH wants to renegotiate the end time of its service. Three dependencies with regard to the end time of Warehouse HH are found and evaluated. A conflict is detected with services Truck HH and HH Local.

Renegotiation request: Warehouse HH.endTime = 18:30

Relevant dependencies:

1. Warehouse HH.endTime *finish-to-start* Truck HH.startTime
2. Warehouse HH.endTime *finish-to-start* HH Star Truck.startTime
3. Warehouse HH.endTime *finish-to-start* HH Local.startTime

Evaluation:

1. 18:30 $\leq$ 18:00 $\rightarrow$ Error
2. 18:30 $\leq$ 19:00 $\rightarrow$ OK
3. 18:30 $\leq$ 18:00 $\rightarrow$ Error

Result: Truck HH and HH Local will be affected

In the healthcare use case the provider of the Expert Examination service wants to renegotiate the start time of the service.

Renegotiation request: Expert Examination.startTime = 16:30

Relevant dependencies: Patient Transport.endTime *finish-to-start* Expert Examination.startTime

Evaluation: 17:00 $\leq$ 16:30 $\rightarrow$ Error

Result: Patient Transport will be affected

Renegotiate location (TC-12 EX2): In the logistics use case the provider of the Warehouse HH wants to refine (i.e. renegotiate) the location for goods pickup. The evaluation shows that the three local transport providers are affected.

Renegotiation request: Warehouse HH.endLocation = Gutestr. 76, Warehouse Gate 27

Relevant dependencies:

- Warehouse HH.endLocation *equals* Truck HH.startLocation
- Warehouse HH.endLocation *equals* HH Local.startLocation
- Warehouse HH.endLocation *equals* HH Star Truck.startLocation

Evaluation:

- Gutestr. 76, Warehouse Gate 27 *equals* Gutestr. 76 → Error
- Gutestr. 76, Warehouse Gate 27 *equals* Gutestr. 76 → Error
- Gutestr. 76, Warehouse Gate 27 *equals* Gutestr. 76 → Error

Result: The services Truck HH, HH Local, and HH Star Truck will be affected.

In the healthcare use case the service provider of the Expert Examination wants to renegotiate the location of the service. One dependency is found. The evaluation shows that Patient Transport to the examination will be affected by the change. Renegotiation of the end location of the service would follow the same pattern.

Renegotiation request: Expert Examination.startLocation = Unit for Internal Medicine - Center for Cardiology

Relevant dependencies: Patient Transport.endLocation *equals* Expert Examination.startLocation

Evaluation: Unit for Internal Medicine *equals* Unit for Internal Medicine - Center for Cardiology → Error

Result: Patient Transport service will be affected.

Renegotiate QoS (TC-12 EX3): Truck DD requests the renegotiation of the maxTemperature of the transport. Thus, the calculation formula is evaluated. No service is found to be affected by the change.

Renegotiation request: Truck DD.maxTemperature = 15.0°C

Relevant dependencies: OTML TransWorld.maxTemperature *dependsOn* max(Truck DD.maxTemperature, AutoTrans DD.maxTemperature, Warehouse DD.maxTemperature, ...)

Evaluation: 15.0°C ≥ max(15.0°C, 13°C, 15°C, ...) → OK

Result: No services will be affected.

Table 5.7: Test cases for handling SLO violations

Test case	Description	Realizing concept
TC-13	Atomic service provider violates SLA	Evaluating SLO violations (4.4.4)
TC-14	Consumer violates the composite SLA	Evaluating SLO violations (4.4.4)

Violate time (TC-13 EX1): Service AutoTrans DD violates its end time. One relevant dependency is found. Warehouse DD is affected by the violation.

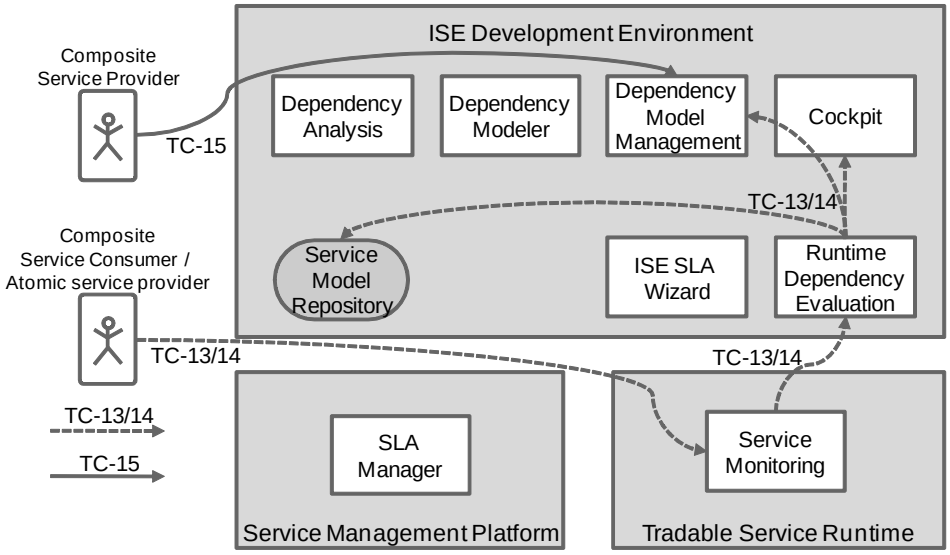

Figure 5.8: Illustration of test cases TC-13 to TC-15

While this example shows that for time violations the affected services are found, this information is not useful. The negative effect of the violation occurs at the time of the violation. No management steps are possible to avoid that.

Violation: AutoTrans DD.endTime = 20:30

Relevant dependencies: AutoTrans DD.endTime *finish-to-start* Warehouse DD.startTime

Evaluation: 20:30 $\leq$ 20:00 $\rightarrow$ Error

Result: Warehouse DD is affected.

In the healthcare use case the Procurement of Medication service violates its end time.

Violation: Procurement of Medication.endTime = 18:30

Relevant dependencies: Procurement of Medication.endTime *finish-to-start* Give Medication.startTime

Evaluation: 18:30 $\leq$ 19:00 $\rightarrow$ OK

Result: No service is affected.

Violate resources (TC-13 EX2): In this example Warehouse DD fails to deliver one of the required resources. Truck DE is found to be dependent on this resource and evaluated to be affected by the violation. Due to the transitive nature of resource dependencies further services are found to be affected.

Violation: Warehouse DD.$resources_{out}$ = R1 missing

Relevant dependencies: Truck DE.$resources_{in}$ *dependsOn* Warehouse DD.$resources_{out}$

Evaluation: (R1, R2, R3, R4, R5, R6) *contains* (R1) $\rightarrow$ OK

Result: Truck DE as well as several other services (Warehouse HH, Truck HH, Receive Goods, Prepare Shipment, Export Handling, Security Check, Ship Goods, Sea shipping DE-US, OTML Warehouse NYC, Eastern Truck US) will be affected.

In the healthcare example Examine Blood fails to deliver the results of the blood examination. A respective SLO violation message is sent. The service Check Examination Results is evaluated to be affected by the violation.

Violation: Examine Blood.$resources_{out}$ = laboratory test result missing

Relevant dependencies: Check Examination Results.$resources_{in}$ *dependsOn* Examine Blood.$resources_{out}$

Evaluation: (laboratory test result) *contains* (laboratory test result) $\rightarrow$ OK

Result: Check Examination Results will be affected.

Violate QoS (TC-13 EX3): In this example OTML Truck DE violates the negotiated maxTemperature. The calculation formula is evaluated with the result that no problem occurs with regard to the composition.

Violation: OTML Truck DE.maxTemperature = 15°C

Relevant dependencies: OTML TransWorld.maxTemperature *dependsOn* Truck DD.maxTemperature, AutoTrans DD.maxTemperature, Warehouse DD.maxTemperature, OTML Truck DE.maxTemperature, ...

Evaluation: 15°C $\geq$ max(14.5, 13, 15, ..., 15, ...) $\rightarrow$ OK

Result: No service will be affected.

Violate time (TC-14 EX1): In the logistics use case DSP fails to provide the resources on time and thus violates the start time. Truck DD and AutoTrans DD are affected.

Violation: OTML Trans World.startTime = 18:00

Relevant dependencies:

- OTML Trans World.startTime *start-to-start* Truck DD.startTime
- OTML Trans World.startTime start-to-start AutoTrans DD.startTime

Evaluation:

- 18:00 $\leq$ 17:00 $\rightarrow$ Error

- 18:00 $\leq$ 17:30 $\rightarrow$ Error

Result: Truck DD and AutoTrans DD are affected

In the healthcare use case the customer of the Stationary Patient Checkup service arrives late and thus violates the start time. The Patient Admission service is affected by the violation.

Violation: Stationary Patient Checkup.startTime = 10:30

Relevant dependencies: Stationary Patient Checkup.startTime *start-to-start* Patient Admission.startTime

Evaluation: 10:30 $\leq$ 9:00 $\rightarrow$ Error

Result: Patient Admission is affected

Violate resources (TC-14 EX2): The service consumer fails to provide the required resources for the composite service. A message about the missing resource is sent in the form of an SLO violation. Truck DD, AutoTrans DD, and Export Handling have a resource dependency on the composite service input resources. The Evaluation shows that Truck DD is affected by the violation, because it has a dependency regarding the specific resource. Further affected services are found due to the transitive nature of resource dependencies.

Violation: OTML TransWorld.$resources_{in}$ = R1 missing

Relevant dependencies:

1. Truck DD.$resources_{in}$ *dependsOn* OTML TransWorld.$resources_{in}$
2. AutoTrans DD.$resources_{in}$ *dependsOn* OTML TransWorld.$resources_{in}$
3. Export Handling.$resources_{in}$ *dependsOn* OTML TransWorld.$resources_{in}$

Evaluation:

- (R1, R2, R3) *contains* (R1) $\rightarrow$ OK
- (R4, R5, R6) *contains* (R1) $\rightarrow$ Error
- (export declaration) *contains* (R1) $\rightarrow$ Error

Result: Truck DD and further services (Warehouse DD, Truck DE, Truck HH, Receive Goods, Prepare Shipment, Export Handling, Security Check, Ship Goods, Sea shipping DE-US, OTML Warehouse NYC, Eastern Truck US) will be affected.

In the healthcare use case the composite service fails to provide the billing information.

Violation: Stationary Patient Checkup.$resources_{in}$ = billing information missing

Relevant dependencies: Patient Admission.$resources_{in}$ *dependsOn* Stationary Patient Checkup.$resources_{in}$

Evaluation: (billing information) *contains* (billing information) $\rightarrow$ OK

Result: The Patient Admission service will be affected.

5.3.4 Dependency Model Retirement

After the provisioning of a service was terminated (successfully or with errors) and the negotiated SLA is not valid any more, the dependency model is retired, i.e. it is removed from the *Dependency Model Store*. The respective functionality is provided by the *Dependency Model Manager*.

Table 5.8: Test case termination

Test case	Description	Realizing concept
TC-15	Terminate dependency handling	Dependency model retirement (4.4.5)

5.3.5 Discussion of Test Cases

In this section the functionality of the dependency management approach was analyzed using test cases. The test cases covered different steps of the entire dependency model lifecycle. It illustrated the discovery, validation, and evaluation functionality based on examples taken from a logistics and healthcare use case. Important created artifacts were presented.

The evaluation showed that the dependency management concepts presented in Chapter 4 form an integrated approach. It also made clear that the approach is general enough to be applied to use cases from different domains. Finally, the evaluation also illustrated the functionality of the prototype and with that confirms the feasibility of concept, i.e. the different concepts can be implemented and integrated into an existing architecture for service management.

5.4 Performance Evaluation

As a third part of the evaluation the performance of presented algorithms is analyzed. The determination of the runtime behavior of the different algorithms is important to prove their applicability for real life use cases. As a first step this section provides a theoretical discussion about the complexity of the different algorithms. As a second step a number of measurements are presented, which illustrate the performance of the algorithms.

5.4.1 Theoretical Discussion of Complexity

In order to determine the complexity of the presented dependency management approach, the different algorithms are analyzed with regard to their complexity. Computational complexity theory is concerned with determining the resources (e.g. time and space) needed for solving a certain problem. The complexity of an algorithm can be expressed using the *Big O notation* [Ski08]. It describes the upper bound resource usage of an algorithm as a function $f(n)$ where n is the input length for the algorithm. In order to determine the complexity of the different algorithms it is therefore necessary to analyze the algorithm behavior with regard to an increasing length of the input value. The relevant input varies for the different algorithms (e.g. number of activities in the workflow of the composite service, number of linear paths, number of dependencies in the dependency model). An increasing length of the input value occurs due to the varying length of a path or the number of elements in a dependency model.

The complexity of a workflow, which is the input e.g. for the path creation algorithm, is influenced by several relevant factors including the number of activities in the workflow, the number and type of split nodes, and the number of outgoing edges of the different split nodes.

Cardoso [Car06] argues that the workflow complexity depends on four main complexity characteristics, namely: control-flow complexity, data-flow complexity, activity complexity, and resource complexity. For the creation of paths the control-flow complexity is of major importance. In order to determine control-flow complexity the control-flow complexity measure (CFC) was introduced by Cardoso [Car05]. It calculates the complexity of a control-flow based on the combined complexity of the contained *AND*, *OR*, and *XOR* split nodes. The split node complexity is defined as "...the number of induced states that are introduced with the split..." [Car05]. An induced state is a valid state following a split node. Depending on the type of split node the same number n of outgoing edges introduces a different number of valid states. An *AND* split creates one valid state, a *XOR* split creates n valid states, and an *OR* split creates 2^{n-1} valid states. Accord-

ing to [Car05] the CFC calculation for a process P is performed using the following formula:

$$CFC(P) = \sum_{i \in \{XOR-splits\}} CFC_{XOR}(i) + \sum_{j \in \{OR-splits\}} CFC_{OR}(j) + \sum_{k \in \{AND-splits\}} CFC_{AND}(k)$$

The number of splits of a workflow and the number of valid states introduced by each split are the basis for calculating the control-flow complexity. For the creation of paths one important difference needs to be considered. The path creation does not distinguish the different split node types. Instead, it creates n linear paths for each kind of split. A further consideration is the arrangement of splits with regard to other splits. A different number of paths is created if two split nodes are arranged in a parallel or in a sequential fashion and if there is a join node between the two split nodes.

The CFC measure as well as the number of outgoing split node edges provide an indication of the complexity of the workflow. For that reason they are used in Section 5.4.2 as a basis for comparing the complexity of different workflows. However, for the determination of the computational complexity of the path creation algorithm, the number of activities in a workflow was considered.

The computational complexity of the following algorithms was determined. For each algorithm the input and complexity factors are listed.

Path creation algorithm (see Algorithm 1):

- **a:** number of activities in workflow
- **Relevant complexity factors:** number of paths in complete graphs grows exponentially
- **Complexity:** $O(2^a)$ (exponential complexity)

Algorithm to create horizontal time dependencies (see Algorithm 2):

- **a:** number of activities in a path
- **Relevant complexity factors:** *for* loop executed max a times
- **Complexity:** $O(a)$ (linear complexity)

Algorithm to create horizontal resource dependencies (see Algorithm 3):

- **a:** number of activities in a path; **r:** number of resources
- **Relevant complexity factors:** first *for* loop executed max a times and second *for* loop executed max a times; matching of each resource

- **Complexity:** $O(a^2 * r)$ (quadratic complexity)

Algorithm to create vertical time dependencies (see Algorithm 4):

- **a:** number of activities in a path
- **Relevant complexity factors:** basic assignments executed once
- **Complexity:** $O(1)$ (constant time)

Algorithm to create vertical resource dependencies (see Algorithm 5):

- **a:** number of activities in a path; **r:** number of resources
- **Relevant complexity factors:** outer *for* loop executed max a times and two inner *for* loops executed max a times; matching of each resource
- **Complexity:** $O(a^2 * r)$ (quadratic complexity)

Algorithm to add dependency to dependency model (see Algorithm 6):

- **d:** number of dependencies in dependency model
- **Relevant complexity factors:** basic assignments executed once
- **Complexity:** $O(1)$ (constant time)

Algorithm to validate dependency model (see Algorithm 7):

- **d:** number of dependencies in dependency model; **r:** number of resources (for resource dependencies)
- **Relevant complexity factors:** first for loop executed max $2d$ times and second loop executed max d times; for resource dependencies each resource needs to be validated
- **Complexity:** $O(d * r)$ (linear complexity)

Algorithm to find affected services (see Algorithm 8):

- **d:** number of dependencies in dependency model; **m:** number of dependency models; **r:** number of resources (for resource dependencies)
- **Relevant complexity factors:** outer loop executed max m times and inner loop executed max d times
- **Complexity:** $O(m * d * r)$ (linear complexity)

While most algorithms have constant time, linear or quadratic complexity, the path creation algorithm has exponential complexity. This results in an exponential complexity for the overall process of dependency model creation. As a consequence composite services with a very complex workflow cannot be handled by the approach. Two relevant factors are the number of activities in a workflow and the number of connections each activity has to other activities.

In order to get a realistic estimation of the number of activities within real-world workflows, the 82 SAP Integration Scenarios as well as the SAP R/3 Reference model were investigated. The SAP Integration Scenarios group process components of a business processes such as Patient Administration, Service Parts Management, and Supplier Relationship Management [SAP]. For each Integration Scenario the number of process components was determined. The largest Integration Scenarios contained 30 (Supplier Relationship Management) and 27 (Supplier Collaboration in the Supply Chain) process components.

In [MMN+06] the authors present a fault analysis study of more than 600 EPC business processes of the SAP R/3 Reference Model. These EPC processes (Event-driven Process Chain [KNS92]) are ordered into 29 industry branches. As part of that study the authors determined the mean number of functions for these business processes. Depending on the industry domain of the respective processes the mean number of functions ranges from 1.5 to 10.2 functions. These numbers provide an indicator regarding the typical size of business processes. A composite service with equivalent functionality would be likely to have the same number of atomic services, i.e. activities in its workflow.

It becomes clear that real-world workflows only contain a relatively small number of activities. Another important aspect is that activities in a workflow are typically connected to only a relatively small number of other activities, so that the resulting graph has a much lower complexity than a complete graph. Thus, the exponential time complexity of the dependency creation algorithm does not present a problem for discovering the dependencies in real-world workflows.

5.4.2 Performance Measurements for Use Cases

In this section the two use cases from the logistics and healthcare domain are used to analyze the performance of the different steps of the dependency management approach. The measurements cover the dependency model creation, i.e. path creation, discovery of horizontal and vertical dependencies, and validation, as well as the runtime dependency evalu-

ation. The discovery of QoS and price dependencies was not implemented. Therefore no measurements are presented for QoS and price dependencies.

The goal of this section is to analyze the performance of the different parts of the approach in order to show their applicability for business applications. This analysis focuses on temporal performance aspects.

Table 5.9 presents an overview of different attributes of the two use cases. Both have a similar number of activities. However, the healthcare process is slightly more complex than the logistics process. It has a larger number of outgoing split node edges and a higher control-flow complexity measure.

Table 5.9: Overview attributes of analyzed use cases

Artifact	**Logistics**	**Healthcare**
Activities	16	15
Split node edges	5	8
CFC	2	4

In order to derive the measured times, each measurement was executed 10000 times. The measured values were used to calculate an average time measurement. All performance measurements presented in this thesis were executed on a laptop with the following parameters:

- CPU: Intel Core 2 2.0 GHz
- Memory: 2 GB RAM
- Operating System: MS Windows XP Professional (Service Pack 2)
- Java Version: Java SE 6

Dependency Model Discovery

The dependency discovery process consists of the following main steps:

1. Creation of paths
2. Analysis of horizontal time dependencies
3. Analysis of vertical time dependencies
4. Analysis of horizontal resource dependencies
5. Analysis of vertical resource dependencies
6. Dependency model validation

Table 5.10 presents the average measurement results for the time taken to execute the different steps. The numbers of the different steps are used to identify the respective steps in the table.

The largest amount of time is used for the analysis of horizontal resource dependencies (logistics use case 10 ms and healthcare use case 14 ms). Here it is necessary to match input and output resources of the atomic services with other atomic services along a path. The time taken for the path creation, the remaining discovery steps, and the validation is relatively short.

Table 5.10: Performance analysis results for dependency discovery

Step	**1**	**2**	**3**	**4**	**5**	**6**
Logistics (time in ms)	4	5	2	10	2	1
Healthcare (time in ms)	5	7	2	14	1	2

Runtime Dependency Model Evaluation

In order to measure the performance of the dependency model evaluation algorithm, which is executed at runtime, different test cases from the logistics and healthcare use cases were executed.

The goal of these measurements is to analyze the performance of the approach for relevant exemplary dependency evaluations. However, it is important to state that the size of the dependency models was not varied. The dependency models for the two use cases are taken as a basis for these evaluations.

The measurements are presented in Table 5.11 and Table 5.12. One important observation from the results is that the evaluation of renegotiation requests and SLO violations of resource related SLOs takes much more time in the logistics use case. This is due to two facts. The resource dependencies in the logistics example occur with regard to a larger number of resources. Most resource dependencies occur with regard to 3 or 6 resources instead of only one resource in the healthcare use case. Furthermore, resource dependencies have a transitive nature. In the logistics example most resources are handled by many services consecutively. This is not the case in the healthcare scenario. The determination of services, which are affected due to the transitive nature, requires extra time. The test case 11 EX2 determines 14 affected services for the logistics use case but only 1 affected service for the healthcare use case.

Table 5.11: Performance analysis for dependency evaluation - 1

Test case	11 EX1	11 EX2	11 EX3	12 EX1	12 EX2
Logistics (time in ms)	24	298	31	39	37
Healthcare (time in ms)	-	17	19	19	17

Table 5.12: Performance analysis for dependency evaluation - 2

Test case	13 EX1	13 EX2	14 EX1	14 EX2
Logistics (time in ms)	23	193	28	222
Healthcare (time in ms)	17	18	19	42

Extended Performance Measurements

The work presented in [MMN+06] indicates that typical business processes only have a limited number of functions. However, the performance of the dependency management approach was tested also for much larger business processes. Several measurements were executed in addition to the ones presented before. Five different business process models (P1,..., P5) with increasing complexity were modeled as BPMN workflow descriptions.

Different steps of the dependency model creation process were analyzed with regard to the number of artifacts being created during the respective step. Important steps are the creation of workflow paths and the discovery of horizontal as well as vertical dependencies. For the creation of paths, time measurements are also presented. One limitation of the analysis described here is that no SLAs were available for the different activities within each workflow. Thus, a matching of input and output resources was not possible. For that reason no realistic time measurements could be made for resource dependency analysis. However, the presented numbers provide an understanding of how many steps are necessary for the discovery of horizontal and vertical dependencies in complex business processes.

In Table 5.13 some key information is given on the five different processes. The number of activities ranges from 12 to 87. The number of split node edges refers to the number of outgoing edges from split nodes. This number provides an idea of the complexity of the processes. Finally, the Control-flow Complexity measure (CFC) for each process is provided.

Performance of Path Creation As a first step of the dependency analysis linear paths were created from the composite service workflow. In Section 5.4.1 the complexity of this algorithm was determined to be polynomial. The measurements undertaken and presented in this section ex-

Table 5.13: Overview of analyzed processes

Artifact	P1	P2	P3	P4	P5
Activities	12	30	34	53	84
Split node edges	8	19	35	45	78
CFC	9	70	39	44	173

tend this theoretical discussion with concrete measurements. Table 5.14 presents the number of different paths created for each workflow as well as the measured time for path creation. A general observation is that an increasing complexity of the workflow (according to the number of activities, the number of split node edges, or the CFC) does not always result in a larger number of created paths or a larger amount of time for path creation. For workflow P5, which has the highest complexity (i.e. number of activities and split node edges, and CFC), the number of created paths is smaller than the number of paths created for P4. This is due to the fact that the number of linear paths within a workflow also depends on the concrete workflow design.

Table 5.14: Performance analysis path creation

Artifact	P1	P2	P3	P4	P5
Paths	21	60	952	1821	908
Time (in ms)	1	2	11	21	10

Performance of Horizontal Dependency Discovery The second step during the creation of a dependency model is the discovery of horizontal dependencies. Time and resource dependencies are discovered along each path created during the prior step. One important aspect is, however, that the different paths may have common segments. This is due to the fact that for each outgoing edge of a split node a new path is created. The first part of the different paths up to the split node are equal. After a join node equal path segments are created as well. An important implication of this aspect is that different combinations of services are handled multiple times, i.e. once in each path where they occur. In order to account for that, a check is executed, which avoids the creation of duplicate dependencies.

In order to illustrate the performance of the horizontal dependency discovery, the number of pairs of services handled by the respective algorithm was determined. For each occasion where a service pair was handled for a second (or n-th) time, the counter for duplicate pairs was increased. The actual number of handled service pairs is the sum of duplicate and non-duplicate pairs.

Table 5.15: Performance of horizontal dependency analysis

Artifact	P1	P2	P3	P4	P5
Non-duplicate time pairs	14	32	45	69	91
Duplicate time pairs	76	102	7387	13811	5535
Non-duplicate resource pairs	57	103	422	640	861
Duplicate resource pairs	222	139	33686	62724	21006

Table 5.15 shows the numbers of duplicate and non-duplicate time and resource pairs handled by the algorithms. Several observations can be made from the numbers:

- The increase of non-duplicate time pairs accompanying the increasing complexity of the workflow is very moderate. The increase of resource pairs is stronger. This is in line with the determined complexity of both operations, i.e. linear complexity of the horizontal time creation algorithm and quadratic complexity of the horizontal resource creation algorithm.
- The number of duplicate service pairs handled increases very strongly with the increasing number of paths created for a workflow.
- The number of handled duplicates surpasses the number of non-duplicate pairs by far. This implies that a large amount of time would be spent handling duplicates. However, since the different algorithms account for that, i.e. no two services are checked for dependencies twice, the time for handling duplicates is minimized.

Performance of Vertical Dependency Discovery The third major step of the process is the creation of vertical dependencies. Time and resource dependencies are discovered between the composite service and single services along the different paths. Similarly to the process of discovering horizontal resource dependencies, the algorithms for the discovery of vertical dependencies have to deal with duplicates. Services may occur within multiple paths. Table 5.16 presents the measurement results for the five different processes.

5.4.3 Discussion of Performance Evaluation Results

In this section different aspects of the performance of the dependency management approach were presented. As a first step the approach was analyzed with regard to its computational complexity. The complexity of the different algorithms was given in Big O notation. One result was that the

Table 5.16: Performance of vertical dependency analysis

Artifact	P1	P2	P3	P4	P5
Non-duplicate time pairs	7	27	13	32	55
Duplicate time pairs	35	93	1891	3610	1761
Non-duplicate resource pairs	24	60	68	106	168
Duplicate resource pairs	198	328	16700	31296	12900

creation of workflow paths and thus the dependency model creation has an exponential complexity. However, because the number of activities in real-world workflows is limited and each activity only has a limited number of connections to other activities, this complexity does not create a problem.

Following that the results of performance measurements for the different algorithms were presented. Based on two use cases from the logistics and healthcare domain time measurements for dependency model discovery (i.e. path creation, horizontal and vertical dependency discovery, validation) and runtime dependency evaluation were executed. In order to extend these measurements for more complex processes, five processes with increasing complexity were analyzed with regard to the number of operations necessary during the discovery process. The execution of measurements with regard to validation as well as runtime dependency evaluation was not executed because no SLAs were available for the services of these processes.

From this performance evaluation a number of results are important:

- The discovery of dependencies for small processes, such as the logistics and healthcare use cases (i.e. about 15 activities), has a duration of only a few milliseconds. Thus, the discovery process can be integrated into the process of negotiating SLAs for the services of a service composition without slowing down the work in a significant way.

- The evaluation of dependencies at runtime can be executed in less than 1 second. When a composite service provider receives SLA renegotiation requests or information about service level objective violations, information about affected services is gained very quickly and can be displayed along with the renegotiation or violation information.

- Handling of large processes, which contain more than about 20 activities, requires the consideration of handling duplicates. The number of operations necessary to handle non-duplicate services increases only in a moderate way as compared to the number of operations, which

would be necessary, if also duplicates would have to be analyzed. Avoiding the handling of duplicates thus ensures that even large processes can be handled efficiently by the approach. The number of necessary operations is below 1000 for all processes. In many cases it is even below 100. Analyzing also duplicate service pairs would result in up to around 62000 operations.

5.5 Evaluation of Requirements and Discussion of Results

As a final step of the validation the requirements, which were described in Section 2.5, are discussed with regard to their fulfillment by the concepts of this thesis.

DR1: Dependencies need to be expressed with a granularity of single service properties. A metamodel for representing dependencies between services was introduced in Section 4.3.2. It allows the expression of dependencies for each supported property, i.e. time, resource, location, different QoS properties, and price. **Fulfillment:** Fulfilled

DR2: The representation of dependencies needs to support the expression of multiple dependencies between two services. The metamodel allows the expression of different types of dependencies (e.g. time and resource) between two services. This is true for dependencies between the composite service and an atomic service as well as for dependencies between atomic services. **Fulfillment:** Fulfilled

DR3: The representation of dependencies needs to support the expression of multiple dependencies of the same type. The metamodel allows the expression of multiple dependencies of the same type. A single service in the role of the dependent may have a dependency of the same type on different services. It is also possible that different services depend on one service (antecedent) with regard to the same dependency type. **Fulfillment:** Fulfilled

DR4: The representation of dependencies needs to support the expression of type-specific dependency descriptions. The metamodel was designed so that each dependency has a type-specific description. **Fulfillment:** Fulfilled

DR5: The representation of dependencies needs to support the expression of dependencies between services with 1:1 and 1:n relationship. The metamodel for the dependency model enables the expression of 1:1 and 1:n relationships between a single dependent service and one or more antecedents. **Fulfillment:** Fulfilled

DR6: The representation of dependencies should support the automatic evaluation of dependencies. The dependencies are captured in a dependency model, which conforms to a formal metamodel. This lays the foundation for automatic processing. The model is stored in the form of an XML file, which is automatically handled by the Dependency Model Management component. **Fulfillment:** Fulfilled

MAR1: Dependencies between atomic services need to be managed. The discovery of horizontal dependencies enables the discovery of dependencies between atomic services. Test case TC-4 illustrated this concept. The validation and runtime handling of dependencies is independent of the occurrence of the dependency between atomic services or between a composite service and atomic services. **Fulfillment:** Fulfilled

MAR2: Dependencies between atomic services and the composite service need to be managed. The discovery of vertical dependencies fulfills this requirement. Furthermore, validation and runtime handling are supported. **Fulfillment:** Fulfilled

MAR3: Dependencies between services, which are directly or indirectly connected, need to be managed. The automatic creation of time, resource, QoS, and price dependencies supports the handling of dependencies between directly connected services. Resource dependencies can also be discovered between indirectly connected services. **Fulfillment:** Fulfilled

MAR4: The approach should support application domain experts in their work. The dependency management approach was developed with the goal to be used by domain experts. The modeling of composite services, the negotiation of SLAs, and the management of the compositions requires knowledge of the application domain. While the automatic discovery of dependencies does not impose any requirements with regard to the knowledge of the user, the manual extension of the dependency model requires an explicit understanding of the process underlying the composite service, i.e. domain knowledge. The tool for dependency modeling does not require specific technical knowledge beyond the notion of a dependency or the dependency description. **Fulfillment:** Fulfilled

GR1 - The work necessary to model service dependencies at design time should be minimized. In order to minimize the work necessary to model service dependencies between services the discovery of horizontal and vertical dependencies was developed (see Section 4.4.2). The test cases TC-2 to TC-5 demonstrate this concept. However, not all dependencies are discovered. Instead the approach allows for the manual extension of the automatically generated dependency model. **Fulfillment:** Partially fulfilled

GR2: The completeness of the recognized dependencies needs to be assured. The presented dependency management approach is capable of dis-

covering a variety of dependency types including time, resource, QoS, and price dependencies. QoS dependencies for different QoS properties and price dependencies are discovered completely. Different types of time dependencies are created between the composite service and atomic services (i.e. *start-to-start, finish-to-finish*), as well as between different atomic services along a path. These time dependencies are discovered completely. Time dependencies between services in parallel workflow paths or time dependencies of different types cannot be discovered. **Fulfillment:** Partially fulfilled

GR3: The effects of SLO violations as well as SLA renegotiation on other services need to be determined automatically. In Section 4.4.4 the concept for automatic evaluation of renegotiation requests as well as SLO violations was described. It determines all services which are affected by a renegotiation request or a SLO violation. Test cases TC-11 to TC-14 and the respective examples illustrated this concept. **Fulfillment:** Fulfilled

GR4: The approach should be flexible enough to be applicable for services from different application domains. In order to show that the approach is applicable to services from different domains, it was demonstrated based on use cases from two different domains in Section 5.3. **Fulfillment:** Fulfilled

GR5: The approach should be integrated with the methodology and toolset for developing and trading services in the Internet of Services. In Section 5.2 the prototypical implementation of the concepts was described. The different components for dependency management were integrated with the ISE Development Environment, the Service Management Platform, and the Tradable Service Runtime. **Fulfillment:** Fulfilled

GR6: The approach should be independent of a specific SLA technology and SLA management approach. The dependency management approach was designed to support the management of composite SLAs. The dependency management functionality is provided by self-contained components. They were described in Section 4.5.2. The functionality can be accessed via specific interfaces. Furthermore, the dependency information is kept separately from the SLA information in a dependency model (see Section 4.3.2). **Fulfillment:** Fulfilled

5.6 Summary and Discussion

In this chapter the concepts of the dependency management approach were validated. As a first step of the validation a prototypical implementation of the concepts was described. A number of components for the handling of dependencies at design time of the service, as well as during its provision-

ing, were implemented. Furthermore, these components were integrated into the ISE Development Environment and connected with the infrastructure for negotiating, renegotiating, and monitoring SLAs. The prototypical implementation shows that the presented dependency management concepts can be implemented and integrated into a suitable environment for service handling and SLA management.

As a second step the dependency management approach was demonstrated based on two use cases from the logistics and healthcare domain. A set of test cases covered the complete lifecycle of dependency management. For each test case the prototype components implementing this task were presented. Furthermore, important artifacts created during different steps were presented. Some test cases were illustrated by giving specific examples from both use cases. The goal of this test case based evaluation was to better illustrate the approach as well as to demonstrate the functioning of the prototype. The artifacts presented as results of dependency management tasks as well as the examples for some test cases show the correct functioning of the prototype for the two use cases. Since the two use cases come from different domains, the test cases demonstrate that the approach is general enough to be applied to different domains.

As a third step the performance of the approach was analyzed. The computational complexity of the different algorithms of the approach was outlined. To illustrate this theoretical work, a number of performance measurements were undertaken based on the prototype and the two use cases. The tests illustrate that the time needed to fulfill the different dependency management tasks is within the range of few milliseconds. Even for larger services the number of necessary operations can be limited. Thus, the approach can be applied to manage relatively complex composite services.

Finally, the evaluation of requirements showed that the concepts and the prototypical implementation fulfilled most requirements.

6 Summary and Outlook

This chapter briefly summarizes the main results of this thesis. Furthermore, the research questions identified in Section 1.4 are discussed with regard to the concepts presented in this thesis. On that basis the core contributions of this thesis are outlined, followed by an outlook on future work.

6.1 Summary and Discussion of Contributions

In the Internet of Services providers and consumers of services engage in business interactions by trading services via internet service marketplaces. Service provisioning is regulated by formal contracts, i.e. service level agreements. When service compositions, which are created by composing atomic services from different service providers, are sold to consumers, the composite service provider needs to manage a complex setup of SLAs. These SLAs were negotiated with the composite service consumer (i.e. the CSLA) and different atomic service providers (i.e. different ASLAs). Managing these SLAs is a complex task due to dependencies, which exist between the composite service and the atomic services as well as between the different atomic services. A system for the management of composite service SLAs needs to consider these dependencies.

Several approaches exist for the management of SLAs. However, they are mostly limited to the handling of single SLAs and do not consider dependency information. Existing composite service SLA management approaches, which also consider dependency information, are limited in sev-

eral ways. Limitations include missing support for handling dependencies between atomic services, inefficient processes for capturing dependency information, and tight coupling between dependency and SLA information as well as between the respective components for SLA and dependency management.

In order to resolve these limitations, this thesis introduced an approach for the management of dependency information. In Section 1.4 several research questions were raised. To provide solutions for these questions, different concepts were described.

1. Research question: What would be a suitable method for managing dependencies in service compositions at design time and during service provisioning?

A lifecycle for the handling of dependency information was developed. Dependency information is captured in a semi-automatic fashion during the design time of the service. This information is used to verify negotiated SLAs at design time during the negotiation process and to evaluate renegotiation requests as well as SLO violation events during service provisioning. Changes to the composite service, such as the adaptation of the service workflow structure or exchanging atomic services, may require the recomputation of the dependency model. Once the CSLA is terminated, the dependency model is retired. The different phases of the lifecycle were integrated with the composite SLA management approach, i.e. the negotiation and renegotiation of SLAs as well as their monitoring. Up to now no existing work in the area of composite SLA management presented a comprehensive approach for the handling of dependency information. All approaches have limitations with regard to the dependencies they can handle and the process of capturing dependency information.

2. Research question: How can dependencies between services be determined in an efficient manner? The dependency management approach enables the handling of different types of dependencies including time, resource, location, QoS, and price dependencies. The semi-automatic creation of the dependency model allows the discovery of various time and resource dependencies based on information in the composite service workflow description as well as the different ASLAs and the CSLA. Furthermore, the discovery of QoS and price dependencies was included into the dependency management approach. However, the concepts for this work originate from different related work. Most relevant is the work developed by Ludwig [Lud09]. Location dependencies as well as time, QoS, and price dependencies, which cannot be discovered automatically, can be modeled. This approach extends the state-of-the-art by further automating the dependency model creation process.

3. Research question: What is a suitable way to represent dependency information? As part of the dependency management approach relevant dependency information is captured explicitly in a dependency model in order to facilitate the handling of this information. A metamodel was presented in Section 4.3.2, which allows the representation of dependency information. The different services as well as their respective SLAs, which are relevant for the composite service, are referenced. The model also allows a detailed description of the different dependencies. Each dependency references the actual service properties in the SLA which cause the dependency. Thus, the dependency model is independent of the actual approach for formalizing SLAs. The design choice to keep SLA information and dependency information separately is different from the COSMA approach [Lud09] and rather follows different approaches such as the work presented by Tolksdorf [Tol03] or by Ensel and Keller [EK02]. However, the metamodel was specifically created to support SLA management and thus has a different design from these approaches.

4. Research question: What would be a suitable architecture to support the handling of service dependencies? An architecture for the handling of dependencies was developed (see Section 4.5). Components for dependency discovery and modeling, validation, evaluation, and model management were specified and their interplay was formally described. Furthermore, the integration into an architecture for composite service modeling and execution, as well as SLA negotiation, renegotiation, and monitoring was realized. The presented architecture decouples SLA management functionality and dependency handling from each other. This way the dependency handling can be integrated with different SLA management approaches more easily than a more tightly integrated approach.

Finally, the different concepts presented in this thesis were validated in three steps. In a first step a sample implementation of the presented concepts and architecture was realized. The components for dependency handling as well as for SLA management were implemented and integrated into an existing Internet of Services architecture. This implementation serves as a proof of concept for the dependency management approach as well as for the developed architecture. As a second step a number of test cases were specified based on two use cases from the healthcare and logistics domains. These test cases cover the different steps of the dependency handling and SLA management approaches. Several detailed examples provide a better illustration of the dependency management steps during SLA negotiation, renegotiation and monitoring. Using the developed prototype the different test cases and examples were implemented. The full process of dependency management was executed. This way the overall dependency management approach, the dependency discovery, and the metamodel were validated. As a third step a performance evaluation of the

dependency discovery and the runtime dependency evaluation was executed. The two logistics and healthcare use cases served as a basis for the different performance measurements. The results of these measurements showed that all dependency management tasks are achieved within a time frame of less than a second.

This thesis demonstrated, that the presented dependency management approach can be applied to support the management of SLAs is service compositions. Its functionality for handling dependency information during design time and run time extends SLA management functionality such as SLA negotiation, renegotiation, and monitoring. It enables composite service providers not only to manage single service SLAs but to consider the complex context of a service in a composition. In the IoS, where composite service providers offer services composed of a variety of other services, the consideration of dependency information for the management of composition is very important. It is especially relevant for the management of business services involving human tasks and the handling of material goods, because service failure is very costly in such cases.

6.2 Future Work

The dependency management approach improves existing work, e.g. by supporting different types of dependencies and partially automating the dependency model creation process. Based on the results of this work, possibilities for enhancing the developed approach were identified.

One interesting area for further work can be the extension of the system to handle different dependency types, which are currently not supported. An example is the support for handling dependencies regarding the resources needed for service execution (e.g. processing power or a certain work force). The extension of the overall approach to support such functionality would require an analysis of how to describe such a dependency and an extension of the dependency metamodel, i.e. adding a new package for capturing such dependencies. The core elements and structure of the metamodel should remain unchanged. Furthermore, the development of an approach to discover such dependencies is needed. Otherwise the manual modeling of this dependency information would be necessary. Finally, the functionality for evaluating this dependency information with regard to the different ASLAs and the CSLA would have to be developed.

A second option can be the extension of the discovery of dependencies. In the current approach, for example,location dependencies require explicit modeling due to the fact that not enough information is specified to enable the discovery. However, a trade-off exists between a more

detailed description of SLAs, which would enable better discovery options, and a reduced SLA description resulting in a need for the manual modeling of dependency information. Further work regarding dependency discovery requires an evaluation of these aspects. The discovery of QoS dependencies could also be extended. Existing approaches (e.g. [CMSA04, JRGM04, CP04, LF08]) provide a good basis for QoS dependency discovery. However, the number of supported QoS parameters is still limited. Furthermore, these approaches are limited to the composition of QoS from atomic values to composite ones. Breaking composite QoS values down into atomic values based on a certain workflow is still a very challenging task and is currently executed manually. An approach for supporting users with this work would be very helpful.

When the composite service is changed, this requires adjustments to the dependency model. Currently, the dependency model needs to be recomputed. This means that dependencies which were modeled manually need to be modeled again. A more advanced solution for recalculating the dependency model would facilitate this process. Only the dependency model parts which are related to the changed parts of the composite service should be recalculated.

The dependency management approach allows the determination of affected services upon the occurrence of SLO violation events or requests for renegotiation. However, it does not enable the automatic handling of these events. This would be a very helpful extension of the overall approach. A possible solution could involve the modeling of business rules, which specify how to react to certain events. Reactions will most likely vary depending on the business domain of the service as well as on the contract negotiated for the respective service. Options for reactions include the notification of the service that it will be affected by an error, or renegotiating its SLA. It may also be a valid option not to react and instead compensate the service provider for the problems.

Bibliography

[Aal97] AALST, Wil M. P. van d.: Verification of Workflow Nets. In: *ICATPN '97: Proceedings of the 18th International Conference on Application and Theory of Petri Nets*. London, UK : Springer-Verlag, 1997. – ISBN 3–540–63139–9, S. 407–426

[ACD+07] ANDRIEUX, Alain ; CZAJKOWSKI, Karl ; DAN, Asit ; KEAHEY, Kate ; LUDWIG, Heiko ; NAKATA, Toshiyuki ; PRUYNE, Jim ; ROFRANO, John ; TUECKE, Steve ; XU, Ming: *Web Services Agreement Specification (WS-Agreement)*. Specification. `https://forge.gridforum.org/projects/graap-wg/`. Version: March 2007. – Open Grid Forum

[AF08] AMELLER, David ; FRANCH, Xavier: Service Level Agreement Monitor (SALMon). In: *ICCBSS '08: Proceedings of the Seventh International Conference on Composition-Based Software Systems (ICCBSS 2008)*. Washington, DC, USA : IEEE Computer Society, 2008. – ISBN 978–0–7695–3091–8, S. 224–227

[All83] ALLEN, James F.: Maintaining knowledge about temporal intervals. In: *Commun. ACM* 26 (1983), Nr. 11, S. 832–843. – ISSN 0001–0782

[All09] ALLIANCE, OSGi: *OSGi Service Platform Core Specification Release 4*. OSGi Alliance Specification. `http://www.osgi.org/download/r4v42/r4.core.pdf`. Version: June 2009

[BCD08] BASU, Sujoy ; CASATI, Fabio ; DANIEL, Florian: Toward Web Service Dependency Discovery for SOA Management. In: *SCC '08: Proceedings of the 2008 IEEE International Conference on Services Computing*. Washington, DC, USA : IEEE Computer Society, 2008. – ISBN 978–0–7695–3283–7–02, S. 422–429

[BDB05] BARROS, Alistair ; DUMAS, Marlon ; BRUZA, Peter: The Move to Web Service Ecosystems. In: *BPTrends* (2005)

[BGO04] BAIDA, Ziv ; GORDIJN, Jaap ; OMELAYENKO, Borys: A shared service terminology for online service provisioning. In: *ICEC '04: Proceedings of the 6th international conference on Electronic commerce*. New York, NY, USA : ACM, 2004. – ISBN 1–58113–930–6, S. 1–10

[BIR] BIOMEDICAL INFORMATICS RESEARCH, Stanford C.: *protégé*. Project web page. `http://protege.stanford.edu`

[BKLW95] BARBACCI, Mario ; KLEIN, Mark H. ; LONGSTAFF, Thomas A. ; WEINSTOCK, Charles B.: Quality Attributes / Software Engineering Institute, Carnegie Mellon University. 1995 (ESC-TR-95-021). – Technical Report

[BLM08] BIANCO, Philip ; LEWIS, Grace A. ; MERSON, Paulo: Service Level Agreements in Service-Oriented Architecture Environments / Software Engineering Institute, Carnegie Mellon University. 2008 (CMU/SEI-2008-TN-021). – Technical Note

[BM04] BECKETT, Dave ; MCBRIDE, Brian: *RDF/XML Syntax Specification (Revised)*. W3C Recommendation. `http://www.w3.org/TR/rdf-syntax-grammar/`. Version: February 2004

[BPSMS07] BONIFACE, Mike ; PHILIPS, Stephen C. ; SANCHEZ-MACIAN, Alfonso ; SURRIDGE, Mike: Dynamic Service Provisioning Using GRIA SLAs. In: *Proceedings of ICSOC*, 2007

[Bri] BRITANNICA, Encyclopædia: *Council of Logistics Management*. Encyclopædia Britannica Online. `http://www.britannica.com/EBchecked/topic/346430/Council-of-Logistics-Management`. – Retrieved October 06, 2008

[BRS+08] BRAUN, Iris ; REICHERT, Sandro ; SPILLNER, Josef ; STRUNK, Anja ; SCHILL, Alexander: Zusicherung Nichtfunktionaler Eigenschaften und Dienstgüte im Future Internet of Services. In: *PIK - Special Issue on Service-oriented Computing* (2008)

[BSMM00] BRONSTEIN, I.N. ; SEMENDJAJEW, K.A. ; MUSIOL, G. ; MÜHLIG, H.: *Taschenbuch der Mathematik*. Bd. 5. Verlag Harri Deutsch, 2000

[BVGM07] BUCHSEIN, Ralf ; VICTOR, Frank ; GÜNTHER, Holger ; MACHMEIER, Volker: *IT-Management mit ITIL V3*. Friedr. Vieweg & Sohn Verlag, Wiesbaden, 2007

[BWRJ08] BODENSTAFF, Lianne ; WOMBACHER, Andreas ; REICHERT, Manfred ; JAEGER, Michael C.: Monitoring Dependencies for SLAs: The MoDe4SLA Approach. In: *IEEE SCC (1)*, IEEE Computer Society, 2008. – ISBN 978–0–7695–3283–7, S. 21–29

[Car05] CARDOSO, Jorge: Control-flow Complexity Measurement of Processes and Weyuker's Properties. In: *Engineering and Technology* Bd. 8 World Academy of Science, 2005

[Car06] CARDOSO, Jorge: Approaches to Compute Workflow Complexity. In: LEYMANN, Frank (Hrsg.) ; REISIG, Wolfgang (Hrsg.) ; THATTE, Satish R. (Hrsg.) ; AALST, Wil M. P. d. (Hrsg.): *The Role of Business Processes in Service Oriented Architectures* Bd. 06291, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2006 (Dagstuhl Seminar Proceedings)

[CH91] CLARK, John ; HOLTON, Derek A.: *A First Look at Graph Theory*. World Scientific Publishing Company, 1991

[CL06] CARDOSO, Jorge ; LENIC, Mitja: Web process and workflow path mining using the Multimethod approach. In: *Int. J. Business Intelligence and Data Mining* 1 (2006), Nr. 3, S. 304–328

[CLRS03] CORMEN, Thomas H. ; LEISERSON, Charles E. ; RIVEST, Ronald L. ; STEIN, Clifford: *Introduction to Algorithms*. Second Edition. MIT Press, 2003

[CMRW07] CHINNICI, Roberto ; MOREAU, Jean-Jacques ; RYMAN, Arthur ; WEERAWARANA, Sanjiva: *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. http://www.w3.org/TR/wsdl20/, June 2007. – W3C Recommendation

[CMSA04] CARDOSO, Jorge ; MILLER, John ; SHETH, Amit ; ARNOLD, Jonathan: Quality of service for workflows and web service processes. In: *Journal of Web Semantics* 1 (2004), S. 281–308

[CP04] CANFORA, Gerardo ; PENTA, Massimiliano D.: A lightweight approach for QoS-aware service composition. In: *In Proc. 2nd International Conference on Service Oriented Computing (ICSOC04) - short papers*, 2004

[CVW08] CARDOSO, Jorge ; VOIGT, Konrad ; WINKLER, Matthias: Service Engineering for The Internet of Services. In: FILIPE, Joaquim (Hrsg.) ; CORDEIRO, José (Hrsg.): *Enterprise Information Systems, 10th International Conference, ICEIS 2008, Barcelona, Spain, June 12-16, 2008, Revised Selected Papers* Bd. 19, Springer, 2008 (Lecture Notes in Business Information Processing), S. 15–27

[CWV09] CARDOSO, Jorge ; WINKLER, Matthias ; VOIGT, Konrad: *A Service Description Language for the Internet of Services*. 3 2009. – Proceedings of ISSS 2009 - International Symposium on Services Science

[Dau04] DAUM, Berthold: *Java-Entwicklung mit Eclipse 3*. 2. dpunkt.verlag, 2004

[DDO08] DIJKMAN, Remco M. ; DUMAS, Marlon ; OUYANG, Chun: Semantics and analysis of business process models in BPMN. In: *Inf. Softw. Technol.* 50 (2008), Nr. 12, S. 1281–1294. – ISSN 0950–5849

[EK02] ENSEL, Christian ; KELLER, Alexander: An Approach for Managing Service Dependencies with XML and the Resource Description Framework. In: *Journal of Network and Systems Management* 10 (2002), S. 147–170

[FL07] FARRELL, Joel ; LAUSEN, Holger: *Semantic Annotations for WSDL and XML Schema*. W3C Recommendation. `http://www.w3.org/TR/sawsdl/`. Version: August 2007

[FMA06] FRANKOVA, Ganna ; MALFATTI, Daniela ; AIELLO, Marco: Semantics and Extensions of WS-Agreement. In: *Journal of Software* 1 (2006), July, S. 23–31

[Foua] FOUNDATION, The Apache S.: *ActiveMQ*. Project web page. `http://activemq.apache.org/`. – [Online; accessed 30-December-2009]

[Foub] FOUNDATION, The E.: *Eclipse Modeling Framework Project (EMF)*. Project page. `http://www.eclipse.org/modeling/emf/`. – [Online; accessed 21-September-2009]

[Fouc] FOUNDATION, The E.: *Eclipse.org*. Project page. http://www.eclipse.org/. – [Online; accessed 23-December-2009]

[Foud] FOUNDATION, The E.: *Graphical Editing Framework (GEF)*. Project page. http://www.eclipse.org/gef/. – [Online; accessed 29-December-2009]

[FTS06] FLEHMIG, Marcus ; TROEGER, Peter ; SAAR, Alexander: *Design and Integration of SLA Monitoring and Negotiation Capabilities*. Adaptive Services Grid Deliverable D5.II-7. Version: August 2006. http://asg-platform.org/cgi-bin/twiki/view/Public/PublicDeliverables

[Glo05] GLOZIC, Dejan: *Eclipse Forms: Rich UI for the Rich Client*. Eclipse Corner Article. http://www.eclipse.org/articles/Article-Forms/article.html. Version: January 2005. – [Online; accessed 29-December-2009]

[Gmb09] GMBH ontoprise: *OntoStudio*. OntoStudio product web page. http://www.ontoprise.de/en/home/. Version: 2009

[Gro09] GROUP, Object M.: *Business Process Modeling Notation (BPMN) Version 1.2*. OMG Specification. http://www.omg.org/spec/BPMN/1.2. Version: January 2009

[Gru93] GRUBER, Thomas R.: Toward Principles for the Design of Ontologies Used for Knowledge Sharing / Knowledge Systems Laboratory, Stanford University. 1993 (KSL 93-04). – Technical Report

[HAW08] HEUSER, Lutz ; ALSDORF, Claudia ; WOODS, Dan: *Research Forum 2007*. Evolved Technologists Press, 2008. – 100 S.

[HBCS03] HULL, Richard ; BENEDIKT, Michael ; CHRISTOPHIDES, Vassilis ; SU, Jianwen: E-services: a look behind the curtain. In: *PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. New York, NY, USA : ACM, 2003. – ISBN 1–58113–670–6, S. 1–14

[HMP04] HEVNER, Alan R. ; MARCH, Salvatore T. ; PARK, Jinsoo: DESIGN SCIENCE IN INFORMATION SYSTEMS RESEARCH. In: *MIS Quarterly* 28 (2004), March, S. 75–105

[HP07] HEWLETT-PACKARD: *HP Discovery and Dependency Mapping software*. Data sheet, October 2007

[HP08] HEWLETT-PACKARD: *Predictive Change Impact Analysis and your Configuration Management System (CMS)*. White Paper, 2008

[Ini09] INITIATIVE, The Rule M.: *RuleML*. project web page. `http://ruleml.org/`. Version: 2009. – [Online; accessed 10-August-2009]

[JEA+07] JORDAN, Diane ; EVDEMON, John ; ALVES, Alexandre ; ARKIN, Assaf ; ASKARY, Sid ; BARRETO, Charlton ; BLOCH, Ben ; CURBERA, Francisco ; FORD, Mark ; GOLAND, Yaron ; GUIZAR, Alejandro ; KARTHA, Neelakantan ; LIU, Canyang K. ; KHALAF, Rania ; KÖNIG, Dieter ; MARIN, Mike ; MEHTA, Vinkesh ; THATTE, Satish ; RIJN, Danny van d. ; YENDLURI, Prasad ; YIU, Alex: *Web Services Business Process Execution Language Version 2.0*. OASIS Standard. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`. Version: April 2007

[JRGM04] JAEGER, M. C. ; ROJEC-GOLDMANN, G. ; MUHL, G.: QoS aggregation for Web service composition using workflow patterns. In: *Proceedings of the Eighth International IEEE Enterprise Distributed Object Computing Conference, 2004.* (2004), S. 149–159. `http://dx.doi.org/10.1109/EDOC.2004.1342512`. – DOI 10.1109/EDOC.2004.1342512

[KDA06] KONIEWSKI, Ryszard ; DZIELINSKI, Andrzej ; AMBORSKI, Krzysztof: Use of Petri Nets and Business Processes Management Notation in Modelling and Simulation of Multimodal Logistics Chains. In: BORUTZKY, Wolfgang (Hrsg.) ; ORSONI, Alessandra (Hrsg.) ; ZOBEL, Richard (Hrsg.): *Proceedings 20th European Conference on Modelling and Simulation (ECMS)*, 2006

[KMK09] KARAENKE, Paul ; MICSIK, Andras ; KIRN, Stefan: Adaptive SLA Management along Value Chains for Service Individualization. In: ALT, Rainer (Hrsg.) ; FÄHNRICH, Klaus-Peter (Hrsg.) ; FRANCZYK, Bogdan (Hrsg.): *Proceedings First International Symposium on Services Science (ISSS'2009)* Bd. 5, Logos Verlag Berlin, 2009 (Leipziger Beiträge zur Wirtschaftsinformatik), S. 217–228

[KNS92] KELLER, G. ; NÜTTGENS, M. ; SCHEER, A.W.: Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter Prozeßketten (EPK). In: *Veröffentlichungen des Instituts für Wirtschaftsinformatik* Heft 89 (1992)

[LDK04] LUDWIG, Heiko ; DAN, Asit ; KEARNEY, Robert: Cremona: an architecture and library for creation and monitoring of WS-agreents. In: *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*. New York, NY, USA : ACM, 2004. – ISBN 1–58113–871–7, S. 65–74

[LF08] LUDWIG, André ; FRANCZYK, Bogdan: COSMA–An Approach for Managing SLAs in Composite Services. In: BOUGUETTAYA, A. (Hrsg.) ; KRUEGER, I. (Hrsg.) ; MARGARIA, T. (Hrsg.): *ICSOC 2008*, Springer-Verlag Berlin Heidelberg, 2008 (LNCS 5364), S. 626–632

[LKD+03] LUDWIG, Heiko ; KELLER, Alexander ; DAN, Asit ; KING, Richard P. ; FRANCK, Richard: *Web Service Level Agreement (WSLA) Language Specification*. Specification. `www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf`. Version: 2003. – IBM

[LSE03] LAMANNA, D.D. ; SKENE, J. ; EMMERICH, W.: *Specification Language for Service Level Agreements*. Deliverable Tapas Project, March 2003

[Lud09] LUDWIG, André: *COSMA - Management of Service Level Agreements in Composite Services*. Logos Verlag Berlin, 2009 (Leipziger Beiträge zur Wirtschaftsinformatik 4)

[Mal07] MALINVERNO, Paolo: *How to Get Value Now (and in the Future) From SAP's Enterprise SOA*. GARTNER REPORT, 2007

[MC94] MALONE, Thomas W. ; CROWSTON, Kevin: The Interdisciplinary Study of Coordination. In: *ACM Computing Surveys* 26 (1994), March, Nr. 1, S. 87–119

[MFT+09] MILLER, George A. ; FELLBAUM, Christiane ; TENGI, Randee ; WAKEFIELD, Pamela ; LANGONE, Helen: *WordNet*. WordNet Online Database. `http://wordnet.princeton.edu/`. Version: 2009. – [Online; accessed 21-July-2009]

[MH01] MÖRSCHEL, Inka C. ; HÖCK, Hendrik: *Grundstruktur für die Beschreibung von Dienstleistungen in der Auss-*

chreibungsphase. Beuth Verlag GmbH, 2001. – Ref.Nr. PAS1018:2002-12

[MH04] MCGUINNESS, Deborah L. ; HARMELEN, Frank van: *OWL Web Ontology Language*. W3C Recommendation. `http://www.w3.org/TR/owl-features/`. Version: February 2004

[MLM+06] MACKENZIE, C. M. ; LASKEY, Ken ; MCCABE, Francis ; BROWN, Peter F. ; METZ, Rebekah: *Reference Model for Service Oriented Architecture 1.0*. OASIS Committee Specification, August 2006

[MM03] MILLER, Joaquin ; MUKERJI, Jishnu: *MDA Guide Version 1.0.1*. June 2003

[MMN+06] MENDLING, J. ; MOSER, M. ; NEUMANN, G. ; VERBEEK, H. M. W. ; DONGEN, B. F. V.: *A quantitative analysis of faulty EPC in the SAP reference model*. BPM Center Report BPM-06-08, BPMcenter.org. `http://is.tm.tue.nl/staff/wvdaalst/BPMcenter/reports/2006/BPM-06-08.pdf`. Version: 2006

[Mom06] MOMOTKO, Mariusz: *Integrated demonstrator for service composition enactment, monitoring and visualisation*. Adaptive Services Grid Deliverable D4.III-4. `http://asg-platform.org/cgi-bin/twiki/view/Public/PublicDeliverables`. Version: August 2006

[MPA+07] MEDEIROS, A.K. A. ; PEDRINACI, C. ; AALST, W.M.P. van d. ; DOMINGUE, J. ; SONG, M. ; ROZINAT, A. ; NORTON, B. ; CABRAL, L.: An Outlook on Semantic Business Process Mining and Monitoring. In: R.MEESMANN (Hrsg.) ; Z.TARI (Hrsg.) ; AL., P.Herrero et (Hrsg.): *OTM 2007 Ws, Part II*, Springer-Verlag, 2007 (LNCS 4806), S. 1244–1255

[MTV09] MODICA, Giuseppe D. ; TOMARCHIO, Orazio ; VITA, Lorenzo: A Framework for the Management of Dynamic SLAs in Composite Service Scenarios. In: NITTO, E. D. (Hrsg.) ; RIPEANU, M. (Hrsg.): *ICSOC 2007 Workshops* Bd. LNCS 4907, Springer-Verlag Berlin Heidelberg, 2009, S. 139–150

[N.J80] N.J.NILSSON: *Principles of Artificial Intelligence*. Tioga Publishing Company, 1980

[NRA08] NETJES, Mariska ; REIJERS, Hajo A. ; AALST, Wil M. d.: On the Formal Generation of Process Redesigns. In: ARDAGNA, Danilo (Hrsg.) ; MECELLA, Massimo (Hrsg.) ; JIANYANG (Hrsg.): *Business Process Management Workshops*, Springer, 2008 (LNBIP 17)

[OBB09] OBERLE, Daniel ; BHATTI, Nadeem ; BROCKMANS, Saartje: Countering Service Information Challenges in the Internet of Services. In: *Business & Information Systems Engineering* 1 (2009), 10, Nr. 5, S. 370–390

[OEC06] OECD: *Electronic Data Interchange (EDI)*. Glossary of Statistical Terms. `http://stats.oecd.org/glossary/detail.asp?ID=3418`. Version: January 2006. – last visited: 11.11.2008

[OMG05] OMG: *OCL 2.0 Specification*. OMG Specification. `http://www.omg.org/docs/ptc/05-06-06.pdf`. Version: June 2005

[OMG06] OMG: *Meta Object Facility (MOF) Core Specification*. OMG Available Specification 2.0. `http://www.omg.org/docs/formal/06-01-01.pdf`. Version: January 2006

[OMG09] OMG: *OMG Unified Modeling LanguageTM (OMG UML), Superstructure*. OMG Specification. `http://www.omg.org/spec/UML/2.2/Superstructure`. Version: February 2009

[ope] OPENARCHITECTUREWARE.ORG: *openArchitectureWare*. Project page. `http://www.openarchitectureware.org/`

[Ore67] ORE, Oystein: *Colloquium Publications*. Bd. 38: *Theory of Graphs*. American Mathematical Society, 1967

[O'S06] O'SULLIVAN, Justin: *Towards a Precise Understanding of Service Properties*, Queensland University of Technology, Diss., 2006

[OS08] OMER, Abrehet M. ; SCHILL, Alexander: A Framework for Dependency Based Automatic Service Composition. In: ARDAGNA, Danilo (Hrsg.) ; MECELLA, Massimo (Hrsg.) ; YANG, Jian (Hrsg.): *Business Process Management Workshops, BPM 2008 International Workshops, Milano, Italy, September 1-4, 2008. Revised Papers* Bd. 17, Springer, 2008

(Lecture Notes in Business Information Processing), S. 535–541

[Pas05] PASCHKE, Adrian: RBSLA A declarative Rule-based Service Level Agreement Language based on RuleML. In: *CIMCA '05: Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce Vol-2 (CIMCA-IAWTIC'06)*. Washington, DC, USA : IEEE Computer Society, 2005. – ISBN 0–7695–2504–0–02, S. 308–314

[PMI08] PMI: *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*. 4. Project Management Institute, 2008

[Pre08] PRESSEBÜRO, THESEUS: *TEXO - Business Webs im Internet der Dienste*. http://theseus-programm.de/scenarios/de/texo, March 2008

[RBFD01] REICHERT, Manfred ; BAUER, Thomas ; FRIES, Thomas ; DADAM, Peter: Realisierung flexibler, unternehmensweiter Workflow-Anwendungen mit ADEPT. In: HORSTER, P. (Hrsg.): *Proc. Elektronische Geschäftsprozesse–Grundlagen, Sicherheitsaspekte, Realisierungen, Anwendungen*, 2001, S. 217–228

[RBHJ07] ROSARIO, Sidney ; BENVENISTE, Albert ; HAAR, Stefan ; JARD, Claude: Probabilistic QoS and soft contracts for transaction based Web services orchestrations. In: *Proceedings of IEEE International Conference on Web Services (ICWS)*, 2007

[RM09] RINDERLE-MA, Stefanie: Data Flow Correctness in Adaptive Workflow Systems. In: *Emisa Forum* 29 (2) (2009), S. 25–35

[RPU+07] RAEDTS, Ivo ; PETKOVIĆ, Marija ; USENKO, Yaroslav S. ; WERF, Jan M. d. ; GROOTE, Jan F. ; SOMERS, Lou: Transformation of BPMN models for Behaviour Analysis. (2007), S. 126–137

[SAP] SAP: *Enterprise Services Workplace*. `http://esworkplace.sap.com/sdn`. – last visited: June 18, 2010

[SBPM09] STEINBERG, Dave ; BUDINSKY, Frank ; PATERNOSTRO, Marcelo ; MERKS, Ed ; GAMMA, Erich (Hrsg.) ; NACKMAN,

Lee (Hrsg.) ; WIEGAND, John (Hrsg.): *EMF Eclipse Modeling Framework Second Edition*. Addison Wesley, 2009

[SFRM08] STATHEL, Stephan ; FINZEN, Jan ; RIEDL, Christoph ; MAY, Norman: Service Innovation in Business Value Networks. In: *Proceedings of XVIII International RESER Conference 2008*, 2008

[SJSJ05] SANGAL, Neeraj ; JORDAN, Ev ; SINHA, Vineet ; JACKSON, Daniel: Using dependency models to manage complex software architecture. In: *Proceedings of the 20th annual ACM SIGPLAN conference on Object oriented programming systems languages and applications* 40 (2005), Nr. 10, S. 167–176. `http://dx.doi.org/http://doi.acm.org/10.1145/1103845.1094824`. – DOI http://doi.acm.org/10.1145/1103845.1094824. – ISSN 0362–1340

[Ski08] SKIENA, Steven S.: *The Algorithm Design Manual*. 2. Springer, 2008

[SMCC] SUN MICROSYSTEMS, Inc. ; COLLABNET, Inc. ; COGNISYNC, llc.: *GlassFish Metro*. GlassFish project page. `https://metro.dev.java.net/`. – [Online; accessed 07-January-2010]

[TBMM04] THOMPSON, Henry S. ; BEECH, David ; MALONEY, Murray ; MENDELSOHN, Noah: *XML Schema Part 1: Structures Second Edition*. W3C Recommendation. `http://www.w3.org/TR/xmlschema-1/`. Version: October 2004

[TMPE04] TOSIC, Vladimir ; MA, Wei ; PAGUREK, Bernard ; ESFANDIARI, Babak: Web Service Offerings Infrastructure (WSOI) - a management infrastructure for XML Web services. In: *Managing Next Generation Convergence Networks and Services, IEEE/IFIP Network Operations and Management Symposium, NOMS 2004, Seoul, Korea, 19-23 April 2004, Proceedings*, IEEE, 2004, S. 817–830

[Tol03] TOLKSDORF, Robert: A Dependency Markup Language for Web Services. In: AL., A.B. C. (Hrsg.): *Web Databases and Web Services 2002*, Springer-Verlag Berlin Heidelberg, 2003 (LNCS), S. 129–140

[TPP02] TOSIC, Vladimir ; PATEL, Kruti ; PAGUREK, Bernard: WSOL - Web Service Offerings Language. In: *CAiSE '02/ WES '02: Revised Papers from the International Workshop on*

Web Services, E-Business, and the Semantic Web. London, UK : Springer-Verlag, 2002. – ISBN 3–540–00198–0, S. 57–67

[VDATHKB03] VAN DER AALST, W. M. P. ; TER HOFSTEDE, A. H. M. ; KIEPUSZEWSKI, B. ; BARROS, A. P.: Workflow Patterns. In: *Distrib. Parallel Databases* 14 (2003), Nr. 1, S. 5–51. – ISSN 0926–8782

[VOH+06] VEDAMUTHU, Asir S. ; ORCHARD, David ; HIRSCH, Frederick ; HONDO, Maryann ; YENDLURI, Prasad ; BOUBEZ, Toufic ; YALÇINALP Ümit: *Web Services Policy 1.5 - Framework (WS-Policy)*. W3C Member Submission. `http://www.w3.org/TR/ws-policy/`. Version: April 2006. – last visited: 26.02.2010

[VVL07] VANHATALO, Jussi ; VÖLZER, Hagen ; LEYMANN, Frank: Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. (2007), S. 43–55. ISBN 978–3–540–74973–8

[Wae08] WAELDRICH, Oliver: *WSAG4J*. WSAG4J project page. `http://packcs-e0.scai.fraunhofer.de/mss-project/wsag4j/index.html`. Version: 2008. – last visited: 29.01.2009

[WPSB07] WU, Qinyi ; PU, Calton ; SAHAI, Akhil ; BARGA, Roger: Categorization and Optimization of Synchronization Dependencies in Business Processes. In: *Proceedings of IEEE 23rd International Conference on Data Engineering (ICDE'07)*, 2007, S. 306–315

[WS09] WINKLER, Matthias ; SPRINGER, Thomas: SLA Management for the Internet of Services. In: SHISHKOV, Boris (Hrsg.) ; CORDEIRO, José (Hrsg.) ; RANCHORDAS, AlpeshKumar (Hrsg.) ; INSTICC (Veranst.): *ICSOFT 2009 - Proceedings of the 4th International Conference on Software and Data Technologies, Volume 2, Sofia, Bulgaria, July 26-29, 2009* Bd. 2 INSTICC, INSTICC Press, 2009, S. 384–391

[ZBH08] ZHOU, Zhangbing ; BHIRI, Sami ; HAUSWIRTH, Manfred: Control and Data Dependencies in Business Processes Based on Semantic Business Activities. In: KOTSIS, Gabriele (Hrsg.) ; TANIAR, David (Hrsg.) ; PARDEDE, Eric (Hrsg.) ; KHALIL, Ismail (Hrsg.): *Proceedings of iiWAS2008*, ACM, 2008

[ZPPN07] ZHOU, Jiehan ; PAKKALA, Daniel ; PERALA, Juho ; NIEMELA, Eila: Dependency-aware Service Oriented Architecture and Service Composition. In: *IEEE International Conference on Web Services*, 2007, S. 1146–1149

[Zsc04] ZSCHALER, Steffen: Towards a semantic framework for non-functional specifications of component-based systems. In: STEINMETZ, Mauthe A. R. (Hrsg.): *Proc. EUROMICRO Conf. 2004, Rennes, France*, IEEE Computer Society, 2004

A

Validation Test Cases

TC-1 Negotiation of atomic and composite SLAs

Description: The composite service provider negotiates SLAs for the atomic services, which are part of the composite service. Upon customer request a SLA is also negotiated for the composite service.

Pre-conditions: The composite service provider has modeled a composite service and selected atomic service providers for all services. No SLAs have been negotiated with the atomic service providers or the composite service consumer.

Post-conditions: All SLAs are available as SLA offer documents.

Prototype support: The prototype supports this test case.

TC-2 Composite service provider initiates dependency model discovery

Description: The composite service provider initiates the dependency model creation process by starting the dependency discovery. He triggers the discovery process via the ISE Workbench. The Analysis Manager component loads the BPMN process description and the SLA offer documents for the respective services. The analysis procedure is executed.

Pre-conditions: The composite service provider has modeled a composite service workflow (in BPMN notation) and selected atomic service providers for all services. The SLA negotiation process has been started.

SLA offer documents or fully negotiated SLAs are available for all atomic services and the composite service.

Post-conditions: The first version of the dependency model is created.

Prototype support: The prototype supports this test case.

TC-3 Creation of workflow paths

Description: As a first step of the analysis procedure the composite service workflow is analyzed for all linear paths leading from the start node to the end node of the process.

Pre-conditions: The composite service workflow was modeled e.g. in BPMN notation.

Post-conditions: All linear paths for the composite service workflow are created.

Prototype support: The prototype supports this test case.

TC-4 Creation of horizontal dependencies

Description: Based on a number of linear paths the horizontal dependencies between the different activities within a path are determined.

Pre-conditions: Linear paths of composite service workflow are available.

Post-conditions: Time and resource dependencies between the atomic services in the different paths are created.

Prototype support: The prototype supports this test case.

TC-5 Creation of vertical dependencies

Description: Based on the linear paths vertical dependencies are discovered between the composite service and the atomic services.

Pre-conditions: Linear paths of composite service workflow are available.

Post-conditions: Vertical time and resource dependencies are created.

Prototype support: The prototype supports this test case.

TC-6 Manual extension of dependency model

Description: In order to extend and refine the dependency model, the composite service provider opens the created dependency model using the Dependency Modeler. If no dependency model exists up to that point, a new

model is created. This way it is also possible to create the dependency model in a fully manual fashion. After finishing the modeling, the dependency model is saved.

Pre-conditions: The composite service provider has modeled a composite service (in BPMN notation) and selected atomic service providers for all services. The SLA negotiation process has been started. SLA offer documents or fully negotiated SLAs are available for all atomic services and the composite service.

Post-conditions: The (extended) dependency model is saved.

Prototype support: The prototype supports this test case.

TC-7 Recomputation of dependency model

Description: Upon structural changes of the composite service workflow or changes to the SLAs of a service composition with regard to resources, the dependency model needs to be adapted. This is realized by recomputing the complete model.

Pre-conditions: The composite service workflow or the handled resources of services have been changed.

Post-conditions: The new dependency model has been created.

Prototype support: The prototype supports this test case.

TC-8 Conflict detection / validation with regard to dependency model and negotiated SLAs

Description: Based on the created dependency model the set of SLAs under negotiation are validated. Errors are detected based on the automatically discovered as well as modeled dependencies.

Pre-conditions: A dependency model was created and all SLAs have reached the status of an agreement offer.

Post-conditions: Detected errors are displayed to the composite service provider for further handling.

Prototype support: The prototype supports this test case.

TC-9 Conflict resolution between SLA offer documents

Description: When a conflict has been detected between the dependency model and the SLAs of the composite service, this conflict needs to be resolved by the composite service provider. There may be different reasons

for the conflict including modeling errors with respect to the composite service, false negotiated SLAs, or incorrectly specified dependencies in the dependency model. The conflict is handled by the composite service provider, e.g. by changing the respective SLA offers or by refining the dependency model, depending on the cause for the error.

Pre-conditions: A conflict is detected between the dependency model and the negotiated SLAs.

Post-conditions: The conflict, which was detected earlier, is resolved.

Prototype support: Conflict resolution is a manual task which is not supported by the prototype. However, the composite service provider uses the tools for dependency modeling or SLA negotiation to handle the conflict once the underlying cause was identified.

TC-10 Finalization of SLA negotiation

Description: After the successful completion of the validation of the dependency model and the respective SLA offers (i.e. no conflicts are detected any more), the negotiation of all SLAs is finished. First of all, the composite service SLA is sent to the consumer for approval. If it is accepted the respective offers of atomic service providers are also accepted. If the offer is rejected by the consumer further steps need to be determined by the composite service provider (e.g. replan the composite service or renegotiate atomic service SLAs).

Pre-conditions: Dependency model as well as all SLA offers are without conflict.

Post-conditions: All SLAs are successfully negotiated and the dependency model is available for runtime handling.

Prototype support: The prototype supports this test case.

TC-11 Request to renegotiate composite service SLO by consumer

Description: The consumer of the composite service wants to renegotiate its contract (the composite SLA). The request is received by the composite service provider. The Runtime Dependency Evaluation components extract the relevant information from the request and validates it. Based on the dependency model as well as relevant SLAs of other services, affected services are determined.

Pre-conditions: The dependency model and the SLAs for the respective service as well as potentially affected services are available.

Post-conditions: Services affected by the respective request have been determined. Depending on the special case the renegotiation request is successful or requires further handling by the composite service provider.

Prototype support: The prototype supports this test case.

TC-12 Atomic service provider requests the renegotiation of a SLO

Description: An atomic service provider requests the renegotiation of one of the SLOs of a contract (atomic SLA). The Runtime Dependency Evaluation components determine the effects of the request. In the case of effects on other services this information is displayed to the composite service provider.

Pre-conditions: The dependency model and the SLAs for the respective service as well as potentially affected services are available.

Post-conditions: Services affected by the respective request have been determined. Depending on the special case the renegotiation request is successful or requires further handling by the composite service provider.

Prototype support: The prototype supports this test case.

TC-13 Atomic service provider violates SLA

Description: A provider of an atomic service violates different SLOs of a SLA. The information about the violation is delivered to the composite service provider, where the information is evaluated and the effects are determined by the Runtime Dependency Evaluation components.

Pre-conditions: The dependency model and the SLAs for the respective service as well as potentially affected services are available.

Post-conditions: The services affected by the violation have been determined.

Prototype support: The prototype supports this test case.

TC-14 Consumer violates the composite SLA

Description: The consumer of the composite service violates its SLA.

Pre-conditions: The dependency model and the SLAs for the respective service as well as potentially affected services are available.

Post-conditions: The services affected by the violation have been determined.

Prototype support: The prototype supports this test case.

TC-15 Terminate dependency handling

Description: Once the contract for a composite service is terminated, the dependency model is removed from the Dependency Model Store. This is necessary in order avoid the further evaluation of this dependency model when SLO violations or renegotiation requests occur.

Pre-conditions: The contract for provisioning of the composite service has been terminated.

Post-conditions: The dependency model has been removed from the Dependency Model Store.

Prototype support: The prototype partially supports this test case. The retirement needs to be triggered manually because the SLA management system does not determine the state (e.g. terminated) of a SLA.

B

WS-Agreement SLA Document

```
1 <texoag:Template TemplateId="H738T378-U323-83H3-83GZ-839
      U348E8329"
2  xmlns:wsag="http://texo.net/wsag/type" xmlns:usdl="http:
       //texo.net/usdl/type"
3  xmlns:texoag="http://texo.net/texoag/type" xmlns:xsi="
       http://www.w3.org/2001/XMLSchema-instance"
4  xmlns:xs="http://www.w3.org/2001/XMLSchema">
5  <texoag:Name>OTMLTruckDE</texoag:Name>
6  <texoag:Context>
7   <wsag:TemplateId>H738T378-U323-83H3-83GZ-839U348E8329
8   </wsag:TemplateId>
9   <wsag:TemplateName>OTMLTruckDETemplate
10  </wsag:TemplateName>
11  <wsag:AgreementInitiator xsi:type="xs:string" />
12  <wsag:AgreementResponder xsi:type="xs:string">Jorge
13   Smith</wsag:AgreementResponder>
14  <wsag:ServiceProvider>AgreementResponder
15  </wsag:ServiceProvider>
16  <wsag:ExpirationTime>31-12-2010</wsag:ExpirationTime>
17  <usdl:provider>
18   <usdl:providerKey>Z283JD392-F738-8Z83-23JE-374
        ZE278ZUE23
19   </usdl:providerKey>
20   <usdl:providerName>OnTheMove Logistics</
        usdl:providerName>
21   <usdl:providerAddress>3672 Fifth Avenue, New York, NY
        10028, USA
22   </usdl:providerAddress>
```

```
23    <usdl:personName>Jorge Smith</usdl:personName>
24    <usdl:phone>+1 212 457 5842</usdl:phone>
25    <usdl:email>Jorge.Smith@onthemove-logistics.com
26    </usdl:email>
27    <usdl:www>www.onthemove-logistics.com
28    </usdl:www>
29   </usdl:provider>
30  </texoag:Context>
31
32  <texoag:Terms>
33   <texoag:All>
34    <texoag:ServiceDescriptionTerm>
35     <texoag:service>
36      <usdl:serviceKey>OTML Truck
37       DE_H738T378-U323-83H3-83GZ-839U348E8329</
             usdl:serviceKey>
38      <usdl:serviceType>Atomic</usdl:serviceType>
39      <usdl:serviceName>OTMLTruckDE</usdl:serviceName>
40      <usdl:serviceVersion>V1.0</usdl:serviceVersion>
41     </texoag:service>
42
43     <texoag:legal>
44      <usdl:providerRights>Provider has the right not to
           handle
45       the shipment request if there is any suspicions of
           unserious
46       business</usdl:providerRights>
47      <usdl:providerObligations></usdl:providerObligations
           >
48      <usdl:consumerRights>User is obliged to hand in no
           unserious
49       shipment requests</usdl:consumerRights>
50      <usdl:consumerObligations></usdl:consumerObligations
           >
51     </texoag:legal>
52     <usdl:marketing>
53      <usdl:price>
54       <usdl:payPerUse>
55        <usdl:priceName>OTMLTruckDE</usdl:priceName>
56        <usdl:priceCurrency>USD</usdl:priceCurrency>
57        <usdl:priceVat>13.0</usdl:priceVat>
58        <usdl:priceUnitType>Transportation exesution
59        </usdl:priceUnitType>
60        <usdl:priceUnit>600.0</usdl:priceUnit>
61        <usdl:priceCountry>USA</usdl:priceCountry>
62        <usdl:priceRegion></usdl:priceRegion>
63        <usdl:paymentMethod>MasterCard</usdl:paymentMethod
             >
64       </usdl:payPerUse>
```

```
65      </usdl:price>
66
67      <usdl:channels>
68       <usdl:channel>
69        <usdl:channelType>Promotional campaign
70        </usdl:channelType>
71        <usdl:channelName>Transport in Germany
72        </usdl:channelName>
73        <usdl:uri>www.onthemove-logistics.com/TG.avi
74        </usdl:uri>
75       </usdl:channel>
76      </usdl:channels>
77
78      <usdl:certifications>
79       <usdl:certification>
80        <usdl:certificationType>CSMS</
              usdl:certificationType>
81        <usdl:certificationName>CSA283-072
82        </usdl:certificationName>
83       </usdl:certification>
84      </usdl:certifications>
85     </usdl:marketing>
86     <usdl:rating>
87      <usdl:ratingType>Community</usdl:ratingType>
88      <usdl:ratingName>German transport stars</
           usdl:ratingName>
89      <usdl:ratingRange>1-5</usdl:ratingRange>
90      <usdl:ratingValue>1.0</usdl:ratingValue>
91      <usdl:ratingSupport>150.0</usdl:ratingSupport>
92     </usdl:rating>
93     <usdl:interaction />
94     <usdl:operational>
95      <usdl:classifications>
96      </usdl:classifications>
97      <usdl:operations>
98       <usdl:operation>
99        <usdl:operationName>OTMLTruckDE</
              usdl:operationName>
100       <usdl:operationInterfaces>
101        <usdl:operationInterface>
102         <usdl:interfaceName></usdl:interfaceName>
103         <usdl:inputInterface>
104          <usdl:ports>
105           <usdl:port>
106            <usdl:resourceType>Auto spare parts</
                  usdl:resourceType>
107            <usdl:resourceName></usdl:resourceName>
108            <resourceID>R1</resourceID>
109           </usdl:port>
```

```
110             <usdl:port>
111              <usdl:resourceType>Auto spare parts</
                     usdl:resourceType>
112              <usdl:resourceName></usdl:resourceName>
113              <resourceID>R2</resourceID>
114             </usdl:port>
115             <usdl:port>
116              <usdl:resourceType>Auto spare parts</
                     usdl:resourceType>
117              <usdl:resourceName></usdl:resourceName>
118              <resourceID>R3</resourceID>
119             </usdl:port>
120             <usdl:port>
121              <usdl:resourceType>Auto spare parts</
                     usdl:resourceType>
122              <usdl:resourceName></usdl:resourceName>
123              <resourceID>R4</resourceID>
124             </usdl:port>
125             <usdl:port>
126              <usdl:resourceType>Auto spare parts</
                     usdl:resourceType>
127              <usdl:resourceName></usdl:resourceName>
128              <resourceID>R5</resourceID>
129             </usdl:port>
130             <usdl:port>
131              <usdl:resourceType>Auto spare parts</
                     usdl:resourceType>
132              <usdl:resourceName></usdl:resourceName>
133              <resourceID>R6</resourceID>
134             </usdl:port>
135            </usdl:ports>
136            <time>2009-11-10T06:00:00+0100/2009-11-10
                   T08:00:00+0100
137            </time>
138            <location>Bahnhofstr. 85, 01269 Dresden,
139             Germany
140            </location>
141           </usdl:inputInterface>
142           <usdl:outputInterface>
143            <usdl:ports>
144             <usdl:port>
145              <usdl:resourceType>Auto spare parts</
                     usdl:resourceType>
146              <usdl:resourceName></usdl:resourceName>
147              <resourceID>R1</resourceID>
148             </usdl:port>
149             <usdl:port>
150              <usdl:resourceType>Auto spare parts</
                     usdl:resourceType>
```

```
151             <usdl:resourceName></usdl:resourceName>
152             <resourceID>R2</resourceID>
153            </usdl:port>
154            <usdl:port>
155             <usdl:resourceType>Auto spare parts</
                    usdl:resourceType>
156             <usdl:resourceName></usdl:resourceName>
157             <resourceID>R3</resourceID>
158            </usdl:port>
159            <usdl:port>
160             <usdl:resourceType>Auto spare parts</
                    usdl:resourceType>
161             <usdl:resourceName></usdl:resourceName>
162             <resourceID>R4</resourceID>
163            </usdl:port>
164            <usdl:port>
165             <usdl:resourceType>Auto spare parts</
                    usdl:resourceType>
166             <usdl:resourceName></usdl:resourceName>
167             <resourceID>R5</resourceID>
168            </usdl:port>
169            <usdl:port>
170             <usdl:resourceType>Auto spare parts</
                    usdl:resourceType>
171             <usdl:resourceName></usdl:resourceName>
172             <resourceID>R6</resourceID>
173            </usdl:port>
174           </usdl:ports>
175           <time>2009-11-10T12:00:00+0100/2009-11-10
                  T14:00:00+0100
176           </time>
177           <location>Gutestr. 76, 20095 Hamburg, Germany
178           </location>
179          </usdl:outputInterface>
180         </usdl:operationInterface>
181        </usdl:operationInterfaces>
182       </usdl:operation>
183      </usdl:operations>
184     </usdl:operational>
185     <usdl:logisticsService>
186      <transportService>
187       <transportZone>Germany</transportZone>
188       <transportationModes>ROAD</transportationModes>
189       <natureOfCargo>General Cargo</natureOfCargo>
190       <maxValueOfGoods>1000000.0</maxValueOfGoods>
191       <currency>EURO</currency>
192       <shipper>
193        <organization>Dresden Logistics</organization>
194        <address>Bahnhofstr. 85, 01269 Dresden, Germany
```

```
195          </address>
196          <contactPerson>Wilma Berger</contactPerson>
197          <phone>+49 (351) 455 84 45</phone>
198          <email>W.Berger@dresden-logistics.de
199          </email>
200          <www>www.dresden-logistics.de</www>
201        </shipper>
202        <consignee>
203          <organization>OnTheMove Logistics</organization>
204          <address>Gutestr. 76, 20095 Hamburg, Germany</
                 address>
205          <contactPerson>Helga Scholze</contactPerson>
206          <phone>+49 (40) 839 74 38</phone>
207          <email>H.Scholze@onthemove-logistics.com
208          </email>
209          <www>www.onthemove-logistics.com</www>
210        </consignee>
211        <source>Bahnhofstr. 85, 01269 Dresden, Germany</
               source>
212        <destination>Gutestr. 76, 20095 Hamburg, Germany
213        </destination>
214        <transportStages>
215          <stage>
216            <source>Bahnhofstr. 85, 01269 Dresden,Germany
217            </source>
218            <destination>Gutestr. 76, 20095 Hamburg,Germany
219            </destination>
220            <starttime>2009-11-10T06:00:00+0100/2009-11-10
                   T08:00:00+0100
221            </starttime>
222            <endtime>2009-11-10T12:00:00+0100/2009-11-10
                   T14:00:00+0100
223            </endtime>
224            <modeOfTransportation>ROAD</modeOfTransportation>
225            <meansOfTransport>Truck</meansOfTransport>
226            <logisticsServiceProvider>OnTheMove Logistics
227            </logisticsServiceProvider>
228          </stage>
229        </transportStages>
230
231        <orderingParty>
232          <organization>Dresden Spare Parts</organization>
233          <address>Teegasse 83, 01159 Dresden, Germany
234          </address>
235          <contactPerson>Herbert Hase</contactPerson>
236          <phone>+49 (351) 456-7451</phone>
237          <email>H.Hase@spare-parts-dd.de
238          </email>
239          <www>www.spare-parts-dd.de</www>
```

```
240         </orderingParty>
241
242         <logisticsServiceProviders>
243          <logisticsServiceProvider>
244           <organization>OnTheMove Logistics</organization>
245           <address>3672 Fifth Avenue, New York, NY 10028,
                 USA
246           </address>
247           <contactPerson>Jorge Smith</contactPerson>
248           <phone>+1 212 457 5842</phone>
249           <email>J.Smith@onthemove-logistics.com
250           </email>
251           <www>www.onthemove-logistics.com
252           </www>
253          </logisticsServiceProvider>
254         </logisticsServiceProviders>
255
256         <usdl:productData>
257          <productID>AU-2839</productID>
258          <usdl:natureOfCargo>General cargo, Palletized
259          </usdl:natureOfCargo>
260          <usdl:freightUnit>
261          </usdl:freightUnit>
262          <numberOfFreightUnits>0.0</numberOfFreightUnits>
263          <usdl:valueOfGoods>5546.0</usdl:valueOfGoods>
264          <usdl:currency>EURO</usdl:currency>
265          <usdl:totalWeight>600.0</usdl:totalWeight>
266          <usdl:weightUOM>kg</usdl:weightUOM>
267          <usdl:totalVolume>6.0</usdl:totalVolume>
268          <usdl:volumeUOM>m3</usdl:volumeUOM>
269          <regulatoryInformation>http://www.spare-parts-dd.
                 de/regInfo.pdf
270          </regulatoryInformation>
271          <items>
272           <usdl:item>
273            <itemID>R1</itemID>
274            <usdl:natureOfCargo>General cargo, Palletized
275            </usdl:natureOfCargo>
276            <description>spare parts</description>
277            <pieces>1.0</pieces>
278            <usdl:piecesUnit>europallet</usdl:piecesUnit>
279            <volume>1.0</volume>
280            <usdl:totalVolume>6.0</usdl:totalVolume>
281            <usdl:volumeUOM>m3</usdl:volumeUOM>
282            <usdl:totalWeight>600.0</usdl:totalWeight>
283            <usdl:weightUOM>kg</usdl:weightUOM>
284            <usdl:valueOfGoods>5546.0</usdl:valueOfGoods>
285            <usdl:currency>EURO</usdl:currency>
286            <regulatoryInformation>
```

```
287           http://www.spare-parts-dd.de/regInfo.pdf
288         </regulatoryInformation>
289       </usdl:item>
290
291       <usdl:item>
292         <itemID>R2</itemID>
293         <usdl:natureOfCargo>General cargo, Palletized
294         </usdl:natureOfCargo>
295         <description>spare parts</description>
296         <pieces>1.0</pieces>
297         <usdl:piecesUnit>europallet</usdl:piecesUnit>
298         <volume>1.0</volume>
299         <usdl:totalVolume>1.0</usdl:totalVolume>
300         <usdl:volumeUOM>cm3</usdl:volumeUOM>
301         <usdl:totalWeight>1000.0</usdl:totalWeight>
302         <usdl:weightUOM>kg</usdl:weightUOM>
303         <usdl:valueOfGoods>0.0</usdl:valueOfGoods>
304         <usdl:currency>USD</usdl:currency>
305         <regulatoryInformation></regulatoryInformation>
306       </usdl:item>
307
308       <usdl:item>
309         <itemID>R3</itemID>
310         <usdl:natureOfCargo>General cargo, Palletized
311         </usdl:natureOfCargo>
312         <description>spare parts</description>
313         <pieces>1.0</pieces>
314         <usdl:piecesUnit>europallet</usdl:piecesUnit>
315         <volume>1.0</volume>
316         <usdl:totalVolume>1.0</usdl:totalVolume>
317         <usdl:volumeUOM>cm3</usdl:volumeUOM>
318         <usdl:totalWeight>0.0</usdl:totalWeight>
319         <usdl:weightUOM>kg</usdl:weightUOM>
320         <usdl:valueOfGoods>0.0</usdl:valueOfGoods>
321         <usdl:currency>USD</usdl:currency>
322         <regulatoryInformation></regulatoryInformation>
323       </usdl:item>
324
325       <usdl:item>
326         <itemID>R4</itemID>
327         <usdl:natureOfCargo>General cargo, Palletized
328         </usdl:natureOfCargo>
329         <description>spare parts</description>
330         <pieces>1.0</pieces>
331         <usdl:piecesUnit>europallet</usdl:piecesUnit>
332         <volume>1.0</volume>
333         <usdl:totalVolume>1.0</usdl:totalVolume>
334         <usdl:volumeUOM>cm3</usdl:volumeUOM>
335         <usdl:totalWeight>0.0</usdl:totalWeight>
```

```
336           <usdl:weightUOM>kg</usdl:weightUOM>
337           <usdl:valueOfGoods>0.0</usdl:valueOfGoods>
338           <usdl:currency>USD</usdl:currency>
339           <regulatoryInformation></regulatoryInformation>
340         </usdl:item>
341
342         <usdl:item>
343           <itemID>R5</itemID>
344           <usdl:natureOfCargo>General cargo, Palletized
345           </usdl:natureOfCargo>
346           <description>spare parts</description>
347           <pieces>1.0</pieces>
348           <usdl:piecesUnit>europallet</usdl:piecesUnit>
349           <volume>1.0</volume>
350           <usdl:totalVolume>1.0</usdl:totalVolume>
351           <usdl:volumeUOM>cm3</usdl:volumeUOM>
352           <usdl:totalWeight>0.0</usdl:totalWeight>
353           <usdl:weightUOM>kg</usdl:weightUOM>
354           <usdl:valueOfGoods>0.0</usdl:valueOfGoods>
355           <usdl:currency>USD</usdl:currency>
356           <regulatoryInformation></regulatoryInformation>
357         </usdl:item>
358
359         <usdl:item>
360           <itemID>R6</itemID>
361           <usdl:natureOfCargo>General cargo, Palletized
362           </usdl:natureOfCargo>
363           <description>spare parts</description>
364           <pieces>1.0</pieces>
365           <usdl:piecesUnit>europallet</usdl:piecesUnit>
366           <volume>1.0</volume>
367           <usdl:totalVolume>1.0</usdl:totalVolume>
368           <usdl:volumeUOM>cm3</usdl:volumeUOM>
369           <usdl:totalWeight>0.0</usdl:totalWeight>
370           <usdl:weightUOM>kg</usdl:weightUOM>
371           <usdl:valueOfGoods>0.0</usdl:valueOfGoods>
372           <usdl:currency>USD</usdl:currency>
373           <regulatoryInformation></regulatoryInformation>
374         </usdl:item>
375       </items>
376     </usdl:productData>
377   </transportService>
378  </usdl:logisticsService>
379 </texoag:ServiceDescriptionTerm>
380
381 <!-- WSAG-Service_Reference Section-->
382 <texoag:ServiceReference ServiceName="OTMLTruckDE"
383  Name="OTMLTruckDEReference" />
384
```

```
385   <!-- WSAG-Service_Properties Section-->
386   <texoag:ServiceProperties ServiceName="OTMLTruckDE"
387    Name="OTMLTruckDEProperties">
388    <wsag:VariableSet>
389     <wsag:Variable Name="temperature" Metric="°C">
390      <wsag:Location />
391     </wsag:Variable>
392    </wsag:VariableSet>
393   </texoag:ServiceProperties>
394
395   <!-- WSAG Terms_Guarantee Section -->
396   <!-- WSAG SLOs from USDL ServiceLevelObjectives -->
397   <texoag:GuaranteeTerm Obligated="ServiceProvider"
398    Name="temperature_Guarantee">
399    <texoag:ServiceLevelObjective SLOID="
         SLOID_temperature_Guarantee" renegotiatable="true"
         >
400     <monitoringURI></monitoringURI>
401     <wsag:KPITarget>
402      <wsag:KPIName>temperature</wsag:KPIName>
403      <wsag:Target xsi:type="texoag:SimpleTarget">
404       <texoag:operator>lte</texoag:operator>
405       <texoag:value xsi:type="xs:double">15.0</
             texoag:value>
406      </wsag:Target>
407     </wsag:KPITarget>
408    </texoag:ServiceLevelObjective>
409   </texoag:GuaranteeTerm>
410
411   <texoag:GuaranteeTerm Obligated="ServiceProvider"
412    Name="OTMLTruckDE_startLocation_Guarantee">
413    <texoag:ServiceLevelObjective SLOID="
         SLOID_OTMLTruckDE_startLocation" renegotiatable="
         true">
414     <wsag:KPITarget>
415      <wsag:KPIName>startLocation</wsag:KPIName>
416      <wsag:Target xsi:type="texoag:SimpleTarget">
417       <texoag:operator>eq</texoag:operator>
418       <texoag:value xsi:type="xs:string">Bahnhofstr. 85,
419        01269 Dresden, Saxony, Germany</texoag:value>
420      </wsag:Target>
421     </wsag:KPITarget>
422    </texoag:ServiceLevelObjective>
423   </texoag:GuaranteeTerm>
424
425   <texoag:GuaranteeTerm Obligated="ServiceProvider"
426    Name="OTMLTruckDE_startTime_Guarantee">
```

```
427     <texoag:ServiceLevelObjective SLOID="
            SLOID_OTMLTruckDE_startTime" renegotiatable="true"
            >
428      <wsag:KPITarget>
429       <wsag:KPIName>startTime</wsag:KPIName>
430       <wsag:Target xsi:type="texoag:SimpleTarget">
431        <texoag:operator>eq</texoag:operator>
432        <texoag:value xsi:type="xs:string">
433         2009-11-10T06:00:00+0100/2009-11-10T08:00:00+0100
              </texoag:value>
434       </wsag:Target>
435      </wsag:KPITarget>
436     </texoag:ServiceLevelObjective>
437    </texoag:GuaranteeTerm>
438
439    <texoag:GuaranteeTerm Obligated="ServiceConsumer"
440     Name="OTMLTruckDE_inputResource_R1_Guarantee">
441     <texoag:ServiceLevelObjective
442      SLOID="SLOID_OTMLTruckDE_inputResource_R1"
           renegotiatable="true">
443      <wsag:KPITarget>
444       <wsag:KPIName>inputResource_R1
445       </wsag:KPIName>
446       <wsag:Target xsi:type="texoag:SimpleTarget">
447        <texoag:operator>eq</texoag:operator>
448        <texoag:value xsi:type="usdl:PortType">
449         <usdl:resourceType>Auto spare parts</
              usdl:resourceType>
450         <usdl:resourceName></usdl:resourceName>
451         <resourceID>R1</resourceID>
452        </texoag:value>
453       </wsag:Target>
454      </wsag:KPITarget>
455     </texoag:ServiceLevelObjective>
456    </texoag:GuaranteeTerm>
457
458    <texoag:GuaranteeTerm Obligated="ServiceConsumer"
459     Name="OTMLTruckDE_inputResource_R2_Guarantee">
460     <texoag:ServiceLevelObjective
461      SLOID="SLOID_OTMLTruckDE_inputResource_R2"
           renegotiatable="true">
462      <wsag:KPITarget>
463       <wsag:KPIName>inputResource_R2
464       </wsag:KPIName>
465       <wsag:Target xsi:type="texoag:SimpleTarget">
466        <texoag:operator>eq</texoag:operator>
467        <texoag:value xsi:type="usdl:PortType">
468         <usdl:resourceType>Auto spare parts</
              usdl:resourceType>
```

```
469          <usdl:resourceName></usdl:resourceName>
470          <resourceID>R2</resourceID>
471         </texoag:value>
472        </wsag:Target>
473       </wsag:KPITarget>
474      </texoag:ServiceLevelObjective>
475     </texoag:GuaranteeTerm>
476
477     <texoag:GuaranteeTerm Obligated="ServiceConsumer"
478      Name="OTMLTruckDE_inputResource_R3_Guarantee">
479      <texoag:ServiceLevelObjective
480       SLOID="SLOID_OTMLTruckDE_inputResource_R3"
             renegotiatable="true">
481       <wsag:KPITarget>
482        <wsag:KPIName>inputResource_R3
483        </wsag:KPIName>
484        <wsag:Target xsi:type="texoag:SimpleTarget">
485         <texoag:operator>eq</texoag:operator>
486         <texoag:value xsi:type="usdl:PortType">
487          <usdl:resourceType>Auto spare parts</
                 usdl:resourceType>
488          <usdl:resourceName></usdl:resourceName>
489          <resourceID>R3</resourceID>
490         </texoag:value>
491        </wsag:Target>
492       </wsag:KPITarget>
493      </texoag:ServiceLevelObjective>
494     </texoag:GuaranteeTerm>
495
496     <texoag:GuaranteeTerm Obligated="ServiceConsumer"
497      Name="OTMLTruckDE_inputResource_R4_Guarantee">
498      <texoag:ServiceLevelObjective
499       SLOID="SLOID_OTMLTruckDE_inputResource_R4"
             renegotiatable="true">
500       <wsag:KPITarget>
501        <wsag:KPIName>inputResource_R4
502        </wsag:KPIName>
503        <wsag:Target xsi:type="texoag:SimpleTarget">
504         <texoag:operator>eq</texoag:operator>
505         <texoag:value xsi:type="usdl:PortType">
506          <usdl:resourceType>Auto spare parts</
                 usdl:resourceType>
507          <usdl:resourceName></usdl:resourceName>
508          <resourceID>R4</resourceID>
509         </texoag:value>
510        </wsag:Target>
511       </wsag:KPITarget>
512      </texoag:ServiceLevelObjective>
513     </texoag:GuaranteeTerm>
```

```
514
515   <texoag:GuaranteeTerm Obligated="ServiceConsumer"
516    Name="OTMLTruckDE_inputResource_R5_Guarantee">
517    <texoag:ServiceLevelObjective
518     SLOID="SLOID_OTMLTruckDE_inputResource_R5"
          renegotiatable="true">
519     <wsag:KPITarget>
520      <wsag:KPIName>inputResource_R5
521      </wsag:KPIName>
522      <wsag:Target xsi:type="texoag:SimpleTarget">
523       <texoag:operator>eq</texoag:operator>
524       <texoag:value xsi:type="usdl:PortType">
525        <usdl:resourceType>Auto spare parts</
              usdl:resourceType>
526        <usdl:resourceName></usdl:resourceName>
527        <resourceID>R5</resourceID>
528       </texoag:value>
529      </wsag:Target>
530     </wsag:KPITarget>
531    </texoag:ServiceLevelObjective>
532   </texoag:GuaranteeTerm>
533
534   <texoag:GuaranteeTerm Obligated="ServiceConsumer"
535    Name="OTMLTruckDE_inputResource_R6_Guarantee">
536    <texoag:ServiceLevelObjective
537     SLOID="SLOID_OTMLTruckDE_inputResource_R6"
          renegotiatable="true">
538     <wsag:KPITarget>
539      <wsag:KPIName>inputResource_R6
540      </wsag:KPIName>
541      <wsag:Target xsi:type="texoag:SimpleTarget">
542       <texoag:operator>eq</texoag:operator>
543       <texoag:value xsi:type="usdl:PortType">
544        <usdl:resourceType>Auto spare parts</
              usdl:resourceType>
545        <usdl:resourceName></usdl:resourceName>
546        <resourceID>R6</resourceID>
547       </texoag:value>
548      </wsag:Target>
549     </wsag:KPITarget>
550    </texoag:ServiceLevelObjective>
551   </texoag:GuaranteeTerm>
552
553   <texoag:GuaranteeTerm Obligated="ServiceProvider"
554    Name="OTMLTruckDE_endLocation_Guarantee">
555    <texoag:ServiceLevelObjective SLOID="
          SLOID_OTMLTruckDE_endLocation" renegotiatable="
          true">
556     <wsag:KPITarget>
```

```
557       <wsag:KPIName>endLocation</wsag:KPIName>
558       <wsag:Target xsi:type="texoag:SimpleTarget">
559        <texoag:operator>eq</texoag:operator>
560        <texoag:value xsi:type="xs:string">Gutestr. 76,
               20095
561         Hamburg, Germany</texoag:value>
562       </wsag:Target>
563      </wsag:KPITarget>
564     </texoag:ServiceLevelObjective>
565    </texoag:GuaranteeTerm>
566
567    <texoag:GuaranteeTerm Obligated="ServiceProvider"
568     Name="OTMLTruckDE_endTime_Guarantee">
569     <texoag:ServiceLevelObjective SLOID="
           SLOID_OTMLTruckDE_endTime" renegotiatable="true">
570      <wsag:KPITarget>
571       <wsag:KPIName>endTime</wsag:KPIName>
572       <wsag:Target xsi:type="texoag:SimpleTarget">
573        <texoag:operator>eq</texoag:operator>
574        <texoag:value xsi:type="xs:string">
575         2009-11-10T12:00:00+0100/2009-11-10T14:00:00+0100
               </texoag:value>
576       </wsag:Target>
577      </wsag:KPITarget>
578     </texoag:ServiceLevelObjective>
579    </texoag:GuaranteeTerm>
580
581    <texoag:GuaranteeTerm Obligated="ServiceProvider"
582     Name="OTMLTruckDE_outputResource_R1_Guarantee">
583     <texoag:ServiceLevelObjective
584      SLOID="SLOID_OTMLTruckDE_outputResource_R1"
           renegotiatable="false">
585      <wsag:KPITarget>
586       <wsag:KPIName>outputResource_R1
587       </wsag:KPIName>
588       <wsag:Target xsi:type="texoag:SimpleTarget">
589        <texoag:operator>eq</texoag:operator>
590        <texoag:value xsi:type="usdl:PortType">
591         <usdl:resourceType>Auto spare parts</
               usdl:resourceType>
592         <usdl:resourceName></usdl:resourceName>
593         <resourceID>R1</resourceID>
594        </texoag:value>
595       </wsag:Target>
596      </wsag:KPITarget>
597     </texoag:ServiceLevelObjective>
598    </texoag:GuaranteeTerm>
599
600    <texoag:GuaranteeTerm Obligated="ServiceProvider"
```

```
601      Name="OTMLTruckDE_outputResource_R2_Guarantee">
602      <texoag:ServiceLevelObjective
603       SLOID="SLOID_OTMLTruckDE_outputResource_R2"
              renegotiatable="false">
604       <wsag:KPITarget>
605        <wsag:KPIName>outputResource_R2
606        </wsag:KPIName>
607        <wsag:Target xsi:type="texoag:SimpleTarget">
608         <texoag:operator>eq</texoag:operator>
609         <texoag:value xsi:type="usdl:PortType">
610          <usdl:resourceType>Auto spare parts</
                 usdl:resourceType>
611          <usdl:resourceName></usdl:resourceName>
612          <resourceID>R2</resourceID>
613         </texoag:value>
614        </wsag:Target>
615       </wsag:KPITarget>
616      </texoag:ServiceLevelObjective>
617     </texoag:GuaranteeTerm>
618
619     <texoag:GuaranteeTerm Obligated="ServiceProvider"
620      Name="OTMLTruckDE_outputResource_R3_Guarantee"
             renegotiatable="false">
621      <texoag:ServiceLevelObjective
622       SLOID="SLOID_OTMLTruckDE_outputResource_R3">
623       <wsag:KPITarget>
624        <wsag:KPIName>outputResource_R3
625        </wsag:KPIName>
626        <wsag:Target xsi:type="texoag:SimpleTarget">
627         <texoag:operator>eq</texoag:operator>
628         <texoag:value xsi:type="usdl:PortType">
629          <usdl:resourceType>Auto spare parts</
                 usdl:resourceType>
630          <usdl:resourceName></usdl:resourceName>
631          <resourceID>R3</resourceID>
632         </texoag:value>
633        </wsag:Target>
634       </wsag:KPITarget>
635      </texoag:ServiceLevelObjective>
636     </texoag:GuaranteeTerm>
637
638     <texoag:GuaranteeTerm Obligated="ServiceProvider"
639      Name="OTMLTruckDE_outputResource_R4_Guarantee">
640      <texoag:ServiceLevelObjective
641       SLOID="SLOID_OTMLTruckDE_outputResource_R4"
              renegotiatable="false">
642       <wsag:KPITarget>
643        <wsag:KPIName>outputResource_R4
644        </wsag:KPIName>
```

```
645       <wsag:Target xsi:type="texoag:SimpleTarget">
646        <texoag:operator>eq</texoag:operator>
647        <texoag:value xsi:type="usdl:PortType">
648         <usdl:resourceType>Auto spare parts</
               usdl:resourceType>
649         <usdl:resourceName></usdl:resourceName>
650         <resourceID>R4</resourceID>
651        </texoag:value>
652       </wsag:Target>
653      </wsag:KPITarget>
654     </texoag:ServiceLevelObjective>
655    </texoag:GuaranteeTerm>
656
657    <texoag:GuaranteeTerm Obligated="ServiceProvider"
658     Name="OTMLTruckDE_outputResource_R5_Guarantee">
659     <texoag:ServiceLevelObjective
660      SLOID="SLOID_OTMLTruckDE_outputResource_R5"
           renegotiatable="false">
661      <wsag:KPITarget>
662       <wsag:KPIName>outputResource_R5
663       </wsag:KPIName>
664       <wsag:Target xsi:type="texoag:SimpleTarget">
665        <texoag:operator>eq</texoag:operator>
666        <texoag:value xsi:type="usdl:PortType">
667         <usdl:resourceType>Auto spare parts</
               usdl:resourceType>
668         <usdl:resourceName></usdl:resourceName>
669         <resourceID>R5</resourceID>
670        </texoag:value>
671       </wsag:Target>
672      </wsag:KPITarget>
673     </texoag:ServiceLevelObjective>
674    </texoag:GuaranteeTerm>
675
676    <texoag:GuaranteeTerm Obligated="ServiceProvider"
677     Name="OTMLTruckDE_outputResource_R6_Guarantee"
          renegotiatable="false">
678     <texoag:ServiceLevelObjective
679      SLOID="SLOID_OTMLTruckDE_outputResource_R6">
680      <wsag:KPITarget>
681       <wsag:KPIName>outputResource_R6
682       </wsag:KPIName>
683       <wsag:Target xsi:type="texoag:SimpleTarget">
684        <texoag:operator>eq</texoag:operator>
685        <texoag:value xsi:type="usdl:PortType">
686         <usdl:resourceType>Auto spare parts</
               usdl:resourceType>
687         <usdl:resourceName></usdl:resourceName>
688         <resourceID>R6</resourceID>
```

```
689          </texoag:value>
690         </wsag:Target>
691        </wsag:KPITarget>
692       </texoag:ServiceLevelObjective>
693     </texoag:GuaranteeTerm>
694    </texoag:All>
695  </texoag:Terms>
696  <texoag:CreationConstraints />
697 </texoag:Template>
```

Listing B.1: SLA template document for logistics service

C

Logistics Scenario Evaluation Documents

Workflow paths

Based on the workflow of the logistics scenario, the following paths were created:

Path 1: AutoTrans DD – WarehouseDD – Truck DE – Warehouse HH – HH Local – Receive Goods – Prepare Shipment – Export Handling – Security Check – Ship Goods – Sea Shipping DE-US – OTML Warehouse NYC – Eastern Truck US

Path 2: AutoTrans DD – WarehouseDD – Truck DE – Warehouse HH – HH Star Truck – Receive Goods – Prepare Shipment – Export Handling – Security Check – Ship Goods – Sea Shipping DE-US – OTML Warehouse NYC – Eastern Truck US

Path 3: AutoTrans DD – WarehouseDD – Truck DE – Warehouse HH – Truck HH – Receive Goods – Prepare Shipment – Export Handling – Security Check – Ship Goods – Sea Shipping DE-US – OTML Warehouse NYC – Eastern Truck US

Path 4: Truck DD – WarehouseDD – Truck DE – Warehouse HH – HH Local – Receive Goods – Prepare Shipment – Export Handling – Security Check – Ship Goods – Sea Shipping DE-US – OTML Warehouse NYC – Eastern Truck US

Path 5: Truck DD – WarehouseDD – Truck DE – Warehouse HH – HH Star Truck – Receive Goods – Prepare Shipment – Export Handling – Security Check – Ship Goods – Sea Shipping DE-US – OTML Warehouse NYC – Eastern Truck US

Path 6: Truck DD – WarehouseDD – Truck DE – Warehouse HH – Truck HH – Receive Goods – Prepare Shipment – Export Handling – Security Check – Ship Goods – Sea Shipping DE-US – OTML Warehouse NYC – Eastern Truck US

Dependency model

```
1 <?xml version="1.0" encoding="ASCII"?>
2 <depend:DependencyModel xmi:version="2.0" xmlns:xmi="
      http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org
      /2001/XMLSchema-instance" xmlns:depend="net.texo.ise/
      dependency" xmlns:resourceDependency="net.texo.ise/
      depencency.resource" xmlns:timeDependency="net.texo.
      ise/depencency.time" id="DEP-MODEL_1274348438937"
      compositeServiceKey="OTMLTransWorld_H378G8329-J378-9
      U47-2ZU3-U378ZE8329EU" compositeServiceAgreementId="
      OTMLTransWorld_H378G8329-J378-9U47-2ZU3-
      U378ZE8329EU_SLA17">
3   <hasServiceEntities serviceName="OTML␣TransWorld"
        serviceKey="OTMLTransWorld_H378G8329-J378-9U47-2ZU3-
        U378ZE8329EU" provider="Z283JD392-F738-8Z83-23JE-374
        ZE278ZUE23" slaId="OTMLTransWorld_H378G8329-J378-9
        U47-2ZU3-U378ZE8329EU_SLA17" isDependent="//
        @hasDependencies.19␣//@hasDependencies.38"
        isAntecedent="//@hasDependencies.17␣//
        @hasDependencies.18␣//@hasDependencies.37␣//
        @hasDependencies.39"/>
4   <hasServiceEntities serviceName="AutoTrans␣DD"
        serviceKey="AutoTrans␣DD_Z3734ZE8-H372-5Z83-23JE
        -4849H3892G32" provider="D0R7G6U9-F738-8Z83-23JE-45
        G738625B64" slaId="AutoTrans␣DD_Z3734ZE8-H372-5Z83
        -23JE-4849H3892G32_SLA6" isDependent="//
        @hasDependencies.17␣//@hasDependencies.37"
        isAntecedent="//@hasDependencies.0␣//
        @hasDependencies.20"/>
5   <hasServiceEntities serviceName="Warehouse␣DD"
        serviceKey="Warehouse␣Dresden_G782T293-U287-IE828-
        K738H-341ER278RTW23" provider="H8ZSH8Z6-J782-U73H8
        -53JE-341ER278RTW23" slaId="Warehouse␣
        Dresden_G782T293-U287-IE828-K738H-341
        ER278RTW23_SLA15" isDependent="//@hasDependencies.0␣
        //@hasDependencies.16␣//@hasDependencies.20␣//
```

```
        @hasDependencies.36" isAntecedent="//
        @hasDependencies.1␣//@hasDependencies.21"/>
6   <hasServiceEntities serviceName="Truck␣DE" serviceKey="
        OTML␣Truck␣DE_H738T378-U323-83H3-83GZ-839U348E8329"
        provider="Z283JD392-F738-8Z83-23JE-374ZE278ZUE23"
        slaId="OTML␣Truck␣DE_H738T378-U323-83H3-83GZ-839
        U348E8329_SLA9" isDependent="//@hasDependencies.1␣//
        @hasDependencies.21" isAntecedent="//
        @hasDependencies.2␣//@hasDependencies.22"/>
7   <hasServiceEntities serviceName="Warehouse␣HH"
        serviceKey="OTML␣Warehouse␣Hamburg_H837Z3782-I839-3
        Z734-Z378-374ZE278ZUE23" provider="Z283JD392-F738-8
        Z83-23JE-374ZE278ZUE23" slaId="OTML␣Warehouse␣
        Hamburg_H837Z3782-I839-3Z734-Z378-374
        ZE278ZUE23_SLA16" isDependent="//@hasDependencies.2␣
        //@hasDependencies.22" isAntecedent="//
        @hasDependencies.3␣//@hasDependencies.12␣//
        @hasDependencies.14␣//@hasDependencies.23␣//
        @hasDependencies.32␣//@hasDependencies.34"/>
8   <hasServiceEntities serviceName="HH␣Local" serviceKey="
        HH␣Local_G637Z337-J3784-I384-H373-74Z392N328"
        provider="D323G344-H324-64ZH-84Z3-374Z392N328" slaId
        ="HH␣Local_G637Z337-J3784-I384-H373-74Z392N328_SLA11
        " isDependent="//@hasDependencies.3␣//
        @hasDependencies.23" isAntecedent="//
        @hasDependencies.4␣//@hasDependencies.24"/>
9   <hasServiceEntities serviceName="Receive␣Goods"
        serviceKey="WarehousePortHH_ReceiveGoods_RG347-221PJ
        " provider="" slaId="
        WarehousePortHH_ReceiveGoods_RG347-221PJ_SLA14"
        isDependent="//@hasDependencies.4␣//@hasDependencies
        .13␣//@hasDependencies.15␣//@hasDependencies.24␣//
        @hasDependencies.33␣//@hasDependencies.35"
        isAntecedent="//@hasDependencies.5␣//
        @hasDependencies.25"/>
10  <hasServiceEntities serviceName="Prepare␣Shipment"
        serviceKey="WarehousePortHH_PrepareShipment_PS347
        -221PJ" provider="" slaId="
        WarehousePortHH_PrepareShipment_PS347-221PJ_SLA3"
        isDependent="//@hasDependencies.5␣//@hasDependencies
        .25" isAntecedent="//@hasDependencies.6␣//
        @hasDependencies.26"/>
11  <hasServiceEntities serviceName="Export␣Handling"
        serviceKey="WarehousePortHH_ExportHandling_EH347-221
        PJ" provider="" slaId="
        WarehousePortHH_ExportHandling_EH347-221PJ_SLA12"
        isDependent="//@hasDependencies.6␣//@hasDependencies
        .26" isAntecedent="//@hasDependencies.7␣//
        @hasDependencies.27"/>
```

```
12  <hasServiceEntities serviceName="Security␣Check"
        serviceKey="WarehousePortHH_SecurityCheck_SC347-221
        PJ" provider="" slaId="
        WarehousePortHH_SecurityCheck_SC347-221PJ_SLA5"
        isDependent="//@hasDependencies.7␣//@hasDependencies
        .27" isAntecedent="//@hasDependencies.8␣//
        @hasDependencies.28"/>
13  <hasServiceEntities serviceName="Ship␣Goods" serviceKey
        ="WarehousePortHH_ShipGoods_SG347-221PJ" provider=""
         slaId="WarehousePortHH_ShipGoods_SG347-221PJ_SLA10"
         isDependent="//@hasDependencies.8␣//
        @hasDependencies.28" isAntecedent="//
        @hasDependencies.9␣//@hasDependencies.29"/>
14  <hasServiceEntities serviceName="Sea␣Shipping␣DE-US"
        serviceKey="SeaShippingDE-US_T736H367-F738-7G37-83N4
        -748H349V3428" provider="U473J482-F738-7G37-83N4-748
        H349V3428" slaId="SeaShippingDE-US_T736H367-F738-7
        G37-83N4-748H349V3428_SLA8" isDependent="//
        @hasDependencies.9␣//@hasDependencies.29"
        isAntecedent="//@hasDependencies.10␣//
        @hasDependencies.30"/>
15  <hasServiceEntities serviceName="OTML␣Warehouse␣NYC"
        serviceKey="OTMLWarehouseNYC_L7838A748-F738-8Z83-23
        JE-374ZE278ZUE23" provider="Z283JD392-F738-8Z83-23JE
        -374ZE278ZUE23" slaId="OTMLWarehouseNYC_L7838A748-
        F738-8Z83-23JE-374ZE278ZUE23_SLA4" isDependent="//
        @hasDependencies.10␣//@hasDependencies.30"
        isAntecedent="//@hasDependencies.11␣//
        @hasDependencies.31"/>
16  <hasServiceEntities serviceName="Eastern␣Truck␣US"
        serviceKey="Eastern␣Truck␣US_H0T34822-U374-84Z7-23JE
        -45G738625B64" provider="T6Z34822-U374-8H38-23JE-45
        H733625B64" slaId="Eastern␣Truck␣US_H0T34822-U374-84
        Z7-23JE-45G738625B64_SLA1" isDependent="//
        @hasDependencies.11␣//@hasDependencies.31"
        isAntecedent="//@hasDependencies.19␣//
        @hasDependencies.38"/>
17  <hasServiceEntities serviceName="HH␣Star␣Truck"
        serviceKey="HH␣Star␣Truck_H378U378-J378-K329-U384-
        H273WU324" provider="U378H834-N393-8H473-H382-
        H273WU324" slaId="HH␣Star␣Truck_H378U378-J378-K329-
        U384-H273WU324_SLA2" isDependent="//@hasDependencies
        .12␣//@hasDependencies.32" isAntecedent="//
        @hasDependencies.13␣//@hasDependencies.33"/>
18  <hasServiceEntities serviceName="Truck␣HH" serviceKey="
        Truck␣HH_T637Z378-H374-9H73-67ZE-374ZE278ZUE23"
        provider="H637Z3942-U238-7W63-73ZE-374ZE278ZUE23"
        slaId="Truck␣HH_T637Z378-H374-9H73-67ZE-374
        ZE278ZUE23_SLA13" isDependent="//@hasDependencies.14
```

```
        ␣//@hasDependencies.34" isAntecedent="//
        @hasDependencies.15␣//@hasDependencies.35"/>
19    <hasServiceEntities serviceName="Truck␣DD" serviceKey="
        Truck␣DD_G273Z389-U2833-J238G3-I493-341ER278RTW23"
        provider="H8ZSH8Z6-J782-U73H8-53JE-341ER278RTW23"
        slaId="Truck␣DD_G273Z389-U2833-J238G3-I493-341
        ER278RTW23_SLA7" isDependent="//@hasDependencies.18␣
        //@hasDependencies.39" isAntecedent="//
        @hasDependencies.16␣//@hasDependencies.36"/>
20    <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_4654816320257276500" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.2" antecedents="//
        @hasServiceEntities.1" isBidirectional="true"
        isTransitive="true">
21      <resourceIDList>R1</resourceIDList>
22      <resourceIDList>R2</resourceIDList>
23      <resourceIDList>R3</resourceIDList>
24    </hasDependencies>
25    <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_2809616772717469205" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.3" antecedents="//
        @hasServiceEntities.2" isBidirectional="true"
        isTransitive="true">
26      <resourceIDList>R1</resourceIDList>
27      <resourceIDList>R2</resourceIDList>
28      <resourceIDList>R3</resourceIDList>
29      <resourceIDList>R4</resourceIDList>
30      <resourceIDList>R5</resourceIDList>
31      <resourceIDList>R6</resourceIDList>
32    </hasDependencies>
33    <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_6890171781742065052" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.4" antecedents="//
        @hasServiceEntities.3" isBidirectional="true"
        isTransitive="true">
34      <resourceIDList>R1</resourceIDList>
35      <resourceIDList>R2</resourceIDList>
36      <resourceIDList>R3</resourceIDList>
37      <resourceIDList>R4</resourceIDList>
38      <resourceIDList>R5</resourceIDList>
39      <resourceIDList>R6</resourceIDList>
40    </hasDependencies>
```

```
41  <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_8054735613852779508" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.5" antecedents="//
        @hasServiceEntities.4" isBidirectional="true"
        isTransitive="true">
42    <resourceIDList>R5</resourceIDList>
43    <resourceIDList>R6</resourceIDList>
44  </hasDependencies>
45  <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_8843868546912478521" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.6" antecedents="//
        @hasServiceEntities.5" isBidirectional="true"
        isTransitive="true">
46    <resourceIDList>R5</resourceIDList>
47    <resourceIDList>R6</resourceIDList>
48  </hasDependencies>
49  <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_7457589716723400975" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.7" antecedents="//
        @hasServiceEntities.6" isBidirectional="true"
        isTransitive="true">
50    <resourceIDList>R1</resourceIDList>
51    <resourceIDList>R2</resourceIDList>
52    <resourceIDList>R3</resourceIDList>
53    <resourceIDList>R4</resourceIDList>
54    <resourceIDList>R5</resourceIDList>
55    <resourceIDList>R6</resourceIDList>
56  </hasDependencies>
57  <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_908671097329286228" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.8" antecedents="//
        @hasServiceEntities.7" isBidirectional="true"
        isTransitive="true">
58    <resourceIDList>R1</resourceIDList>
59    <resourceIDList>R2</resourceIDList>
60    <resourceIDList>R3</resourceIDList>
61    <resourceIDList>R4</resourceIDList>
62    <resourceIDList>R5</resourceIDList>
63    <resourceIDList>R6</resourceIDList>
64  </hasDependencies>
```

```
65  <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_1156137052082001816" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.9" antecedents="//
        @hasServiceEntities.8" isBidirectional="true"
        isTransitive="true">
66    <resourceIDList>R1</resourceIDList>
67    <resourceIDList>R2</resourceIDList>
68    <resourceIDList>R3</resourceIDList>
69    <resourceIDList>R4</resourceIDList>
70    <resourceIDList>R5</resourceIDList>
71    <resourceIDList>R6</resourceIDList>
72  </hasDependencies>
73  <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_5947715936416711726" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.10" antecedents="//
        @hasServiceEntities.9" isBidirectional="true"
        isTransitive="true">
74    <resourceIDList>R1</resourceIDList>
75    <resourceIDList>R2</resourceIDList>
76    <resourceIDList>R3</resourceIDList>
77    <resourceIDList>R4</resourceIDList>
78    <resourceIDList>R5</resourceIDList>
79    <resourceIDList>R6</resourceIDList>
80  </hasDependencies>
81  <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_7864414597975473277" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.11" antecedents="//
        @hasServiceEntities.10" isBidirectional="true"
        isTransitive="true">
82    <resourceIDList>R1</resourceIDList>
83    <resourceIDList>R2</resourceIDList>
84    <resourceIDList>R3</resourceIDList>
85    <resourceIDList>R4</resourceIDList>
86    <resourceIDList>R5</resourceIDList>
87    <resourceIDList>R6</resourceIDList>
88  </hasDependencies>
89  <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_1837412450888701281" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.12" antecedents="//
        @hasServiceEntities.11" isBidirectional="true"
        isTransitive="true">
```

```
90      <resourceIDList>R1</resourceIDList>
91      <resourceIDList>R2</resourceIDList>
92      <resourceIDList>R3</resourceIDList>
93      <resourceIDList>R4</resourceIDList>
94      <resourceIDList>R5</resourceIDList>
95      <resourceIDList>R6</resourceIDList>
96    </hasDependencies>
97    <hasDependencies xsi:type="
          resourceDependency:ResourceDependency" id="
          DEP_4208444788860258083" dependentKpiName="
          inputResource" antecedentKpiName="outputResource"
          dependent="//@hasServiceEntities.13" antecedents="//
          @hasServiceEntities.12" isBidirectional="true"
          isTransitive="true">
98      <resourceIDList>R1</resourceIDList>
99      <resourceIDList>R2</resourceIDList>
100     <resourceIDList>R3</resourceIDList>
101     <resourceIDList>R4</resourceIDList>
102     <resourceIDList>R5</resourceIDList>
103     <resourceIDList>R6</resourceIDList>
104   </hasDependencies>
105   <hasDependencies xsi:type="
          resourceDependency:ResourceDependency" id="
          DEP_560216919220316170" dependentKpiName="
          inputResource" antecedentKpiName="outputResource"
          dependent="//@hasServiceEntities.14" antecedents="//
          @hasServiceEntities.4" isBidirectional="true"
          isTransitive="true">
106     <resourceIDList>R3</resourceIDList>
107     <resourceIDList>R4</resourceIDList>
108   </hasDependencies>
109   <hasDependencies xsi:type="
          resourceDependency:ResourceDependency" id="
          DEP_6808086829702630139" dependentKpiName="
          inputResource" antecedentKpiName="outputResource"
          dependent="//@hasServiceEntities.6" antecedents="//
          @hasServiceEntities.14" isBidirectional="true"
          isTransitive="true">
110     <resourceIDList>R3</resourceIDList>
111     <resourceIDList>R4</resourceIDList>
112   </hasDependencies>
113   <hasDependencies xsi:type="
          resourceDependency:ResourceDependency" id="
          DEP_8589037997630980865" dependentKpiName="
          inputResource" antecedentKpiName="outputResource"
          dependent="//@hasServiceEntities.15" antecedents="//
          @hasServiceEntities.4" isBidirectional="true"
          isTransitive="true">
114     <resourceIDList>R1</resourceIDList>
```

```
115     <resourceIDList>R2</resourceIDList>
116   </hasDependencies>
117   <hasDependencies xsi:type="
          resourceDependency:ResourceDependency" id="
          DEP_5658000557955160558" dependentKpiName="
          inputResource" antecedentKpiName="outputResource"
          dependent="//@hasServiceEntities.6" antecedents="//
          @hasServiceEntities.15" isBidirectional="true"
          isTransitive="true">
118     <resourceIDList>R1</resourceIDList>
119     <resourceIDList>R2</resourceIDList>
120   </hasDependencies>
121   <hasDependencies xsi:type="
          resourceDependency:ResourceDependency" id="
          DEP_4550343590911930243" dependentKpiName="
          inputResource" antecedentKpiName="outputResource"
          dependent="//@hasServiceEntities.2" antecedents="//
          @hasServiceEntities.16" isBidirectional="true"
          isTransitive="true">
122     <resourceIDList>R4</resourceIDList>
123     <resourceIDList>R5</resourceIDList>
124     <resourceIDList>R6</resourceIDList>
125   </hasDependencies>
126   <hasDependencies xsi:type="
          resourceDependency:ResourceDependency" id="
          DEP_3644406465514665996" dependentKpiName="
          inputResource" antecedentKpiName="inputResource"
          dependent="//@hasServiceEntities.1" antecedents="//
          @hasServiceEntities.0" isBidirectional="true"
          isTransitive="true">
127     <resourceIDList>R1</resourceIDList>
128     <resourceIDList>R2</resourceIDList>
129     <resourceIDList>R3</resourceIDList>
130   </hasDependencies>
131   <hasDependencies xsi:type="
          resourceDependency:ResourceDependency" id="
          DEP_3420401336548117185" dependentKpiName="
          inputResource" antecedentKpiName="inputResource"
          dependent="//@hasServiceEntities.16" antecedents="//
          @hasServiceEntities.0" isBidirectional="true"
          isTransitive="true">
132     <resourceIDList>R4</resourceIDList>
133     <resourceIDList>R5</resourceIDList>
134     <resourceIDList>R6</resourceIDList>
135   </hasDependencies>
136   <hasDependencies xsi:type="
          resourceDependency:ResourceDependency" id="
          DEP_5921978982428827668" dependentKpiName="
          outputResource" antecedentKpiName="outputResource"
```

```
        dependent="//@hasServiceEntities.0" antecedents="//
        @hasServiceEntities.13" isBidirectional="true"
        isTransitive="true">
137       <resourceIDList>R1</resourceIDList>
138       <resourceIDList>R2</resourceIDList>
139       <resourceIDList>R3</resourceIDList>
140       <resourceIDList>R4</resourceIDList>
141       <resourceIDList>R5</resourceIDList>
142       <resourceIDList>R6</resourceIDList>
143     </hasDependencies>
144     <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_8683363751178791236" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.2" antecedents="//
        @hasServiceEntities.1" timeOperator="finish_to_start
        "/>
145     <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_5599405348357593374" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.3" antecedents="//
        @hasServiceEntities.2" timeOperator="finish_to_start
        "/>
146     <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_489445985970284373" dependentKpiName="startTime"
         antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.4" antecedents="//
        @hasServiceEntities.3" timeOperator="finish_to_start
        "/>
147     <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_5610421649882518205" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.5" antecedents="//
        @hasServiceEntities.4" timeOperator="finish_to_start
        "/>
148     <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_3085096819141771722" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.6" antecedents="//
        @hasServiceEntities.5" timeOperator="finish_to_start
        "/>
149     <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_1538628474424631540" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
```

```
        @hasServiceEntities.7" antecedents="//
        @hasServiceEntities.6" timeOperator="finish_to_start
        "/>
150     <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_9209396800649480176" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.8" antecedents="//
        @hasServiceEntities.7" timeOperator="finish_to_start
        "/>
151     <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_8890397700087597467" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.9" antecedents="//
        @hasServiceEntities.8" timeOperator="finish_to_start
        "/>
152     <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_6175222539544770722" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.10" antecedents="//
        @hasServiceEntities.9" timeOperator="finish_to_start
        "/>
153     <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_3054666569836456037" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.11" antecedents="//
        @hasServiceEntities.10" timeOperator="
        finish_to_start"/>
154     <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_322242023808470795" dependentKpiName="startTime"
         antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.12" antecedents="//
        @hasServiceEntities.11" timeOperator="
        finish_to_start"/>
155     <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_2258205153677280023" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.13" antecedents="//
        @hasServiceEntities.12" timeOperator="
        finish_to_start"/>
156     <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_4286004822163747665" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
```

```
        @hasServiceEntities.14" antecedents="//
        @hasServiceEntities.4" timeOperator="finish_to_start
        "/>
157     <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_7558257731186336001" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.6" antecedents="//
        @hasServiceEntities.14" timeOperator="
        finish_to_start"/>
158     <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_9030337593121072648" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.15" antecedents="//
        @hasServiceEntities.4" timeOperator="finish_to_start
        "/>
159     <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_4758728845319329828" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.6" antecedents="//
        @hasServiceEntities.15" timeOperator="
        finish_to_start"/>
160     <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_1786128740466725969" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.2" antecedents="//
        @hasServiceEntities.16" timeOperator="
        finish_to_start"/>
161     <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_8556964388459659605" dependentKpiName="startTime
        " antecedentKpiName="startTime" dependent="//
        @hasServiceEntities.1" antecedents="//
        @hasServiceEntities.0" timeOperator="start_to_start"
        />
162     <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_6321987276126315834" dependentKpiName="endTime"
        antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.0" antecedents="//
        @hasServiceEntities.13" timeOperator="
        finish_to_finish"/>
163     <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_7743259469322219250" dependentKpiName="startTime
        " antecedentKpiName="startTime" dependent="//
```

```
        @hasServiceEntities.16" antecedents="//
        @hasServiceEntities.0" timeOperator="start_to_start"
        />
164 </depend:DependencyModel>
```

Listing C.1: Dependency model for logistics service

D

Health Care Scenario Evaluation Documents

Workflow paths

Based on the workflow of the healthcare scenario, the following paths were created:

Path 1: Patient admission – Patient Data collection – Examine blood – Check examination results – Follow-up treatment determination – Discharge patient

Path 2: Patient admission – Patient Data collection – Examine blood – Check examination results – Follow-up treatment determination – Create Report

Path 3: Patient admission – Patient Data collection – Medical record creation – Patient transport – Expert examination – Patient transport – Check examination results – Follow-up treatment determination – Discharge patient

Path 4: Patient admission – Patient Data collection – Medical record creation – Patient transport – Expert examination – Patient transport – Check examination results – Follow-up treatment determination – Create Report

Path 5: Patient admission – Patient Data collection – Medical record creation – Determine medication – Procurement of medication – Give Medication – Check examination results – Follow-up treatment determination – Discharge patient

Path 6: Patient admission – Patient Data collection – Medical record creation – Determine medication – Procurement of medication – Give Medication – Check examination results – Follow-up treatment determination – Create Report

Path 7: Patient admission – Patient examination – Medical record creation – Patient transport – Expert examination – Patient transport – Check examination results – Follow-up treatment determination – Discharge patient

Path 8: Patient admission – Patient examination – Medical record creation – Patient transport – Expert examination – Patient transport – Check examination results – Follow-up treatment determination – Create Report

Path 9: Patient admission – Patient examination – Medical record creation – Determine medication – Procurement of medication – Give Medication – Check examination results – Follow-up treatment determination – Discharge patient

Path 10: Patient admission – Patient examination – Medical record creation – Determine medication – Procurement of medication – Give Medication – Check examination results – Follow-up treatment determination – Create Report

Dependency model

```
1 <?xml version="1.0" encoding="ASCII"?>
2 <depend:DependencyModel xmi:version="2.0" xmlns:xmi="
      http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org
      /2001/XMLSchema-instance" xmlns:depend="net.texo.ise/
      dependency" xmlns:resourceDependency="net.texo.ise/
      depencency.resource" xmlns:timeDependency="net.texo.
      ise/depencency.time" id="DEP-MODEL_1272351127562"
      compositeServiceKey="StationaryPatientCheckup_2345-
      SD234" compositeServiceAgreementId="
      StationaryPatientCheckup_2345-SD234_SLA16">
3   <hasServiceEntities serviceName="Stationary␣Patient␣
        Checkup" serviceKey="StationaryPatientCheckup_2345-
        SD234" provider="VC2235-F55J-PY51" slaId="
        StationaryPatientCheckup_2345-SD234_SLA16"
        isDependent="//@hasDependencies.19␣//
        @hasDependencies.38␣//@hasDependencies.39"
        isAntecedent="//@hasDependencies.18␣//
        @hasDependencies.37"/>
4   <hasServiceEntities serviceName="Patient␣Admission"
        serviceKey="PatientAdmission_98J5-RI57" provider=""
        slaId="PatientAdmission_98J5-RI57_SLA1" isDependent=
        "//@hasDependencies.18␣//@hasDependencies.37"
        isAntecedent="//@hasDependencies.0␣//
        @hasDependencies.1␣//@hasDependencies.6␣//
```

```
        @hasDependencies.12␣//@hasDependencies.16␣//
        @hasDependencies.20␣//@hasDependencies.35"/>
5   <hasServiceEntities serviceName="Patient␣Data␣
        Collection" serviceKey="PatientDataCollection_PDC7-7
        ZI9" provider="" slaId="PatientDataCollection_PDC7-7
        ZI9_SLA3" isDependent="//@hasDependencies.0␣//
        @hasDependencies.20" isAntecedent="//
        @hasDependencies.2␣//@hasDependencies.7␣//
        @hasDependencies.21␣//@hasDependencies.26"/>
6   <hasServiceEntities serviceName="Examine␣Blood"
        serviceKey="ExamineBlood_EB39-TT7P" provider=""
        slaId="ExamineBlood_EB39-TT7P_SLA14" isDependent="//
        @hasDependencies.1␣//@hasDependencies.2␣//
        @hasDependencies.21" isAntecedent="//
        @hasDependencies.3␣//@hasDependencies.22"/>
7   <hasServiceEntities serviceName="Check␣Examination␣
        Results" serviceKey="CheckExaminationResults_J876-
        PWL3" provider="" slaId="
        CheckExaminationResults_J876-PWL3_SLA13" isDependent
        ="//@hasDependencies.3␣//@hasDependencies.9␣//
        @hasDependencies.11␣//@hasDependencies.22␣//
        @hasDependencies.30␣//@hasDependencies.34"
        isAntecedent="//@hasDependencies.23"/>
8   <hasServiceEntities serviceName="Follow-up␣Treatment␣
        Determination" serviceKey="Follow-
        upTreatmentDetermination_FUTD-0815" provider=""
        slaId="Follow-upTreatmentDetermination_FUTD-0815
        _SLA12" isDependent="//@hasDependencies.10␣//
        @hasDependencies.23" isAntecedent="//
        @hasDependencies.4␣//@hasDependencies.5␣//
        @hasDependencies.24␣//@hasDependencies.25"/>
9   <hasServiceEntities serviceName="Discharge␣Patient"
        serviceKey="DischargePatient_DP08-55K9" provider="
        HCKP-15" slaId="DischargePatient_DP08-55K9_SLA6"
        isDependent="//@hasDependencies.4␣//@hasDependencies
        .24" isAntecedent="//@hasDependencies.38"/>
10  <hasServiceEntities serviceName="Create␣Report"
        serviceKey="ReportCreation_RC38-05L1" provider="VC1C
        -T51L-40OR" slaId="ReportCreation_RC38-05L1_SLA4"
        isDependent="//@hasDependencies.5␣//@hasDependencies
        .25" isAntecedent="//@hasDependencies.19␣//
        @hasDependencies.39"/>
11  <hasServiceEntities serviceName="Medical␣Record␣
        Creation" serviceKey="MedicalRecordCreation_MRC1
        -9128" provider="HCKP-8" slaId="
        MedicalRecordCreation_MRC1-9128_SLA8" isDependent="
        //@hasDependencies.6␣//@hasDependencies.7␣//
        @hasDependencies.17␣//@hasDependencies.26␣//
        @hasDependencies.36" isAntecedent="//
```

```
        @hasDependencies.8␣//@hasDependencies.9␣//
        @hasDependencies.10␣//@hasDependencies.13␣//
        @hasDependencies.27␣//@hasDependencies.31"/>
12  <hasServiceEntities serviceName="Expert␣Examination"
        serviceKey="ExpertExamination_EE57-ZV5Q" provider="
        HCKP-13" slaId="ExpertExamination_EE57-ZV5Q_SLA11"
        isDependent="//@hasDependencies.8␣//@hasDependencies
        .28" isAntecedent="//@hasDependencies.11␣//
        @hasDependencies.29"/>
13  <hasServiceEntities serviceName="Patient␣Transport"
        serviceKey="PatientTransport_PT96-JL7Y" provider="
        HCKP-3" slaId="PatientTransport_PT96-JL7Y_SLA7"
        isDependent="//@hasDependencies.27" isAntecedent="//
        @hasDependencies.28"/>
14  <hasServiceEntities serviceName="Patient␣Transport"
        serviceKey="PatientTransport_PT0M-GG9S" provider="
        HCKP-4" slaId="PatientTransport_PT0M-GG9S_SLA9"
        isDependent="//@hasDependencies.29" isAntecedent="//
        @hasDependencies.30"/>
15  <hasServiceEntities serviceName="Procurement␣of␣
        Medication" serviceKey="
        ProcurementOfMedication_POM4_90W1" provider="HCKP-2"
         slaId="ProcurementOfMedication_POM4_90W1_SLA5"
        isDependent="//@hasDependencies.12␣//
        @hasDependencies.14␣//@hasDependencies.32"
        isAntecedent="//@hasDependencies.15␣//
        @hasDependencies.33"/>
16  <hasServiceEntities serviceName="Determine␣Medication"
        serviceKey="DetermineMedication_DM98-1298" provider=
        "HCKP-16" slaId="DetermineMedication_DM98-1298_SLA15
        " isDependent="//@hasDependencies.13␣//
        @hasDependencies.31" isAntecedent="//
        @hasDependencies.14␣//@hasDependencies.32"/>
17  <hasServiceEntities serviceName="Give␣Medication"
        serviceKey="GiveMedication_GM73-668R" provider=""
        slaId="GiveMedication_GM73-668R_SLA2" isDependent="
        //@hasDependencies.15␣//@hasDependencies.33"
        isAntecedent="//@hasDependencies.34"/>
18  <hasServiceEntities serviceName="Patient␣Examination"
        serviceKey="PatientExamination_PE39-12ZX" provider="
        HCKP-5" slaId="PatientExamination_PE39-12ZX_SLA10"
        isDependent="//@hasDependencies.16␣//
        @hasDependencies.35" isAntecedent="//
        @hasDependencies.17␣//@hasDependencies.36"/>
19  <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_3122198522849526067" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.2" antecedents="//
```

```
        @hasServiceEntities.1" isBidirectional="true"
        isTransitive="true">
20    <resourceIDList>patientID</resourceIDList>
21  </hasDependencies>
22  <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_3235787490193518363" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.3" antecedents="//
        @hasServiceEntities.1" isBidirectional="true"
        isTransitive="true">
23    <resourceIDList>patientID</resourceIDList>
24  </hasDependencies>
25  <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_4726928565072001116" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.3" antecedents="//
        @hasServiceEntities.2" isBidirectional="true"
        isTransitive="true">
26    <resourceIDList>bloodSample</resourceIDList>
27  </hasDependencies>
28  <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_6838204256045487745" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.4" antecedents="//
        @hasServiceEntities.3" isBidirectional="true"
        isTransitive="true">
29    <resourceIDList>laboratoryTestResult</resourceIDList>
30  </hasDependencies>
31  <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_1988522268136441289" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.6" antecedents="//
        @hasServiceEntities.5" isBidirectional="true"
        isTransitive="true">
32    <resourceIDList>medicalRecord</resourceIDList>
33  </hasDependencies>
34  <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_8503263781163168326" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.7" antecedents="//
        @hasServiceEntities.5" isBidirectional="true"
        isTransitive="true">
35    <resourceIDList>medicalRecord</resourceIDList>
36  </hasDependencies>
```

```
37  <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_4488314381232341727" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.8" antecedents="//
        @hasServiceEntities.1" isBidirectional="true"
        isTransitive="true">
38    <resourceIDList>patientID</resourceIDList>
39  </hasDependencies>
40  <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_5758159868971947099" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.8" antecedents="//
        @hasServiceEntities.2" isBidirectional="true"
        isTransitive="true">
41    <resourceIDList>patientGeneralData</resourceIDList>
42  </hasDependencies>
43  <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_4571900140280303028" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.9" antecedents="//
        @hasServiceEntities.8" isBidirectional="true"
        isTransitive="true">
44    <resourceIDList>medicalRecord</resourceIDList>
45  </hasDependencies>
46  <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_109277824995689495" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.4" antecedents="//
        @hasServiceEntities.8" isBidirectional="true"
        isTransitive="true">
47    <resourceIDList>medicalRecord</resourceIDList>
48  </hasDependencies>
49  <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_7203788845200534650" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.5" antecedents="//
        @hasServiceEntities.8" isBidirectional="true"
        isTransitive="true">
50    <resourceIDList>medicalRecord</resourceIDList>
51  </hasDependencies>
52  <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_8445576661365936021" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
```

```
        dependent="//@hasServiceEntities.4" antecedents="//
        @hasServiceEntities.9" isBidirectional="true"
        isTransitive="true">
53      <resourceIDList>examinationResults</resourceIDList>
54    </hasDependencies>
55    <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_5692286028398201659" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.12" antecedents="//
        @hasServiceEntities.1" isBidirectional="true"
        isTransitive="true">
56      <resourceIDList>patientID</resourceIDList>
57    </hasDependencies>
58    <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_5768855607347483111" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.13" antecedents="//
        @hasServiceEntities.8" isBidirectional="true"
        isTransitive="true">
59      <resourceIDList>medicalRecord</resourceIDList>
60    </hasDependencies>
61    <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_7350673158800343343" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.12" antecedents="//
        @hasServiceEntities.13" isBidirectional="true"
        isTransitive="true">
62      <resourceIDList>medicationOrder</resourceIDList>
63    </hasDependencies>
64    <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_1878476987512499705" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.14" antecedents="//
        @hasServiceEntities.12" isBidirectional="true"
        isTransitive="true">
65      <resourceIDList>medication</resourceIDList>
66    </hasDependencies>
67    <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_7505131736359921750" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.15" antecedents="//
        @hasServiceEntities.1" isBidirectional="true"
        isTransitive="true">
68      <resourceIDList>patientID</resourceIDList>
```

```
69  </hasDependencies>
70  <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_8119398668251628684" dependentKpiName="
        inputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.8" antecedents="//
        @hasServiceEntities.15" isBidirectional="true"
        isTransitive="true">
71    <resourceIDList>healthCheckData</resourceIDList>
72  </hasDependencies>
73  <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_83379073609732100" dependentKpiName="
        inputResource" antecedentKpiName="inputResource"
        dependent="//@hasServiceEntities.1" antecedents="//
        @hasServiceEntities.0" isBidirectional="true"
        isTransitive="true">
74    <resourceIDList>billing information</resourceIDList>
75  </hasDependencies>
76  <hasDependencies xsi:type="
        resourceDependency:ResourceDependency" id="
        DEP_1483439213203111681" dependentKpiName="
        outputResource" antecedentKpiName="outputResource"
        dependent="//@hasServiceEntities.0" antecedents="//
        @hasServiceEntities.7" isBidirectional="true"
        isTransitive="true">
77    <resourceIDList>examinationReport</resourceIDList>
78    <resourceIDList>medical report</resourceIDList>
79  </hasDependencies>
80  <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_7967556643402429598" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.2" antecedents="//
        @hasServiceEntities.1" timeOperator="finish_to_start
        "/>
81  <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_3408208673744307361" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.3" antecedents="//
        @hasServiceEntities.2" timeOperator="finish_to_start
        "/>
82  <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_1793760504173610380" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.4" antecedents="//
```

```
        @hasServiceEntities.3" timeOperator="finish_to_start
        "/>
83    <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_1904139078890547379" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.5" antecedents="//
        @hasServiceEntities.4" timeOperator="finish_to_start
        "/>
84    <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_3275455577448539318" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.6" antecedents="//
        @hasServiceEntities.5" timeOperator="finish_to_start
        "/>
85    <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_6554961286330239770" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.7" antecedents="//
        @hasServiceEntities.5" timeOperator="finish_to_start
        "/>
86    <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_820379226549464573" dependentKpiName="startTime"
         antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.8" antecedents="//
        @hasServiceEntities.2" timeOperator="finish_to_start
        "/>
87    <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_4875848928162081674" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.10" antecedents="//
        @hasServiceEntities.8" timeOperator="finish_to_start
        "/>
88    <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_5077528085248911993" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.9" antecedents="//
        @hasServiceEntities.10" timeOperator="
        finish_to_start"/>
89    <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_6053961840993521907" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.11" antecedents="//
```

```
        @hasServiceEntities.9" timeOperator="finish_to_start
        "/>
90    <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_2731745990842422567" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.4" antecedents="//
        @hasServiceEntities.11" timeOperator="
        finish_to_start"/>
91    <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_4543076148445051841" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.13" antecedents="//
        @hasServiceEntities.8" timeOperator="finish_to_start
        "/>
92    <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_8588721146078344280" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.12" antecedents="//
        @hasServiceEntities.13" timeOperator="
        finish_to_start"/>
93    <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_3970245298381945313" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.14" antecedents="//
        @hasServiceEntities.12" timeOperator="
        finish_to_start"/>
94    <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_7895712797732936793" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.4" antecedents="//
        @hasServiceEntities.14" timeOperator="
        finish_to_start"/>
95    <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_243188543101594581" dependentKpiName="startTime"
         antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.15" antecedents="//
        @hasServiceEntities.1" timeOperator="finish_to_start
        "/>
96    <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_8218852081092408259" dependentKpiName="startTime
        " antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.8" antecedents="//
```

```
        @hasServiceEntities.15" timeOperator="
        finish_to_start"/>
97    <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_5390268172396839755" dependentKpiName="startTime
        " antecedentKpiName="startTime" dependent="//
        @hasServiceEntities.1" antecedents="//
        @hasServiceEntities.0" timeOperator="start_to_start"
        />
98    <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_4690594766522528545" dependentKpiName="endTime"
        antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.0" antecedents="//
        @hasServiceEntities.6" timeOperator="
        finish_to_finish"/>
99    <hasDependencies xsi:type="
        timeDependency:TimeDependency" id="
        DEP_3711626503308735243" dependentKpiName="endTime"
        antecedentKpiName="endTime" dependent="//
        @hasServiceEntities.0" antecedents="//
        @hasServiceEntities.7" timeOperator="
        finish_to_finish"/>
100 </depend:DependencyModel>
```

Listing D.1: Dependency model for healthcare service