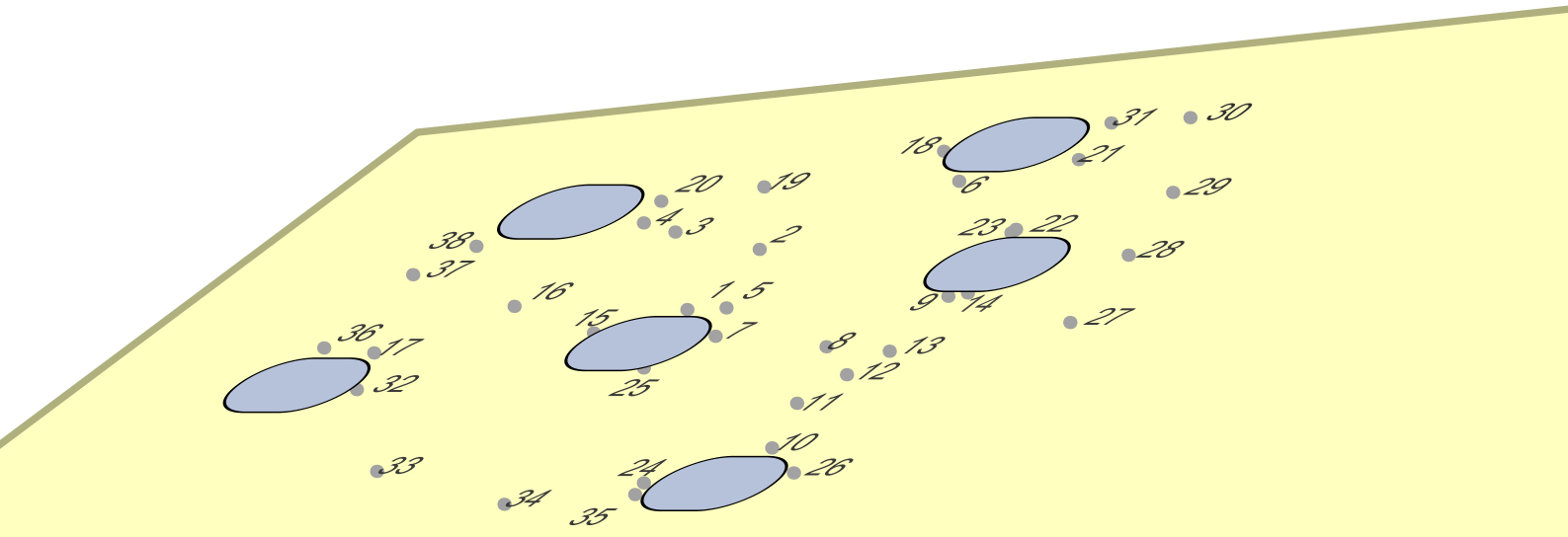
Semantics In Service-Oriented Architectures

Bart Vrijkorte



University of Twente
Enschede - The Netherlands

Semantics in Service-Oriented Architectures

A.B. Vrijkorte

August, 2006

Thesis for a Master of Science degree in
Business Information Technology, Architecture
from the University of Twente, Enschede, The Netherlands

Graduation Committee

dr. L. Ferreira Pires	University of Twente, Enschede
dr. A.B.J.M. Wijnhoven	University of Twente, Enschede
dr. ir. H.J.M. Bastiaansen	TNO ICT, Groningen
W. Pathuis	TNO ICT, Groningen

I have suffered a great deal from writers who have quoted this or that sentence of mine either out of its context or in juxtaposition to some incongruous matter which quite distorted my meaning, or destroyed it altogether.

Alfred North Whitehead
English mathematician & philosopher (1861 – 1947)

Preface

This thesis is the result of an assignment carried out for TNO ICT as part of a Master of Science degree program. I hope to have written a readable, enjoyable and in-depth report on the current state of the art in semantic service-oriented architectures.

I'd like to use this opportunity to thank my supervisors at TNO and at the university of Twente. Thanks go out to Harrie Bastiaansen, Wiltfried Pathuis, Luís Ferreira Pires and Fons Wijnhoven for their insightful advice and their commitment to this project. Furthermore I'd like to thank all the people who proofread earlier versions of this report.

Support of friends and family, and my colleagues at TNO, provided me with the motivation necessary to complete this project. I'd like to express my gratitude towards them.

Bart Vrijkorte
Groningen, July 2006

Executive Summary

In rapidly changing markets it is of primary importance for organizations to be able to cooperate in a flexible manner with diverse external partners. In such a joint venture it is readily required to integrate business processes and their associated IT support.

Unfortunately, in practice it often turns out that integrating applications and processes is time consuming, error-prone and expensive. As a consequence the realized synergy advantages are limited. An important cause for this problem is that business-to-business (B2B) application-integration is a manual task. On one hand some improvement may be realized by using a more structured way of working. On the other hand it is attractive to investigate possibilities for enabling automatic integration of applications and processes.

Proponents of the service-oriented architecture (SoA) forsee a future in which IT support is flexible. In this future IT functions are fulfilled by components that offer their services in a centralized directory. When an application needs a specific service, the available services in the registry are searched. The most appropriate service is found and the application can automatically make use of it.

Web-service technology is generally presented as a means for realizing the service-oriented architecture. Well known web-service techniques for describing (WSDL) and finding (UDDI) services however have an important shortcoming. These techniques namely describe services on a superficial, *syntactic*, level. From the syntactic description of a service it is not possible to unambiguously deduce what will happen in the real world when the service is used. Description of the *semantic* aspect of service provisioning is required to live up to the SoA promise.

Semantic web-service methods acknowledge this requirement and strive to enable architectures which are flexible and can be automatically composed. Currently however it is unknown which requirements should apply to methods for semantic web services and how these methods should be evaluated.

This report discusses the aspects that play a role in service provisioning. An important result is the summarization of important concerns in service provisioning and the resulting requirements on semantic web service methods in chapter 6. Based on this framework we derive requirements that methods for semantic web services should meet.

In chapter 9 two concrete methods are evaluated. In this chapter we show that the methods proposed in the scientific literature still have significant gaps in the offered functionality. Our conclusion is that at the current

time it is unlikely that organizations can build a service-oriented architecture in which components are fully automatically connected to each other.

Organizations can however make first steps toward a more semantic description of their services. Our recommendation is that organizations develop a data model of their business domain. Organizations can then describe their services in terms of this standardized domain. Taking these steps will initially facilitate a more structured approach to business process integration and in the long term it should clear the path towards partial automatic composition of architectures.

Contents

Part I

Introduction

1 Research Design

Information systems architecture is a broad and active research topic in which many developments take place. A current direction in IT architecture research is the Service-oriented Architecture (SoA). In this architectural style, software components work together in a loosely coupled way. The components offer their functionality through so called services that are made public to other components and external parties. Service descriptions can be retrieved from a centralized directory so that any component or third-party can locate a service and make use of it. In this way IT support is flexible and can be easily reconfigured when needs change. Rapid business processes integration between partners then becomes a possibility.

1.1 Motivation

Web services are seen as the main technology for realizing service oriented architectures, offering support for many of the ideas in that architecture. However, realizing dynamic coupling of components through a directory is still problematic. The essential problem is that it is unclear how services in a service-oriented architecture can be described in an unambiguous way. Communication about services is currently done on the basis of syntactic interface descriptions. This kind of descriptions provides no meaning behind the terms used and therefore the effects of using the described service are unclear. Those who wish to make use of a service have to guess for the semantic aspects. As a consequence, locating services and integrating business processes is a manual, error prone, slow and expensive activity.

This problem is especially relevant in the telecom industry. In the internet service provider market competition is fierce and there is a very high pressure to lower costs. Providers purchase connectivity from telecom operators and they want to be able to quickly engage in a business-to-business relationship without having to go through a lengthy and expensive integration process. Similarly, telecom operators want to be able to offer their services to new customers. It is therefore in their best interest to enable a fast business-to-business integration process. Ideally organizing the IT support should not be a limiting factor for building flexible business-to-business relations.

The main cause for the problem that business-to-business integration is slow, is that service description is currently done only on a syntactic basis. If the semantic aspects of services could be unambiguously represented then the integration process could be a lot quicker. It may even

become possible that no human support for the process is necessary and that intelligent software agents can forge business integration relations automatically. In the scientific literature several methods are proposed that claim to enable web services with semantic support, so called semantic web services.

1.2 Objectives

In this project we want to evaluate existing methods for building semantic web services. To do this we build a conceptual framework for semantic web services and we derive requirements from this framework. With these requirements we can assess the suitability of the proposed methods for realizing semantic web services. Furthermore we can see which problems surrounding semantic web services have been solved and which problems are still open for further research.

1.3 Approach

Our approach in this report was to first identify the need for service-oriented architectures. We did this by presenting a motivating case study from the telecom world. This case study illustrates the challenges that the service-oriented architecture was designed to address.

We characterized the service concept and then introduced the service-oriented architecture vision. We set out to clarify the motivation, ideas and concepts behind this architectural style. After that we investigated the web services platform architecture that is meant to be a realization of the service-oriented architecture. We evaluated the platform against the service-oriented architecture promises.

We determined that current web service technology ignores the semantic aspects of service provisioning. This led us to develop a framework that identifies the concepts that surround service provisioning. Based on this framework we developed requirements for methods that strive to enable semantic services.

The final phase in the research was to evaluate semantic web service methods with respect to the conceptual framework. The result of this analysis provides us with insight about what currently can be done and what open issues still exist in semantic web service research.

1.4 Research Questions

The research has been conducted in four phases. In the first phase we needed to clarify the ideas and concepts of the service oriented architecture. The following research questions are associated with these goals:

1. What are the benefits of the service-oriented architecture?

    a) What is the motivation for the service oriented architecture?

    b) What are the promises of the service oriented architecture?

In answering this research question we came up with a conceptual model of the service-oriented architecture. In the second phase of the research we investigated how service-oriented architectures are currently realized.

2. How are service oriented architectures currently realized?

    a) What technologies are available for realizing the service-oriented architecture?

    b) How do these technologies relate to the conceptual model of the service-oriented architecture?

    c) Are all concepts of the service-oriented architecture supported by current technology?

Answering these questions has given us insight in the capabilities and limitations of current technology.

In the third phase we investigated more closely the semantic aspects of service oriented architectures. This aspect is not included in current technologies and it is unclear what support for the semantic aspect should look like. This led us to the following research questions:

3. How can the semantic aspects of services be represented?

    a) Why is it necessary to represent the semantic aspects of services?

    b) What does a conceptual architecture for combining semantics and web services look like?

    c) What technologies exist to support semantic web services?

With the answers to these questions we established a framework that identifies the critical concepts and issues that surround semantic service provisioning. The final step was to assess currently available methods.

4. How do current methods for semantic web services compare to the framework?

    a) What methods are available that claim to support semantic web services?

    b) Can the methods be used in practice?

    c) What gaps still exist in the methods?

    d) What parts of the semantic service oriented architecture can be realized with current technology?

With the answers to these questions we were able to assess the usefulness of current semantic web service methods and to provide an in-depth advice for the migration towards semantically enabled service-oriented architectures.

1.5 Structure

This report consists of four parts. The first part of the report consists of three introductory chapters in which we detail the background for the research. In this first chapter we detail the research objectives and approach. In chapter 2 we introduce a case description situated in the telecom world that illustrates the context in which a semantically enabled service-oriented architecture is highly desirable. In chapter 3 we provide background on the service concept in a business administration context.

In the second part of the report we investigate the service-oriented architecture, its motivation and its promises. In chapter 4 we discuss the service-oriented architecture by means of a layered conceptual model. In chapter 5 we treat web services technology and compare it to the layered model of the service-oriented architecture. In this chapter the current state of technology is determined and gaps between the service-oriented architecture vision and current practice are identified.

The third part of the report is about the semantic aspects of service provisioning which are ignored in current web service technologies. In chapter 6 we provide an overview of these aspects and derive requirements for methods that strive to enable semantic services. Two large topics are discussed separately: in chapter 7 we investigate the central concept of ontologies in more detail, and in chapter 8 the subject of semantic discovery of services is treated.

In the fourth and final part of our report we evaluate proposed methods and present our conclusions. In chapter 9 we evaluate methods that are proposed in the scientific literature for realizing semantic services. In chapter 10 we summarize the findings of this report, provide advice for organizations that want to take steps toward enabling semantic services, and we make recommendations for further research.

2 Case Description

In this report we use a case study from the telecom world to illustrate the need for flexible IT integration. In this chapter we first discuss the way that internet connectivity services are currently provisioned. We then identify the real challenges that currently face the telecom world. We discuss the expected future developments and how the industry should cope with those. We signal a gap between the IT support that is required and what is offered by current technology.

Our case is centered around the way broadband connectivity is offered and managed by large telecom operators. The service we describe has many interesting aspects such as context dependence and significant non-functional characteristics. Part of this case was inspired on the case description found in [Duke et al., 2005].

2.1 Current Situation

In the internet connectivity market, internet service providers (ISPs) offer internet connectivity to customers who are also the end users of the con-

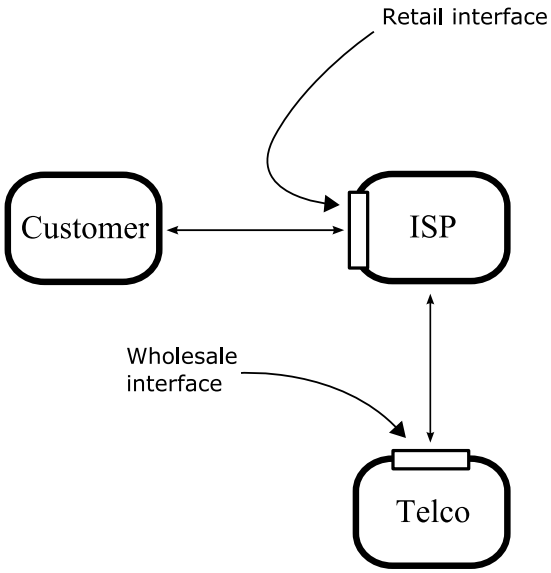


Figure 2.1: Current situation in the telecom world.

nection. The customer orders the service through a retail interface provided by the ISP. The internet service providers offer diverse additional services on top of the internet connection, such as e.g. e-mail, but they usually do not provide the actual connectivity to the customer. Instead, they order the connection from a telecom operator company (Telco). The commercial relationships between the involved parties are depicted in figure 2.1.

The interaction between the telecom operator and the ISP is a business-to-business relation. This relation is supported by enterprise application integration software. The telecom operator uses this software to offer a wholesale interface through which the ISP can order and cease connections for customers. This wholesale interface is the focus of this case study.

The wholesale interface offers a service to the internet service provider, namely to offer connectivity between an end-user of the internet connection and the ISP. The connectivity may be realized in many ways, depending on the location of the end-user. In some areas a high speed glass fiber connection can be offered but in other areas DSL connections may be available. Wireless connections might be offered in sparsely populated areas. There might also be areas where no service is offered at all. The role of the telecom operator is to offer connectivity to the internet service provider at the data link layer of the internet protocol stack. In some areas multiple connection types are offered and each of these is offered with different characteristics and against a different price. The telecom operator can offer multiple variants that differ in the guaranteed connection speed and availability. If a customer selects a different variant then the price is adjusted accordingly. For scoping reasons we assume that the services are offered exclusively in the Netherlands.

In the telecom world connectivity services are commonly described on three levels: the technical, operations and support level.

Technical On this level the precise details of the connection are described that together determine the quality of service. There are many frameworks that describe the quality of service of an IP connection, such as those defined by ITU/ETSI and IETF [Gozdecki et al., 2003]. For simplicity we consider only a limited number of quality of service parameters. These are bit rate of transferring (which defines the throughput of a connection), delay experienced by packets, jitter (variations in IP packet delay) and the packet loss rate.

Operations The operations level describes the processes that are required for fulfillment (providing ordering for connections), assurance (processes for monitoring and maintaining service quality) and billing. These processes are described by the eTOM (Enhanced Telecom Operations Map) framework [Milham, 2004]. This standard prescribes which business processes need to be organized to support the provision of telecom services. The standard does not prescribe the business processes themselves.

Support On this level we find descriptions of the availability of the service, the expected mean time between failures and the time required

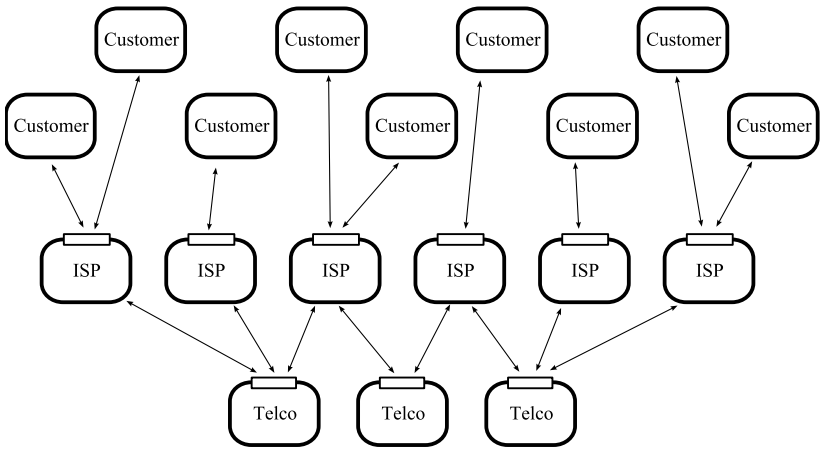


Figure 2.2: Future situation in the telecom world.

to repair faults. Furthermore this level describes the availability of the support desk.

The epBOM (eTOM public B2B Business Operations Map) standard by the TeleManagement forum specifies which operations should be supported by wholesale B2B interfaces in order to fulfill the eTOM framework. According to the epBOM framework, required processes include processes for fulfillment, assurance and billing. For simplicity and understandability we consider only a limited subset of these operations. In this case study, the wholesale management interface offers methods to order a new line, to cease a line and to report trouble. The interface is accessible by ISP's as a web service, supported by telecom operators.

2.2 Future vision

In figure 2.1 we identified only one ISP and only one telecom operator. In reality there are many ISPs and multiple telecom operators. We expect that in the future the number of ISPs will increase as will the number of competing telecom operators. As a consequence, competition in the market will be fierce. This will force each ISP to obtain connectivity at the lowest possible cost. As a result ISPs will want to be able to switch between telecom operators quickly and cheaply. Currently the process that is performed to enable the interaction between the ISP and the telecom operator through the wholesale interface is difficult and time consuming. This is because different systems at the ISP and the telecom operator need to be manually integrated. According to [Duke et al., 2005], up to 50% of IT costs in the telecom industry are attributable to integration activities.

The vision for the future is depicted in figure 2.2. In this vision, ISPs with connectivity requirements will look into a directory of services to find out which telecom operator can provide them with connectivity at some cost. After retrieving a description of the available connectivity services the best offer is selected and the services are procured from the selected

telecom operator. All this should happen automatically. The service description obtained from the directory includes enough information to automatically build the required B2B connections so that all management operations can be performed.

This vision for the future coincides with the service-oriented architecture vision which we discuss in chapter 4. Even though a lot of material on service oriented architectures exists, realizing the sketched ideal situation is not a solved problem. The important unsolved problem is that the semantic information which is necessary for business-to-business integration is lacking. The goal of this report is finding out if and how the service oriented vision can be realized by taking into account semantic aspects.

3 The Service Concept

In this report we are interested in describing the semantic aspects of service provisioning. Therefore we need to know what a service is and how services are typically provisioned.

The word *service* has multiple meanings. In the context of marketing research it has a different meaning than it has in the context of information systems research. In this report we build a bridge between the two worlds. In this chapter we discuss the service concept from a marketing angle. The information systems view on services is presented in section 4.3. We first characterize the service concept and then provide background on how services are managed.

3.1 Characteristics of Services

In the field of marketing there are many definitions of what a service is. There is currently no consensus which makes it very to give an exact definition. It is however possible to provide a workable characterization of the service concept. Often a service is described as "an activity or benefit that one party can offer to another that is essentially intangible and does not result in the ownership of anything. Its production may or may not be tied to a physical product" [Kotler and Armstrong, 2005]. Lovelock [Lovelock, 1992] additionally describes a service as being intangible and perishable, meaning that unused capacity at the time of production translates to lost economic opportunity.

This last point is illustrated by the example of an airline company: if at the time of a flight some seats are unoccupied then the airline company loses money. It is not possible to stock services.

Furthermore services are non-transportable and heterogeneous, meaning that the service is specific to one customer and that it is hard to mass-produce a service. Finally, services tend to be labor intensive, sensitive to demand fluctuations and are likely to involve a high degree of customer interaction. Due to these characteristics it is hard to achieve economics of scale with services. Examples of services include cleaning, medical care, entertainment, consulting and education.

Many researchers note that there is no hard division between a service and a product. Rather, there is a continuum between pure manufacturing and pure service provision [Voss et al., 1985]. All value-adding activities lie somewhere on this scale, depending on the amount of customer interaction and the tangibility of the result.

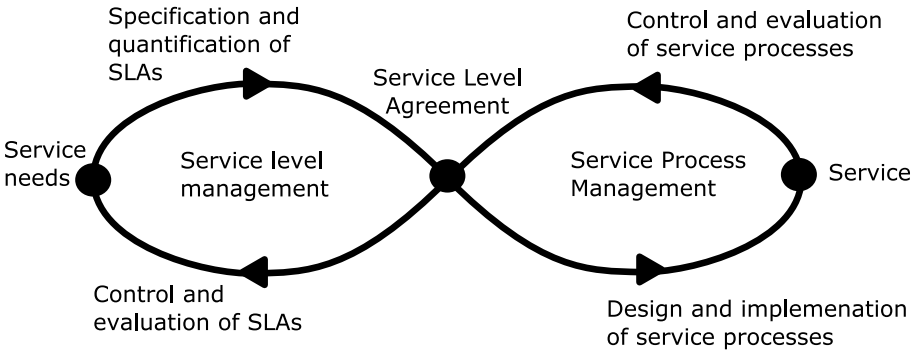


Figure 3.1: The service level lemniscate.

As mentioned in [Baida et al., 2003], a service is often offered as a collection of related services, sometimes called the 'service flower'. The primary service process is often supported by a number of support processes. For example, in the telecom world it is common to offer a service for reporting and rectifying failures as part of a connectivity service. Furthermore [Baida et al., 2003] observes that frequently a service is really an aggregation of simpler services. Providers add value by tuning different services into a coherent valuable package for a customer. This type of aggregation is called bundling. For example, in the telecom world we see that an account with an internet service provider often consists of an internet connectivity service, an e-mail service and sometimes even a phone service.

3.2 Managing Services

According to [Parasuraman et al., 1984] customers estimate the service quality by comparing performance with expectations. If the performance is according to expectations then the service will be considered to be of high quality. Therefore it is essential that a customer of a service knows what quality of service he can expect. In practice, especially in the IT world, Service Level Agreements (SLA) are used to manage service quality by defining the expectations. In an SLA the supplier and the customer of a service negotiate the properties of the service that is to be delivered. The expectations about the service performance are accurately defined in the SLA. This can prevent conflicts between supplier and customer later on since both parties agree on the service performance that is to be expected beforehand. [Thiadens, 2005]

An SLA is realized according to the service management lemniscate (see figure 3.1) [Trienekens et al., 2004]. As can be seen in the figure, there is a gap between the needs of the customer and what a supplier is prepared to deliver. In a continuous process the required service level is agreed upon and evaluated.

A service can be considered to consist of a service pit and a service shell [Trienekens et al., 2004]. The service pit is formed by the core (IT) object that is delivered to the customer, e.g. access to a network infrastructure. The service shell comprises the supporting processes and agreements of a

service. An example is the support provided by a helpdesk or a guaranteed level of data security. We consider the service pit to be made up of the essential criteria of the service and the service shell comprises the support criteria. A service level agreement specifies the characteristics of both service pit and shell.

In the connectivity service from our case description the essential criteria are the throughput of the connection, the delay experienced by packets, jitter and the packet loss rate. The service shell criteria include those criteria about the time required by the support desk to answer questions and resolve problems.

Part II

Service-oriented Architecture

4 Service-Oriented Architectures

In this chapter we introduce the service-oriented architecture vision. First we discuss the motivation behind the service-oriented architecture. Then we discuss the components of the service-oriented architecture using a layered structure. Finally we elaborate on what is meant with a service in a service oriented architecture and we contrast this with the service concept in the marketing field.

4.1 Motivation

In the past business to business relations between companies were often forged for many years. Integrating business processes and information support took a lot of time, but since relationships were stable this was not too much of a problem. Nowadays, however, business environments change quickly thereby forcing companies to adapt rapidly to changing circumstances. Information technology should therefore be flexible to support quickly changing demands. A common vision in the scientific literature is that the software that supports businesses needs to be built up from discrete parts of functionality. It should be possible to reuse these parts and to reconfigure then in order to adapt to changing environments. This vision is propagated by the component-based development approach.

A number of technologies that support the component-based development ideas can be found in [Stojanovic and Dahanayake, 2005]. Among these are CORBA, Enterprise Java Beans (EJB) [Sun Microsystems, 1999] and Microsoft's COM+ and .NET [Microsoft Corporation, 2006]. These technologies all are based on defining reusable objects with a strictly specified interface. In addition to the promised benefits of component-based software development, there are some shortcomings. The application developer who wants to make use of a component needs to know in advance which components are available and what interfaces are exported by the components. This approach is not very scalable since the problem of finding components tends to become harder when more components become available. Another problem of current components based development products is that products by different vendors (such as EJB and COM+) are generally incompatible. This seriously hinders cross organizational cooperation. The service-oriented architecture movement strives to address these shortcomings.

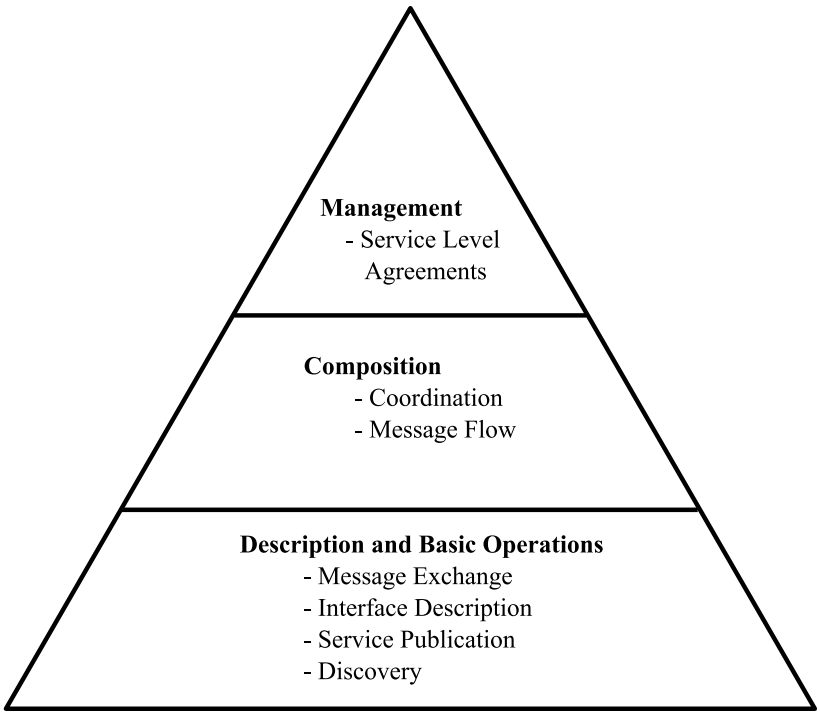


Figure 4.1: The service-oriented architecture pyramid.

4.2 Layering

The service-oriented architecture reuses the design philosophy of components-based development, namely that functionality should be split up over components that are loosely coupled. But while component-based development is mainly a software development methodology, service-oriented architecture can be placed in a broader perspective, namely as an entire information system architecture for organizations. As was discussed in [Orlowska et al., 2003], the SoA concept can be visualized as a pyramid. At the bottom layer, the basic operations layer, basic components (called services) offer discrete parts of functionality. The SoA architecture provides a mechanism for the publication and discovery of these services, thereby offering a solution for the scalability problem that plagues component-oriented approaches. At the second layer, the composition layer, we find technology that supports the aggregation of multiple basic services into composite services. In this layer technology support is necessary for coordination, monitoring of performance and quality of service. The top layer of the pyramid is the management layer, which prescribes that organizations should think about service level agreements, assurance and support for the information systems architecture. This pyramid structure is depicted in figure 4.1.

Service-oriented architectures should eventually enable a distributed computing infrastructure across the internet so that organizations can collaborate and integrate applications. Organizations will offer all of their

value adding activities to the public via the internet and when specific IT support is needed for performing a business process, the required services can be found quickly in a centralized registry. This registry can be shared between a number of companies that operate in the same industry sector. Discovery of services is a crucial aspect in service-oriented architectures. The availability of services will be dynamic and it should be possible to switch quickly and without problems between suppliers that offer a specific service. When this infrastructure is in place, IT support for the activities of organizations will be flexible and easily adjustable to changing circumstances.

The service-oriented architecture is only a vision of which web services are meant to be a concrete realization. Since the service-oriented architecture relies on universally accepted web service standards like XML, WSDL and SOAP, the interoperability problem that plagues component-based development should be eliminated. In chapter 5 we discuss web service technology in more detail and we investigate to what extent it can be used to realize the service-oriented architecture vision that we sketch here.

4.3 The Service Concept in Service-Oriented Architecture

In the context of a service-oriented architecture, a service is essentially a component that offers its functionality to external components through an interface that is described in a standardized way. The coupling between services is done through a registry of available interfaces, often called a directory [Stojanovic and Dahanayake, 2005]. In the directory a user can search for an interface that will meet his or her goals. The directory should also be searchable by individual components in order to support loose coupling.

A service in this sense (a software component that offers an interface) is different from the concept of a service in marketing (an intangible benefit offered by one party to another, see chapter 3). In general a service in the context of a service-oriented architecture is not necessarily a service in the marketing sense of the word. In this report however we take the position that *application services* in a service oriented architecture realize a higher level *business service*.

We think that this two-layer approach to services (the economic perspective combined with the computational perspective) is very useful. It requires components in a service-oriented architecture to offer an economic value to users. Thinking about services in this way makes it possible to make intelligent investment decisions about service development based on expected returns. Additionally, using this definition, it makes sense to offer services to an on-line market and to external parties. Finally, considering the economic benefit of a service can be a good starting point to decide what services to offer.

Examples of services that comply with this definition are: internet banking, electronic messaging and other forms of telecommunication services. In contrast, there exist computational services which do not offer a clear economic benefit. Typical examples are toy-services like “hello world” programs, or services which find out how many results a certain

keyword returns on internet search engines. A gray area is occupied by semi-manufactured products such as database look-up components that are only useful as part of a larger whole.

5 The Web Services Platform Architecture

In this chapter we investigate web service technology. We first detail the general concepts of web service technology. Then we discuss for each of the layers in the service-oriented pyramid (see chapter 4) how the activities at that level are supported by web services technology.

Web services can be seen as an implementation of the service-oriented architecture vision. Services are offered by components that can be implemented in conventional programming languages such as Java or .NET. Web service technology guarantees that components developed using any technology on any infrastructure can communicate with any other component that offers a web service description. The services are described in a standardized way and these descriptions are published in a centralized directory of services. Parties that want to use the service, retrieve the description from the directory and can then interpret that description to make calls to the service. The activity of finding and making use of a service is called binding. Parties can also offer services that are composed of other services. The description of such a composition is called choreography.

Web services technology is described in numerous standards. These standards are generally based on XML technology. For many of the components from the service-oriented architecture pyramid (section 4.2) a web service standard is available. In the web service architecture we find, among others, standards for message exchange, description of service interfaces, discovery of services and the composition of services. This chapter is structured according to the layers of the service-oriented pyramid. We discuss what standard support exists for each of these layers.

5.1 Description and Basic Operations Layer

The foundation of the service-oriented pyramid is formed by the description and basic operations layer. Components in this layer form the building blocks for building web service based architectures. We describe the standards that support message exchange, interface descriptions and service publication and discovery.

Message Exchange

The message exchange layer of the web services stack is provided in the form of the SOAP protocol. A SOAP message is an XML document that consists of a header and a body. The body describes the actual message that is passed to the web service, and header elements can be used for many purposes such as additional routing information or security requirements. The header information is also passed on to the called web service.

SOAP messages are transmitted between so called SOAP nodes. A message may pass any number of SOAP nodes and the SOAP nodes may inspect or alter the message headers. It is possible to specify that it is necessary for SOAP nodes to understand and process a header. In that case if a SOAP node does not understand the header, a fault condition occurs and an error message is sent back to the node that generated the SOAP message.

SOAP messages can be bound to any transport protocol. Examples are HTTP or SMTP (e-mail). HTTP transport, however, is the most commonly used. The SOAP standard intentionally does not prescribe an interaction protocol between the caller and the application service. Other web service standards, such as WSDL and BPEL, can be used to define the allowed message exchange patterns. [Weerawarana et al., 2005].

Interface Description

The set of operations that an application service offers is called the interface of the service. The WSDL standard (Web Service Description Language) is used to describe this interface. WSDL uses the concepts of type, message, portType, binding and service [Weerawarana et al., 2005]. For many of these concepts WSDL uses an extensibility mechanism, based on XML namespaces, so that new technologies can be used with a WSDL description.

In the types part of a WSDL document, data structures are declared that can be used later to define parts of messages. These structures can be defined in any schema language, but XML Schema [Fallside, 2001] is the de facto standard.

In the message part of the WSDL description, the structure of the messages that are exchanged between the web service and the caller is specified. A message consists of one or more parts of a specific type which are either built-in XML Schema types, or types described in the types section.

The portType part describes the input and output messages (if any) of the operations of the web service. Web services can use a request-response or a one-way interaction pattern. Theoretically WSDL makes it possible for the service to initiate the conversation but it is not supported by available implementations due to practical difficulties that arise when this form of interaction is implemented (e.g. it is unclear where to the message should be sent) [Weerawarana et al., 2005].

The binding part of the interface specification describes precisely how the messages are exchanged, using which protocol and using what layout. The de facto standard is to use SOAP over HTTP transport. The recommended *document* operation style entails that the content of the messages

are directly inserted in the SOAP body. There is also an *RPC* style that is supposed to make it easier to directly map methods defined in object oriented programming languages to SOAP messages.

Finally the service part describes the location where the web service can be found. For a SOAP web service this is specified in the form of a URL. The interface description in WSDL format only offers a description of the syntactic properties of an interface (the format of the incoming and outgoing messages), the semantics of the interface cannot be described in the WSDL language.

Service Publication & Discovery

Discovery for web services is provided by the UDDI (Universal Description, Discovery and Integration) standard. The UDDI standard is meant to facilitate the operation of a central registry or directory of web services. The UDDI uses a data model that stores information about service providers and services. A lot of basic information is stored about the service providers, such as names, descriptions, addresses, telephone numbers, etc. This information might even be stored in multiple languages. For services, a plain text description field is provided and one or more bindingTemplates point at the WSDL document necessary to call the service. Searching a UDDI directory can be done with a keyword-based search.

Recent versions of UDDI include support for digital signatures. With these digital signatures it is possible to verify the identity of the publisher of the service. This way the results from an UDDI query are more trustworthy. Furthermore, UDDI offers a subscription API for notifying clients when new services become available. [Weerawarana et al., 2005]

5.2 Composition

On the higher layers of the Service-oriented Architecture we find the need to support aggregation of services into composite services. WS-BPEL (Business Process Execution Language for Web Services) strives to model processes of web service executions. Individual web service calls are the steps in the process. The entire process description can be published as a web service to offer transparent web service aggregation. BPEL is meant to be interpreted and executed by machines so that long running processes can be executed automatically.

BPEL models business process logic in a procedural way. This means that all steps that have to be carried out by the interpreter are explicitly spelled out. Interaction with web services is specified by commands that receive and send messages. The language offers support for branching, loops, variables and calculations. Parallelism is supported as well, allowing business processes of arbitrary complexity to be modeled. Typically BPEL processes are bound to specific web services at deployment time. It is however possible to specify that the web services to be used are determined at runtime. This can be realized, for example, by querying a directory web service at the start of the process.

BPEL makes a distinction between executable and abstract process descriptions. Executable processes define all the logic that is to be followed

in the process and all the messages that are to be sent. This way an executable BPEL process offers a glass box representation of service behavior.

An abstract process is a projection of a BPEL process that only models the interaction with certain external parties. This interaction is defined in terms of the message exchanges that take place. Sharing this information with third parties may be necessary so they know what message exchange patterns are to be expected between the web services. Abstract process descriptions allow sharing of only the interaction information while keeping the actual business process details confidential, thereby providing a black box representation of service behavior. [Weerawarana et al., 2005]

5.3 Management

The management layer of the service-oriented pyramid contains tasks that (mainly) entail the arrangement and enforcement of service level agreements. In section 3.2 Service level agreements are briefly introduced. In this section we discuss what web service standards exist in this area.

For our treatment of the available technologies it is important to realize that an SLA primarily defines a set of metrics that need to be measured periodically. Furthermore an SLA includes a set of parameters that specify algorithms for determining the performance of the service given the input metrics. For each parameter, acceptable values and deviations are specified. Finally SLAs specify the involved parties and the consequences of violating the expectations raised in the agreement.

Important success factors for service level agreements are that the specification of the service contains only a limited number of measuring points that capture the essence of the service. The pricing for the service should be transparent and clear to all parties. A final requirement for a good SLA is that it contains a penalty clause that specifies the sanctions that are applied to a party that does not comply with the agreement [Thiadens, 2005].

For determining the service level parameters one can use an industry standard framework. In [Gozdecki et al., 2003] a number of these frameworks in the context of IP networks and their related parameters are discussed. Additionally they identify a number of relevant intrinsic parameters that apply to all such networks.

According to [Keller and Ludwig, 2003] SLAs are often specified in natural language and therefore they are also monitored manually. Big customers of an organization will want to negotiate a specific SLA for their purposes, and sometimes they want to specify custom measurement methods for assessing service performance. In contrast it is also possible for the supplying organization to define a standardized portfolio of services and associated SLAs. It is desirable however to automate the monitoring of SLAs, this requires a standardized way of specifying SLAs.

WS-Policy

In the current web services platform architecture there is no comprehensive standardized support for monitoring and enforcing the quality criteria

specified in an SLA. There is, however, the WS-Policy web service standard which aims to support non-functional capabilities and requirements for web services [Weerawarana et al., 2005]. We briefly describe this standard here.

A policy specification contains a list of policy containers that each represent a valid policy alternative. Each of these alternatives is described by a list of assertions that must hold about a service. The subject of the assertions is free form, and only presence or absence of an attribute can be described. For example, we can express in the language that the security should be guaranteed by performing the RSA algorithm. We can not specify that at least 128 bit encryption should be used, but we can specify that exactly 128 bit security is to be used. Similarly we can not specify a minimum up-time or a maximum delay of a connection. Other complicated constructs, such as if-then-else constructs or assertions that are only valid on certain dates or times are not available. This all seems too limiting as many current SLAs require this level of complexity for specifying minimum response times, availability during peak hours, etc. Furthermore, the WS-Policy standard offers no support for semantics. The policy descriptions can be interpreted only if the meaning of the subject (encryption in our example) are defined externally. In its current form the WS-Policy standard is not really sufficient to describe the quality criteria of web services.

WSLA

WSLA is an XML based language for electronically representing service level agreements [Keller and Ludwig, 2003]. WSLA strives to model in an unambiguous way the SLA parameters, the way of measuring performance and the consequences of violation of the SLA. A WSLA document contains a description of the involved parties, which have a direct interest in the SLA, such as the supplier and the customer. It also gives the possibility to define third parties, which play a role in monitoring compliance with the SLA. Contact information is specified for all parties. A WSLA document also contains a service description section that defines the observable parameters of the service. It describes the metrics that can be collected, how these can be collected and by whom. The values for the metrics can be aggregated, e.g., to obtain average values over time. Finally, the service description section contains the definitions for the SLA parameters. The obligations section specifies, using logic conditions, under which circumstances (e.g., during what times, or under what system loads) what values for the metrics are acceptable. It also contains a section that describes what actions are taken when a violation is detected. In its current version WSLA only supports the notification of SLA violations to interested parties.

A limitation of WSLA is that there is no provision for specifying and monitoring the behavior of non-automated services such as for example a help desk.

Just like WS-Policy, WSLA offers no support for defining the semantics of the criteria that are measured. As a result we conclude that it seems possible for a computer to understand and monitor the compliance with an

SLA, but that it will be difficult for algorithms to reason about the value of an SLA for a customer. Automatically negotiating and enforcing the terms of an SLA is out of reach with current technology.

5.4 Evaluation

With the technologies discussed in this chapter it is possible to specify a service in the context of service-oriented architectures on a syntactic level. We can specify the input and output parameters of services and we can describe composite services that are built up from smaller services. This information, however, carries little meaning for the user of a web service. A customer will have to contact the supplier of a service to find out its expected behavior or alternatively will have to interpret and guess the meaning of the service from the input and output parameters. This is a time consuming and error prone process that prevents dynamic switching of web service suppliers. The lack of semantics hinders the realization of the service-oriented architecture vision discussed in chapter 4. What is necessary is a complete and unambiguous description of a service so that it is clear to the customer what the application service does. In the next chapters we investigate how the context of a service can be described in a way that can be interpreted by machines.

Part III

Semantic Aspects

6 A Framework for Semantic Web Services

Semantic web services are web services that are described in such a way that machines can reason about their utility. In this report we investigate a number of methods that strive to enable such semantic services. In order to evaluate these methods we need to establish the evaluation criteria which are to be used. In this chapter we therefore discuss what aspects play a role in the semantic description of business and application services.

We first discuss the vision of the semantic web, on which much of the current work in the field of semantic service description is based. Then we introduce our own framework that identifies the important concepts in semantic service description and delivery. Based on the framework we identify a number of concerns that should be addressed by methods that strive to realize automatic, semantic service delivery. Finally we arrive at a list of concrete requirements which we can use to evaluate concrete methods.

6.1 The Semantic Web

In [Berners-Lee, 2001] the usefulness of a semantic web is discussed. The article discusses how in a semantic web information is given a well-defined meaning, thereby "enabling computers and people to work in cooperation". The article sketches a vision where software agents are able to combine information from many heterogeneous sources and draw conclusions based on the combined information. This way complicated tasks can be performed. The article provides the example of making appointments with a medical specialist. The agent has to consider available dates and times of multiple people, the rating of the specialist and whether or not the insurance company has a contract with the specialist. Berners-Lee et al. foresee that in the semantic web, web pages provide information in semantic form to be processed by computers.

Semantic web technology should also be useful to realize service oriented architectures, and to facilitate flexible business-to-business integration. In order to do this services should be described semantically such that a computer can interpret the service description. Describing services in this way will enable dynamic binding of services which is a crucial part of the service-oriented architecture vision. Semantic web technology

promises to be able to describe the meaning of information in a computer understandable way.

6.2 Aspects of Service Provisioning

For our introduction of the service-oriented architecture in chapter 4 we utilized the pyramid structure as proposed by [Orlowska et al., 2003]. In this pyramid, and in the current web services platform architecture, the semantic aspect of service provisioning is disregarded. Without considering this semantic aspect a number of tasks in the service-oriented architecture, such as service description and discovery seem simple. These tasks were therefore grouped in one layer, situated at the bottom of the pyramid.

However in semantic web-service research, tasks like service description and discovery are at the center of attention. Currently there is no scientific consensus about how discovery of services should take place or how services should be described. It is not even clear which are the important issues in semantic service-oriented architectures.

Since the subject matter of semantic web services is so complex and since it does not fit the pyramid model introduced in chapter 4, we decided to develop our own framework of this domain. Our framework provides an overview of the concepts that enable service provisioning in a semantic service-oriented architecture. Instead of splitting up the subject in layers (as was done in the service-oriented pyramid), our framework provides a holistic view of the application domain.

The framework we introduce in this section consists of a structural and a procedural viewpoint. We discuss these viewpoints separately.

The Structure Viewpoint

The structure viewpoint describes the field of semantic service provisioning. In figure 6.1 this viewpoint is depicted. We discuss the the elements in the figure separately.

We consider the *service* concept in a service-oriented architecture to be made up of two parts: a business service and an application service. The *business service* concept represents the economic value of the service. The *application service* is the software component that realizes the business service.

Services can be built by bundling simpler services into a larger package. Bundling services is discussed in chapter 3. In the diagram, bundling of business services is represented by the *business service bundle* concept. Each bundle consists of a number of *atomic business services*. An atomic business service is a business service that is not made up from other business services. The bundling of business services is reflected in the bundling of application services (*application service composition* and *atomic application service*).

A business service contains a *business process* that provides the added value of the service. Business processes can be described in terms of their *effects* or in terms of the *activities* that are carried out in the process.

Another important component of a business service is the *service level agreement*. Service level agreements are used to manage expectations of

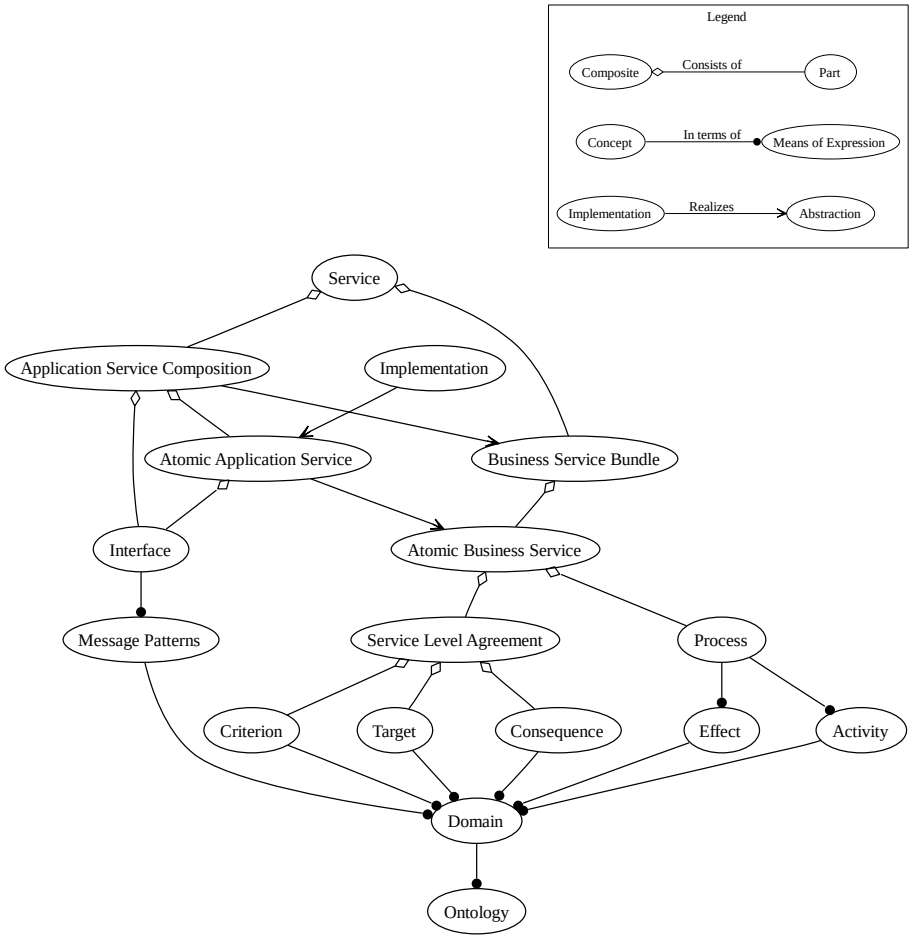


Figure 6.1: The structure viewpoint.

service performance (see chapter 3 for details). A service level agreement contains a list of *criteria* which are used to measure performance. The performance is compared to a predetermined *target*. The *consequences* clause of an SLA determines what happens when the agreed performance was not realized.

An application service offers an *interface* through which it can be used, all communication by external parties with the application service goes through the interface. The interface is defined in terms of allowed *message exchange patterns*. An *implementation* in software is used to realize an atomic application service.

As can be seen in figure 6.1, most concepts in the framework are expressed in terms of the (application) *domain*. It represents the view that in order to enable semantic communication about a service, the application domain must be defined. When we want to communicate about the utility of a service, we will always need to do so this in reference to a certain domain.

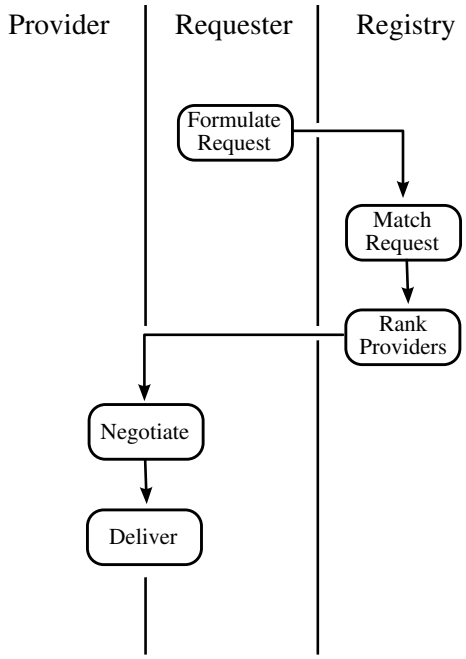


Figure 6.2: The process viewpoint.

To facilitate communication, the parties must share a common vocabulary about the service, this vocabulary can be expressed in an *ontology*. An ontology is a means to represent knowledge in a computer understandable way. In figure 6.1 we see that ontology is the central concept in a semantic service-oriented architecture: all concepts are expressed using an ontology. Ontologies are discussed in detail in chapter 7.

The Process Viewpoint

The process viewpoint is illustrated in figure 6.2. The process viewpoint of the framework describes how services can be found and executed in a semantic service-oriented architecture.

Service delivery is initiated by a requester who formulates a service request. Then, a centralized registry matches this request against service descriptions which were previously registered by providers. The service descriptions that match the service request are ranked by their relevance and returned to the requester. The requester then initiates a negotiation session with the provider to determine the exact service parameters. Finally, when an agreement is reached, the service is delivered in accordance with the agreement. The subject of service discovery is treated in more detail in chapter 8.

6.3 Realization Concerns

From the framework introduced in section 6.2, we can derive a number of specific issues that arise in realizing semantic services. A method for semantic services should provide a means for dealing with each issue. We can group the issues in three levels: the business service level, the communication level and the delivery level. Many of the terms we refer to have been introduced in figure 6.1, these terms are typeset in italics below.

Business Service Level

At the business service level we are concerned with representing the utility of a service. A number of concerns are relevant on this level: modeling *business processes*, representing *service level agreements* and specifying *bundling* of services.

Business processes describe the value-adding activities that take place in the delivery of a service. Processes can be represented in two different ways: a procedural and a declarative way. The procedural way represents processes as an ordered set of *activities* that are carried out in the course of the process. The declarative way describes only the *effects* of the process, not how these outcomes are realized.

Processes can be described in formal or informal ways. Formal specifications have meanings that are precisely defined by mapping statements on mathematical structures, whereas informal specifications rely on the interpretative capabilities of the reader. Computers are only able to interpret formal specifications hence process specifications for semantic services will need to be formal.

Examples of formal procedural specifications of processes are SADT diagrams (as described in [Congram and Epelman, 1995]) or Petri Nets (described in [van der Aalst, 2002]). Prolog, a logic programming language, is an example of a method that can be used for formally describing processes in a declarative way [McIlraith and Son, 2002].

The ideas behind *service level agreements* are introduced in section 3.2. Methods for representing service level agreements are discussed in section 5.3. A semantic web service method should provide support for negotiating and monitoring service level agreements.

Bundling refers to the fact that services can be built by aggregating simpler services into a larger package. Bundling is discussed in chapter 3. Since it is common to bundle services in order to add value, a semantic web service method should offer support for bundling.

Communication Level

At the communication level, the main issue is to represent the terminology of the application *domain* in a machine understandable way. Furthermore there is the problem of *matchmaking* between service requesters and providers. A final concern is how to represent allowed *message patterns*.

Ontologies are a way of representing terminology in a computer understandable way. Usually ontologies are defined using a knowledge repre-

sentation language based on logic. Since ontology representation is such a large topic, it is discussed separately in chapter 7.

The process of *matching* providers and customers of services is called discovery. A semantic web service method should provide an automatic way of locating providers who can meet a customer's demand. This subject is treated in more detail in chapter 8.

During service delivery the provider and requester exchange messages. Semantic web service methods should provide a way of representing these *message patterns*. A difficult issue is that of heterogeneity in message exchange patterns. For communication purposes, the provider and requester must be able agree on the allowed message exchanges.

Delivery Level

At the delivery level we are concerned with resolving details about the exact parameters of service invocations.

The discovery process usually does not result in an unambiguous service description that is correct and complete (see chapter 8 for details). To address this issue [Preist, 2004] proposes to include a *negotiation* phase after the discovery phase, during which the concrete service parameters can be negotiated from an abstract description. During the negotiation phase the provider and requester agree on the exact parameters of a service and the message exchange patterns.

Since advertising concrete services is often infeasible, any semantic web service method should offer support for refining and possibly negotiating the exact parameters of a service.

6.4 Requirements on Semantic Web Service Methods

Based on the conclusions of sections 6.2 and 6.3, we are able to derive concrete requirements for methods that strive to enable semantic web services. In this section we present the requirements that apply to methods that strive to realize semantic service-oriented architectures.

1. Business process modeling. The method should support describing processes that define the value-adding activities of the service. Processes can be represented procedurally or in a declarative manner.

2. Service level agreements. A semantic web service method should provide support for negotiating and monitoring service level agreements.

3. Bundling. Since it is common to aggregate services in order to add value, a semantic web service method should offer support for creating bundles of services.

4. Ontology. An ontology-based language should be used to represent the semantics underlying the service.

    a) A method should be able to account for ontological heterogeneity that occurs when different organizations independently develop their own domain models (see section 7.5).

5. Discovery. A semantic web service method should provide an automatic way of locating providers who can meet a requester's demand.

    a) (preferable) Matching is done on the basis of ontological descriptions.

    b) (preferable) Matching is done on the basis of the transformation(s) of the world that delivery of the service achieves.

6. Message patterns. A semantic web service method should have a way of representing message patterns and conforming to existing message exchange patterns.

7. Parameter refinement. A semantic web service method should offer support for generating concrete parameters from an abstract service description.

    a) (desirable) The method offers support for automatic negotiation about parameter values between provider and requester.

6.5 Validation of Requirements

Validating requirements on software architectures is a difficult issue which has not received extensive treatment in the scientific literature. In this section we therefore discuss a number of ways in which we could validate the requirements introduced in section 6.4.

Validation Approaches

An intuitive approach to validating requirements would be to check if fulfilling the requirements will necessarily lead to a satisfying solution to the problem. The difficulty with this approach is that this method of proof is infeasible. It requires building all possible artifacts that meet the requirements and then to check if the solutions solve the problem and fulfill the requirements. This is not a practical strategy to requirements validation since validating the requirements would entail building the solution to the problem.

We take a more practical approach: validation of individual requirements can be done on the basis of use cases that each address a part of the problem. By checking the necessity of each requirement in section 6.4 by means of a scenario, a reasonable argument for the merit of each requirement can be made. Even though no guarantee for the completeness of the requirements can be given, this approach does provide us with insight in the value of the requirements.

The approach is especially useful for our purposes since we want to check the suitability of semantic web service methods for their intended use. This means that if we know a set of necessary requirements, we can check if the methods fulfill these requirements. If a method does not fulfill all requirements we know that problems may arise when we try to realize a semantic service-oriented architecture using the method. If it does meet

all requirements we have a reasonable indication that the method is suitable for this purpose. The scenarios which we will use are inspired on the case description introduced in chapter 2.

Scenarios

In this section we provide a number of scenarios to motivate the requirements introduced in section 6.4. For each scenario we indicate what requirements are necessary to fulfill the scenario.

1. An ISP wishes to purchase connectivity to connect a customer to its own network. In order to do this the ISP will need to find a service that fulfills this goal.

   The ISP formulates the request for a new connection in terms of pre- and postconditions in an ontology. Then the request is sent to a party that matches supply and demand.

2. A matchmaking party compares an incoming service request with all service offers registered by the providers. The most relevant matches are determined using ontological comparison and send back to the requester.

3. In order to enable service delivery it is necessary to define what messages can be exchanged between the requester and the provider. Often the message exchange patterns are simple, but this scenario provides an example of a more complicated exchange pattern.

   When a problem occurs with the connectivity between the telecom provider and the ISP, a diagnostic procedure needs to be carried out. This procedure might consist of a number of checks that are performed on either end of the connection with the results reported back to the other party. A powerful method for defining these exchanges is required.

4. An ISP wants to order a connectivity service from a specific telecom operator. However the telecom operator does not use the same terminology as the ISP. Mediation is necessary to translate the messages of the ISP into a format that the telecom operator understands and vice versa.

5. In order to remedy failures, the telecom operator offers a fault-clearing service with each connection. Failures can be reported to this service so that they will be resolved. This service is only worthwhile in combination with a connectivity service and must be purchased simultaneously.

6. The service that the telecom provider offers depends on the location of the end user of the connection. In some areas higher connection speeds can be realized than in others. A negotiation process between the ISP and the telecom operator ensures that the desires and possibilities match.

Requirement:	1	2	3	4	5	6	7
Scenario 1:	•			•			
Scenario 2:					•		
Scenario 3:						•	
Scenario 4:				•			
Scenario 5:			•				
Scenario 6:							•
Scenario 7:		•					

Table 6.1: Requirements traceability matrix

7. An ISP wants to order connectivity for a business customer. The business customer requires specific, measurable guarantees about availability and quality of the connection. The ISP searches for telecom operators that are willing to agree to these specific requirements.

Tracing

Table 6.1 indicates which requirements are necessary to fulfill each individual use case. When a requirement is necessary to fulfill a scenario this is indicated with a mark. From this table we can deduce that each requirement mentioned in section 6.4 is covered by a use case. The use cases are realistic and representative for realizing a semantic service-oriented architectures in the telecom sector. The use cases are specific to the telecom sector, but there is no indication to believe that the requirements are only applicable to the telecom sector. It is more likely that these requirements are in general applicable to methods for realizing semantic service-oriented architectures and that therefore they form a good basis to evaluate methods for semantic web services.

7 Ontologies

In this chapter we investigate ontologies as a means for modeling domains. We discuss what ontologies are and why they are useful. We investigate the different approaches for representing ontologies and evaluate those for our purpose. Finally we discuss how ontologies can be built and we discuss the merits of reusing standardized ontologies.

7.1 Definition

As discussed by [Singh and Huhns, 2005], communication about services is meaningless without a common understanding of the domain. Ontologies are a means to realize this shared understanding and are therefore an essential part of any semantic web service architecture. The term ontology is defined in [Guarino and Giaretta, 1995]:

> [An ontology is] a representation of a conceptual system that is characterized by specific logical properties (special type of logical theory containing only necessarily true formulas).

In figure 7.1, taken from [Guizzardi, 2005], the parts that make up an ontology are placed in perspective. This figure shows that a model of the world is represented by a formal specification. This specification is composed using a modeling language. Behind the modeling language are

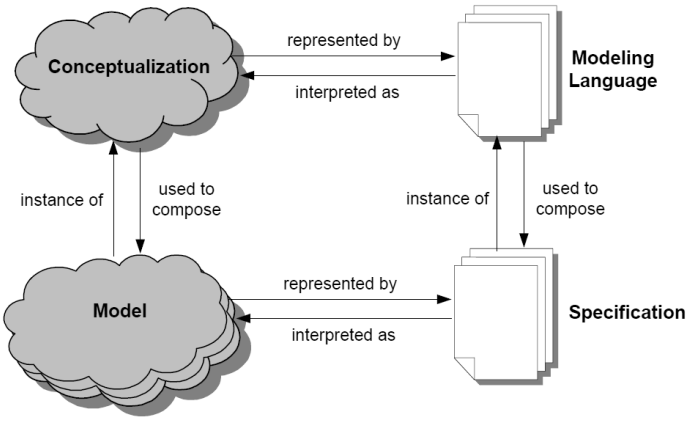


Figure 7.1: Terms surrounding knowledge representation.

assumptions about how knowledge about a domain should be represented: the conceptualization. The entire system is called an ontology.

7.2 Conceptual framework

In [Reichgelt, 1991] a number of the most common conceptualizations for representing knowledge are discussed. This discussion is based around a conceptual framework for knowledge representation which we introduce in this section.

According to the framework a modeling language consists of a *syntactic* and an *inferential* aspect. The syntactic aspect covers the issue of how expressions in the language are made. The inferential aspect addresses how new conclusions can be drawn from existing knowledge. The framework is further subdivided in four levels. These levels are the *implementation*, *logical*, *epistemological* and the *conceptual* level. We discuss each of these levels here.

Implementation: on the implementation level one is concerned with building a computer program that can interpret the knowledge representation language. On this level issues of data structures and algorithms play a role.

Logical: the logical level is concerned with the precise meaning of the constructs in the language. Furthermore on this level we consider what kind of information can be expressed using the constructs. An example of a question on this level is: "can we express that either statement a or statement b is true?"

Epistemological: The epistemological level is concerned with the way that knowledge is structured in the language. The choice for a certain type of structure is linked with the application domain in which the language will be used.
We explain these ideas using some examples inspired on our case study. A possible choice at this level is to represent the telecom domain as a list of unstructured facts: 'a *connection* runs between two points', 'a *person* has a *name*', *'bob* is a *name*', 'connection y belongs to the person named *bob*'.
Another possible choice is to use an object-oriented style of modeling. Using this style *connection* and *person* would be a classes. The classes would have attributes associated with them such as *name* and *throughput.*
The choice for a certain structure should be based on the intended application and the structure of the application domain.

Conceptual: At the conceptual level decisions are made about the primitives that are included in the language. The primitives that are applicable are prescribed by the choices made at the epistemological layer. For example, if at the epistemological level it is decided that a network structure is used, then at the conceptual level the required connection types and their conceptual meanings are determined.

Requirements

Next to the conceptual framework for knowledge representation we discussed previously, [Reichgelt, 1991] also proposes a number of requirements for modeling languages. The requirements are organized according to the layers of the conceptual framework.

Implementation The most important requirement on knowledge representation languages at the implementation layer, is that of efficiency. Drawing inferences should be reasonably fast and representing knowledge should be space efficient.

Logical From a logical stance, a knowledge representation language is required to have clear semantics. This entails that all the syntactically allowed constructs should have a clear and unambiguous meaning. Furthermore, the inference rules should be sound, this means that it should not be possible to draw false conclusions from true information in the model.

Epistemological The main requirement at the epistemological layer is that the construction and interpretation of representations should be natural. The structure of the modeling language should correspond to the structure of the domain in which the language is applied. In some domains a rule based approach might be desirable whereas in other cases an object-oriented language might be more natural.

Additionally a language should be modular so that when a piece of information needs to be changed, only a small part of the representation has to be changed. Furthermore the granularity of the language constructs should be appropriate, knowledge might be represented as a collection of facts (fine granularity) or it might be structured along larger concepts (course granularity).

Finally the primitives chosen at the conceptual level should align with the decisions made at the epistemological level.

Conceptual At the conceptual layer the most important criterion is that knowledge should be representable in a concise way. In [Reichgelt, 1991] it is argued that if it is impossible to represent a simple piece of knowledge in a concise way, then the primitives used at the conceptual level are probably wrong. Similarly if a simple inference requires complicated constructs then the inference procedure is not adequate.

7.3 Types of Knowledge Representation Languages

In this section we discuss the different approaches to knowledge representation found in the literature. We discuss *first order logic, semantic networks*, *frame-based logics, description logics* and *rule based logics*. For each of these approaches we discuss the underlying principles and we indicate languages that take the approach.

First-Order Logic

First-order logic is the classical mathematical way of representing logical statements. The standard notation for symbolic logic, the Peano-Russell notation, was introduced in 1889 and was structured after algebra. In [Sowa, 1983] a brief overview of first order logic is provided.

Knowledge is represented as a set of *predicates* about variables. Predicates can be connected with boolean operators ($\wedge$, 'and'; $\vee$, 'or'; $\neg$, 'not', $\rightarrow$, 'implies') which can be used to create complex expressions. Furthermore, there are the universal ($\forall$, 'for-all') and existential ($\exists$, 'there exists') quantifiers.

A first-order logic language is built up from sentences where each sentence is a logical statement that is true. In the example below we illustrate how we can represent that a connection which is reliable must be expensive:

$$
\begin{aligned}
&Connection(a)\\
&Expensive(b)\\
&Reliable(c)\\
&\forall x : Connection(x) \wedge Reliable(x) \rightarrow Expensive(x)
\end{aligned}
$$

In [Reichgelt, 1991] the benefits and drawbacks of first-order logic are discussed. Firstly, a benefit is that first-order logic notation is very expressive. As a consequence it should be possible to represent almost any knowledge representation language in first-order logic primitives. Secondly, the meanings of the constructs in the language are well defined.

The drawback is that reasoning in first-order logic is inefficient and undecidable, that is, we cannot guarantee that the truth-value of an expression can be determined in finite time. Additionally, reading and writing first-order logic seems not very natural: the notation requires training to understand. Furthermore, first-order logic does not enforce a structured representation of the facts. This can easily lead to domain models that are hard to understand and to maintain. For a large domain such as service provisioning in the telecom world it is likely that these issues pose major problems.

Semantic Networks

Proponents of semantic net languages take the point of view that associations between concepts are the most important concern in representing meaning. As such, a semantic net language is built up from two primitive types: nodes and unidirectional arcs [Reichgelt, 1991]. Nodes represent concepts in the world and the arcs (or links) represent the relations between the concepts. Arcs and nodes can be annotated with labels to link the constructs with the human understanding of a term. Conclusions are drawn by following the arcs between nodes.

RDF The RDF language [Manola and Miller, 2004] is proposed as the facilitating language for the semantic web in [Berners-Lee, 2001]. It is a

semantic network language that is used to state assertions about resources on the internet. Resources are modeled as nodes and are annotated with URLs. Arcs represent relations between nodes and are annotated with URLs as well. The meaning of the links in an RDF graph is determined by the meaning that a parser associates with the used URL. Since RDF specifies no meaning for the arcs between nodes, the language has no logical or conceptual foundation. Consequently, an RDF document by itself has no meaning.

RDF-Schema Since RDF has no meaning on its own, the RDF-Schema language [Brickley and Guha, 2004] was defined. This specification allows us to define, using RDF syntax, classes and the relationships between classes. We can say that a certain class is a subclass of another class and we can define attributes. We can also specify that certain attribute arcs can only take certain kind of destination classes (range restriction). The reverse, specifying that certain arcs only take a certain kind of origin classes (domain restriction) is also possible. This way RDF-Schema presents a conceptual foundation for the RDF language.

However, RDF-Schema does not have a logical foundation. The meaning of many concepts is not defined precisely and there is room for interpretation by reasoners. An example of this is that it is not specified how reasoners should handle range and domain restrictions. One interpretation might be that a violation of this restriction leads to an invalid document. Another interpretation is that the association turns the connected classes into subclasses of the desired type. The RDF Schema specification explicitly leaves open both interpretations.

Another problematic aspect is that it is possible to modify the RDF-Schema language using its own constructs. As a consequence it is almost impossible to build reasoning software for RDF-Schema based models.

Frame-based logics

Frame-based logics are based on the intuition that humans organize knowledge in a hierarchy of concepts [Reichgelt, 1991]. At the top of the hierarchy abstract concepts are placed and deeper in the hierarchy we find more specific concepts. Many frame languages allow for multiple inheritance. This makes it possible to have a specific concept as a specialization of multiple abstract concepts. The concepts are called *frames*. The description of a frame is constructed using *slots* that are associated with it. A slot can be a name-value pair, but it can also point at another frame, thereby forming an association. Slots can be given default value at a frame higher in the hierarchy, which can be overridden by a more specific frame. Additionally it is possible to put restrictions on the possible values for a slot.

Reasoning in a frame logic is generally done on the basis of matching and inheritance. The first step is that some information about an unknown concept is given to the reasoner. On the basis of the known frames, their slots and their restrictions, a set of super-classes of the concept is determined; this is called matching. When the super-classes of the unknown concepts are known, it is possible to look up all defined slots and the de-

fault values for slots in the super-classes. This way statements about the unknown concept can be made.

According to [Reichgelt, 1991], frame logics are appealing from an epistemological standpoint. The expressive power of frame logics, however, is rather limited. It is not possible to specify statements like ("a connection is unreliable, or it is expensive"). This is because traditional frame based logics only make statements about classes of objects. Statements that reason about the relations between classes, such as the one mentioned, are not possible.

F-Logic An example of a frame based logic is the F-Logic language described in [Kifer and Lausen, 1989]. F-Logic additionally contains connectives ($\wedge$, $\vee$, $\neg$) and quantifiers ($\exists$, $\forall$) beyond those found in traditional frame based logic. These can be used to specify complex frames while still keeping the language decidable.

WSML-Flight WSML is a language that is specifically proposed for annotating semantic web services described in [de Bruijn et al., 2005]. The WSML-Flight profile is based on the F-Logic language described above. We discuss the WSMO project (which WSML is part of) in more detail in chapter 9.2.

Description Logics

Description logics are quite similar to the frame-based logics described earlier in this section. Description logics allow us to specify a hierarchy of concepts with associated descriptions. A major difference is that description logics make a distinction between the concepts (called classes) and their instances (called individuals). A further difference is that description logics do not allow subclasses to override attribute values defined in their superclasses. Classes can be connected through *roles* with other classes. This mechanism can be used to specify attribute values. Complex classes can be specified through logic expressions on other classes. Reasoning on description logics is done by determining what classes an instance belongs to or by determining the super-classes of a specific class. [Baader and Nutt, 2003]

It is possible to build reasonably efficient reasoners for description logics and the representation seems natural for the description of services. The expressivity of description logics, however, is (similarly to frame languages) rather limited.

OWL OWL comes in three variants: OWL-Lite, OWL-DL and OWL-Full. Only the first two can be considered description logics. However, as elaborated in a.o. [Preist et al., 2005], there are still some shortcomings. One of these is that OWL-DL cannot reason over restrictions on data ranges even though this is allowed according to the description logic theory. For example, it is not possible to specify a class *fast_connection* for which the *speed* property is defined to be more than 1024 kbs. In OWL-DL we can only restrict a datatype property to a single value (such as speed = 1024 kbs).

OWL-Lite is a restricted version of OWL that is easier to reason about. The OWL-Full profile was designed to offer a migration path from RDF-Schema. However the authors admit that it is close to impossible to build reasoning software for OWL-Full. [McGuinness and van Harmelen, 2004]

WSML-DL As described in [de Bruijn et al., 2005], WSML offers a description logic profile as well. The properties of this profile are comparable to OWL-DL.

Rule-Based Logics

Rule-based logics model the world as a list of antecedent consequent pairs. Reasoning is done by checking which antecedent conditions hold. Based on that knowledge the consequences can be inferred. Rule-based systems lend themselves well for modeling pre-conditions and post-conditions. Instead of a logical condition, a procedure can be specified. In this way pre-conditions and post-conditions for the procedure can be described. It is important to realize that this way of reasoning can be non-deterministic when multiple rules match the current situation but have different effects.

Rule-based systems commonly use a flat representation of the rules, which can easily lead to a disorganized model. However, it is possible to add frame-like constructs to organize these rules.

SWRL SWRL (Semantic Web Rule Language) is a language that is meant to be used as a rule-based extension on OWL [Horrocks et al., 2004]. Its main purpose is to enable the specification of pre-conditions and post-conditions of web services.

Transaction Logic Transaction logic [Kifer, 2005] is a comprehensive rule-based language based on first-order logic. It is exceptional in that it defines clear semantics for executing state-changing procedures. In contrast with SWRL, Transaction Logic allows the user to program procedures using a logic notation.

Conclusion

It table 7.1 we compare the different ontology description approaches. To assess the epistemological qualities of each of the logics we use the domain of service provisioning using web services. For some languages we have not been able to find scientific evidence for their performance on all criteria. In those cases no rating has been given.

The choice of knowledge representation technique for representing ontologies has significant consequences for the suitability of a semantic service method. This is because the choice for a certain knowledge representation logic prescribes the mechanisms used for service description and discovery.

In the domain of formulating service descriptions it is important to realize that services achieve a transformation in the world. Description of this dynamic aspect is of crucial importance. Based on this structural,

	Classic	Sem. Nets		Frame-based		Description		Rule-Based	
	First Order Logic	RDF	RDF Schema	F-Logic	WSML Flight	OWL DL	WSML DL	SWRL	Transaction Logic
Implementation									
efficiency	-	+	+	+	+	+	+		
Logical									
clear semantics	+	-	-	+	+	+	+	-	+
expressivity	+	-	-	□	□	□	□	□	+
sound inference	+	-	-	+	+	+	+	+	+
Epistemological									
structure	-	-	-	-	-	-	-	+	+
modular	-	+	+	+	+	+	+	-	-
primitives	+	+	+	+	+	+	+	+	+
Conceptual									
concise	-	+	+	+	+	□	□	-	-

Table 7.1: Assessment of knowledge representation techniques.

epistemological, criterion a rule-based language would be the best choice for modeling service behavior. A major problem is that no full implementation exists for both the researched rule-based languages, SWRL and Transaction Logic.

7.4 Building Ontologies

In [Uschold and Grüninger, 1996] a methodology is provided for building ontologies. We briefly discuss the steps in this methodology. The following steps are discerned: determining purpose and scope, building the ontology and evaluation.

Determining purpose and scope. The first step is to establish the purpose and the scope of the ontology, it should be clear why the ontology is being built and what its intended uses are.

Building the ontology. The second step is to build the ontology. This step consists of three sub-steps: capture, coding and integration. During the capture step the key concepts and relationships in the domain are identified. A clear textual description for each of the concepts is agreed upon and names are assigned to the concepts and relationships.

Uschold et al. recommend a middle-out approach rather than a top-down or bottom-up approach when determining the concepts that should go in the ontology. The middle-out approach entails finding the central terms in the ontology and to expand from there. Uschold et al. argue that bottom up approaches tend to lead to overly detailed ontologies where no useful abstractions are found. Top-down design can lead to ontologies that are based on theoretical concepts that may have little affiliation with reality. The middle-out approach guarantees that the important concepts are included and, therefore, that the ontology will be relatively stable as it is grounded in reality.

Next, in the coding step, a suitable ontology description language is chosen. Then the ontology is coded in the ontology description language.

During the integration step (which might occur simultaneously with the other two steps) related ontologies are identified and it is decided if portions of these related ontologies can be reused.

Evaluation. The third and final step of ontology building is the evaluation step. During this step the suitability of the proposed ontology for its intended uses is determined.

Uschold et al. do not say anything about the level of detail that should be employed when specifying associations and the relations between them. It seems to be a reasonable approach to define only those relations that will be necessary to support the intended purpose.

Tools

Building ontologies and writing logical statements about the ontology by hand can be a difficult task, especially if the ontology description language has a syntax that is hard to work with. In such cases one can use modeling tools that assist the user. Additionally, tools can provide overview of an ontology with a graphical user interface. This way it might be easier to define relations between concepts.

Protégé [Gennari et al., 2003] is a tool designed for developing knowledge-based systems and consequently it can also be used for modeling ontologies. Recent versions of Protégé are based on a plug-in architecture so that external developers can contribute additional functionality . The user interface displays a list of tabs where each tab offers certain functionality. These tabs are implemented as plug-ins. Support for working with OWL, RDF and OWL-S is offered using such an externally developed plug-in [Elenius et al., 2005].

Protégé presents a graphical user interface with which an ontology can be built. The user can enter classes, properties, constraints and instances. Plug-ins offer support for converting this internal representation to external file formats. Additionally, the tool supports the visualization of ontologies. Plug-ins for specific ontology languages can offer specialized support for the features of that language.

Standard Ontologies

Rather than modeling an ontology for each separate web service, which introduces a huge risk that incompatibilities arise, it can be advantageous to make use of industry standard ontologies. Using a standard ontology can reduce development time. This is because it is not necessary to do extensive domain research before a service can be developed and marketed. Additionally if multiple parties reuse the same ontology, the potential for compatibility problems that hinder communication about a service is much lower.

In contrast a common ontology may lack focus and try to include every possible detail from the domain. Such a model can become hard to understand and it could be difficult to express a specific service in the generic domain model. Further, it may then be necessary to adapt the design of the service to fit in the standardized ontology, which may not be what we want.

In the telecom world the TeleManagement Forum's Shared Information/Data model (SID) offers a standard ontology for provisioning telecom services [Strassner et al., 2003]. The SID is provided as a set of UML models that capture domain knowledge. All information entities that are necessary to support telecom operations are defined in the model. The number of classes that describe various aspects of telecom operations is very large and attributes and associations between the classes are defined as well. The model is meant to be used with Model Driven Architecture (MDA) tools that are able to build application code based on a (UML) model of the application domain. The SID model is very complete and very complicated

and it is hard to become familiar with it. An important benefit of using this model is that the terminology in the telecom world is clearly defined.

A problem that may arise when using this model to describe services is that the model is represented in UML. If one wants to employ a different ontology modeling language to describe a service than it is necessary to convert the SID model semantics to the target ontology language. These ontology languages can be quite different from the UML. More importantly, UML does not have formal semantics comparable to the logic based ontology modeling languages.

7.5 Heterogeneity in ontologies

A big problem in enterprise application integration is that of data integration. When two applications have to be integrated it often becomes clear that the data models used by the applications are quite different. This can occur even if the applications to be integrated have a similar purpose. In [Batini et al., 1986] an overview of data modeling conflicts is provided.

The conflicts can be divided in naming conflicts and structural conflicts. Naming conflicts arise from different people using the same term to mean different things or, reversely, from people using different terms to mean the same thing. Structural conflicts arise when conflicting modeling constructs are used to represent the same concept.

As we said before, in order to enable communication between different parties it is crucial that both parties have a shared understanding of the used terms. It is therefore required that both parties use a common ontology. In a heterogeneous environment such as the internet it is unlikely that two parties will come up with exactly the same ontology.

A method for semantic web services, therefore, has to provide a way to bridge heterogeneity in ontological descriptions. Data integration is a rather old problem, especially in the context of relational databases, and a lot of material on the subject is available. The main problem in data integration is to identify mappings between different ontologies. From the literature we conclude that currently no reliable fully automatic method exists for identifying these mappings. It is unavoidable that the involvement of domain experts during the integration process is required.

Ontology modeling languages are generally more expressive than the entity-relationship model, they probably have even more potential for data integration conflicts. A method for semantic web services therefore needs to recognize that differentiation in ontologies will occur and that a method for bridging ontological differences is required.

8 Discovery

In this chapter we introduce a number of approaches for realizing semantic service discovery. For each of these approaches we discuss their benefits and drawbacks. In the next section we discuss the criteria by which a discovery method should be judged. In the subsequent sections we discuss keyword-based, description logic-based and rule logic-based discovery.

8.1 Preciseness of Service Descriptions

In semantic web service architectures, discovery is the task of matchmaking between providers and requesters. The typical use case is that service providers create descriptions of their services. Subsequently these service advertisements are registered with a central matchmaking party. When a service requester looks for a service, he or she formulates a description of the desired service. The request is then passed on to the central matchmaking party. The matchmaking party searches through all registered service advertisements and determines which providers could fulfill the request. A list of matching providers is returned to the requester.

As discussed in [Preist, 2004], a service description usually provides an abstract description of a service with which countless different concrete service instantiations can be described. The concrete instances differ in terms of the parameters used for a service. Often a service is specific to the location of the customer, or there are multiple variants of a service of which the customer can select one. Some parameters, such as price, might even be negotiable.

A concrete service is an instantiation of the service that performs the service tasks. A concrete description contains details about the involved parties and the message content that is exchanged. Usually a concrete service is not advertised as such, but rather an abstract service is used for that purpose.

An abstract service describes a set of concrete services. It describes a collection of concrete services that a service provider is willing to offer. An abstract service can be defined by listing all possible concrete services, but typically it is specified in a formal way using an ontology.

It is hard to accurately describe an abstract service in a formal way. As indicated by [Hepp, 2006] the information space needed to fully describe all possible service instances can get very big and can be highly dynamic. Consider the following example: a user may specify that he or she is interested in purchasing a specific book. To deliver the service it is not enough to know that a service sells books and that the user is looking for a book.

This namely does not guarantee that the specific book the user wants can be purchased at the specific web service. Oren et al. describe a related problem in the domain of the sale of ADSL lines [Oren et al., 2004a]. In this domain, the availability of the service depends on the phone number of the customer. The problem with this is that the list of valid phone numbers is enormous and might change rapidly.

These concerns are addressed in [Preist, 2004] by means of the correctness and completeness criteria. A complete description describes all possible concrete services. For a description to be correct, it is required that each service that is described by abstract service description is also offered as a concrete service. A discovery mechanism preferably enables correct and complete discovery of services.

8.2 Keyword-Based Discovery

Perhaps the simplest approach to service discovery is to use keyword-based matching. In this approach, service providers describe their service by associating (possibly standardized) keywords with it. The UDDI standard discussed in section 5.1 uses this approach. A requester discovers services by specifying keywords that apply to the desired service. The approach can be extended by allowing boolean operations (such as 'AND', 'OR' and 'NOT') on keywords to make matches more accurate.

The approach is simple, but it has significant drawbacks as it is hard to be precise and complete in keyword descriptions. The reason for this is that keywords derived from natural language are often ambiguous. Furthermore, an important point in service description is that services achieve a transformation in the world. Specifying such a transformation using only keywords seems impossible. If keywords are sufficiently standardized then the technique could be more successful. This, however, requires that all possible services and their associated keywords are predefined. This is unfeasible because it is inflexible and therefore goes against the service-oriented architecture's principles. A keyword-based search is, therefore, not acceptable for automated service discovery.

8.3 Description Logics-Based Discovery

More sophisticated methods of discovery perform matching on ontology-based descriptions of services. These methods have the potential to be more precise. The main benefit of these methods is that the matching can be done on the actual service description instead of just on meta-data. In this section we discuss a method proposed by [Grimm et al., 2004] for discovery based on description logics (see chapter 7).

The method is based on describing services as concepts in a description logics ontology. A connection service could be represented as a class (e.g., *connectivity_offer*). This class could have attributes that indicate the properties of the connection (such as *speed* and *delay*). A *technology* role could be defined that connects the *DSL* class to the *connectivity* class. A service provider could describe their services in this way and register the description with a directory.

The service requester makes a similar description of the desired service, usually in a more generic way. The requester might specify a class *connectivity_request* with a value of *1mbs* for speed while leaving unspecified the delay and realization technology. A centralized registry then determines which provider descriptions match the requester's description.

In [Grimm et al., 2004] a number of methods for matching these kinds of descriptions are presented. A straightforward method is to check if the advertisement and the request are logically equivalent. Two concepts are considered equivalent if they have the same values for all properties. An exact match, however, is expected to be rare, since it is more likely that the advertisement is more specific than the request or vice-versa. In our example, we see that the requester is only interested in the speed of the connection, and the advertisement is much more detailed. A matcher can then check that the advertisement *subsumes* the request, or vice-versa.

In description logic notation, a subsumption condition that checks if the service offer is more specific than the request is written as follows:

$$\iota : (connectivity_{offer} \sqcap \neg connectivity_{request})\, is\, unsatisfiable$$

Another, very weak, check is to see if the advertisement and the request are not disjoint, i.e., whether it is possible to define an instance which is a member of both the advertisement and the request class. According to these checks a matcher can rank the results and return the list of matches to the requester.

This method of service matching adds a semantic aspect to the discovery process and is relatively easy to implement. Description logic reasoners are generally available and the method is conceptually intuitive. The discovery scenario discussed in this paragraph has been executed in the OWL-DL language and is provided as an appendix to this report.

The discovery approach detailed in this section is however not without its drawbacks. A practical problem is that in order to generate reasonably correct matches, this technique requires one description to be more specific than the other. It is likely, however, that in many cases the advertisement is in some parts more specific than the request and the other way around for other parts.

A more fundamental problem with this approach is that it does not acknowledge that services achieve a transformation in the real world, a transformation that has preconditions and effects. In description logic-based modeling, services are static concepts which can be described as if they are physical objects. As a consequence, a large opportunity for ambiguity in service description arises. A discovery mechanism based on description logic is, therefore, likely to return inaccurate results.

8.4 Rule Logic-Based Discovery

The most natural form of discovery seems to be a method that enables a requester to formulate the current state of the world, and the desired end-state. The discovery component then returns the providers that can achieve the desired transformation. An approach like this is promised in [Kifer et al., 2004]. Transaction logic makes it possible to reason about

modifications in knowledge bases. With transaction logic, preconditions and post-conditions can be described and reasoned with.

An example of how semantic service discovery can be performed is provided in [Kifer et al., 2004]. The method presented requires all requesters and providers in the application domain to agree on a set of discovery goals. An example of such a goal, relevant in our case description (see chapter 2), could be *connect_end_user*. This goal represents that an end user of an internet connection will be connected to the network of an ISP. The goal contains parameters to identify which end-user has to be connected. A registry maintains a list of services that achieve the goal, each with their own interface. Associated with each service is a mediation component that can translate from the agreed domain specific goals to the service specific calling interface. The mediator contains a hard-coded list of services of which it knows the calling interface. All intelligence is in the mediator component, which can take a standardized request and pass it on to a specific service it knows about.

Unfortunately, this approach is limited and does not improve on the current approach to web services. The method depends on a standardized list of operations. Mapping between the standardized operations and the services must be performed manually. It is not possible to perform semantic queries on post-conditions. An example of such a query could be 'return all services which at least result in the predicate *connection(customer , internet_service_provider)* to be true'

Furthermore, with the transaction logic approach it becomes very complex to develop service descriptions and to check if they meet certain goals. The main strength of transaction logic lies more in the area of implementing logic-based programs than in the area of discovering web services.

The implementation support for transaction logic is limited. The only implementation, the Flora-2 system [Yang et al., 2003], does not support the important hypothetical update operators. We conclude that transaction logic, in its current state, is not a good basis for semantic service discovery.

Part IV

Evaluation and Conclusions

9 Evaluation of Existing Methods

In the previous chapters we discussed the theory behind semantic service provisioning and developed requirements for assessing methods. In this chapter we evaluate and compare two concrete methods for realizing semantic web applications. In the literature, OWL-S and WSMO methods are proposed as methods with which semantic web services can be realized. We first review both methods separately and after that we compare them.

We discuss the methods using the subdivision of issues around semantic service provisioning introduced in section 6.3. The concerns are subdivided in the *service* level, the *communication* level and the *delivery* level. We further look at the practical applicability and the tool support for each method.

In section 9.4 we evaluate both methods against the list of requirements set apart in section 6.4. We conclude this chapter with an assessment of the applicability of the methods.

9.1 OWL-S

The first method which we have investigated is the OWL-S method. OWL-S, as described in [Martin, 2004, Martin et al., 2004], is a method for describing web services; it is built on top of the OWL ontology description language and hopes to enable automated web service discovery, execution, composition and inter-operation.

Business Service Level At the service level we look at the support the method offers for modeling processes, for bundling services and for representing service level agreements. OWL-S makes no distinction between business services and application services. A service in OWL-S is an application service that might realize a business service, but this is not enforced.

In OWL-S, the essence of the service is represented by two components: the service profile and the process model. The service profile is meant to advertise what the service does. It contains a description of the incoming and outgoing parameters, preconditions and results of using the service. Processes are thus described in a declarative fashion. Additionally it is possible to assign a category to the service in a taxonomy of services.

Bundling of services is supported through the process model, which supports aggregating services with a BPEL-like (see section 5.2) control language.

OWL-S offers no support for specifying or monitoring compliance with service level agreements.

Communication Level At the communication level we look at ontology representation, discovery and representation of message patterns.

OWL-S uses OWL-DL, a description logic based ontology language, for representing the world (see section 7.3). Additionally OWL-S allows authors to describe services using the SWRL rule-based language. Using this language, pre-conditions and post-conditions on services can be represented. These pre-conditions and post-conditions detail the input and output parameters and the relations between the two.

OWL-S does not provide a way to deal with ontology inconsistencies that arise in a heterogeneous environment.

According to [Martin, 2004], discovery can be done in two ways. Firstly the description logic subsumption approach, which has been discussed in section 8.3. This approach is most commonly mentioned in OWL-S related publications. Secondly, it is in principle possible to perform reasoning on SWRL-based descriptions of transformations that services realize. However, no SWRL reasoner implementations are available and it is unclear how the SWRL-based descriptions can be integrated with the ontology-based description of the domain. The OWL-S specification itself does not prescribe a technique for matching service descriptions.

Message exchange patterns can be derived from the input and output parameters. OWL-S provides a mechanism to derive a WSDL specification from an OWL-S service description.

For composite services, the message exchanges can be derived from the process model description. As mentioned in [Preist, 2004], this means that OWL-S does not maintain a separation between the implementation of services and the calling interface. In many cases this separation is desirable, otherwise internal information is exposed, and the interface of the service becomes dependent on the concrete implementation.

Delivery Level The main issue at the delivery level is to determine the exact parameters that are to be used for the service contract. This is necessary to refine the results of the discovery phase.

The OWL-S specification does not specify how service descriptions that are retrieved from the discovery phase can be refined. Neither is it clear how a contract for service delivery is arranged. Similarly, there is no support for negotiating parameters such as price.

Practical Applicability and Tool Support Next to the theoretical considerations about whether a method fulfills the requirements, we also look at the practical applicability of each method. In particular we look at the tool support that exists. Furthermore we investigate if the method has been used successfully in practice.

Tool support for OWL-S comes mainly in the form of OWL ontology editors. The Protégé editor [Gennari et al., 2003] is a featureful ontology editor that has support for graphically representing and editing ontologies.

Tool support for creating OWL-S service descriptions is very limited, but there is a plug-in available for Protégé [Elenius et al., 2005].

So far no practical implementations of OWL-S have been demonstrated. In [Balzer et al., 2004] a number of practical problems that arise when trying to semantically describe OWL-S services are discussed. The authors mention ontological inconsistencies in the OWL-S service model, lack of support for discovery and lack of tool support.

9.2 WSMO

Another method that is proposed in the literature is WSMO. In this section we investigate this method.

WSMO is an implementation of the Web Service Modeling Framework proposed in [Fensel and Bussler, 2002]. The WSMO (Web Service Modeling Ontology) working group started out in early 2004 and its goal is to work towards further standardization in the area of semantic web services and to design a common architecture platform. To this end the working group formulated the WSMO ontology that describes what components are required for building semantic web services. The WSML (Web Services Modeling Language) sub-project developed a formal notation for describing the elements in the WSMO ontology. The WSMX (Web Service Modeling Execution Environment) project strives to build an implementation of the WSMO platform that can be used for discovery and invocation of semantic web services. An overview of the WSMO approach is given in [Oren et al., 2004b].

Business Service Level WSMO does not differentiate between application services and business services, but mainly describes application services. Services in WSMO are represented using the *web service* notion. Another important notion is that of *goals*, which are descriptions of objectives that a client of a web service may have. Web services are described in terms of the goals they achieve (the capability). The goals are described in terms of the preconditions and post-conditions in an ontology. According to [Fensel and Bussler, 2002], goals should be specified separately from web service descriptions because a web service can satisfy multiple goals and multiple web services might be able to achieve the same goal.

At the moment WSMO does not offer support for creating bundles of services or for representing or negotiating service level agreements.

Communication Level In the WSMO framework, ontologies are represented using the WSML language. WSML has several profiles but the commonly recommended profile, WSML-Flight is based on a frame-based logic (see section 7.3). WSMO acknowledges that in a heterogeneous environment it is likely that different ontologies arise. To address this problem, WSMO suggests the use of a mediator component, which is able to map between ontological descriptions. WSMO, however, does not specify how such a component could be built. As discussed in section 7.5, resolving structural integration conflicts in an expressive ontology is likely to be very difficult.

Discovery for WSMO is discussed in [Keller et al., 2004]. In the article, variants of the techniques discussed in chapter 8 are applied to the WSMO framework. The WSMO project has not yet adopted any of these techniques.

In [Roman and Scicluna, 2006] it is discussed how message exchange patterns can be represented in WSMO. In this article, it is argued that message exchange patterns can be represented using abstract state machines as described by [Börger and Stärk, 2003].

Delivery Level The WSMO framework does not make a distinction between an abstract service and a concrete service. As a result there is no mechanism available to refine the results of the discovery process either. Negotiation support is not provided either.

Practical Applicability and Tool Support WSMX is an implementation of the WSMO and will eventually offer support for the complete WSMO conceptual model, including discovery, mediation, selection and invocation of services. The WSMX group strives to build an enterprise application architecture for semantic web services. WSMX is developed in an iterative manner, which means that current early versions have only very limited functionality. Currently it is possible to specify goals and to associate web services with them [Oren et al., 2004b].

Initial experience with the WSMX framework shows that the software is in a very early phase where not all components work together. In its current state functionality is rather limited. The main result available right now is an architecture on which components can be built.

9.3 SWSF

The SWSF (Semantic Web Service Framework) method is another method for realizing semantic web services, introduced in [Battle et al., 2005]. Analogous to WSMO, the SWSF method consists of an ontology language, SWSL, and a service ontology, SWSO. The SWSF initiative places an emphasis on the representation of process models.

Business Service Level Similar to WSMO and OWL-S, SWSF does not distinguish between application services and business services. SWSF does support the notion of composite services which are build up from atomic services. Atomic services are represented using their inputs, outputs, preconditions and effects. Composite services are defined using a process model, specifying which messages are sent and received and in what order atomic services are called. This way, SWSF supports representing business processes and the bundling of services. SWSF offers no support for representing and managing service level agreements.

Communication Level SWSF uses a rule based language, SWSL-Rules, to model the application domain and to represent services. The rule language is defined by mapping its constructs on a first order language, SWSL-FOL. A rule based language is used to enable implementation of efficient

reasoning software. SWSF does not address the problem of ontological heterogeneity that was discussed in section 7.5.

The SWSF method prescribes logic-based matchmaking, as discussed in section 8.4, for discovery of services.

Message patterns between parties can be represented in SWSF process models. The SWSO ontology contains constructs for sending and receiving messages. There are no provisions in SWSF for integrating message flows between parties that expect different message exchange patterns.

Delivery Level SWSF recognizes the problem of refining abstract service descriptions. The approach recommended by the method is to solve the problem at the application level. This entails creating domain specific application services for searching and refining application and business services. There is no generic negotiation support in the SWSF method. This hinders flexibility, because each party is free to choose their own negotiation protocol.

Practical Applicability and Tool Support The SWSF specification is quite recent and at this time no tool support is available. The specification, especially the ontology language, is also quite complex. This complexity stems from the use of first order logic which, as discussed in section 7.3, is so expressive that it is not decidable. The first-order logic part of the specification will therefore be hard to implement. The rule-based ontology language is probably feasible to implement. The Flora-2 rule logic programming system [Yang et al., 2003], some developers of which also partake in SWSF project, demonstrates that such an implementation is possible.

9.4 Evaluation and Comparison

In table 9.1 the methods for semantic web services are compared with each other and with the requirements introduced in section 6.4. We see that no method fulfills all the requirements. A big issue with two of the methods is that no commitments are made as to what algorithms should be used for discovery. It is essential that a method is clear about this subject since discovery is an essential component of a semantic web service architecture. The discovery mechanism influences the way services are described. Without clarity about this subject, the feasibility of the entire method becomes unclear.

Furthermore, no method supports the activities in the management layer of the service-oriented architecture. This means that no explicit support is provided for specifying and negotiating service level agreements.

An important problem with all of the proposed frameworks is that they are very theoretical and experimental. No working practical applications have been demonstrated with either framework for semantic web services. We believe that there are both theoretical and practical reasons for this.

A problem we see with the OWL-S language is that it uses different languages for modeling the ontology and for modeling preconditions and post-conditions. As a result, it is very hard to work with OWL-S service

Requirement	OWL-S	WSMO	SWSF
1: Process modeling	Yes	Yes	Yes
2: Service Level Agreements	No	No	No
3: Bundling of services	Yes	No	Yes
4: Representation of ontologies	Yes	Yes	Yes
4a: Dealing with heterogenity	No	Partial[a]	No
5: Discovery of services	Partial[b]	Partial[c]	Partial[d]
5a: Ontological discovery	Partial	Partial	Partial
5b: Transformation based discovery	No	No	Partial
6: Representation of message patterns	Partial[e]	Yes	Yes
7: Refinement of service parameters	No	No	No
7a: Negotiation of parameters.	No	No	No

[a] The WSMO framework acknowledges the problem but does not offer a concrete solution.
[b] OWL-S does not specify a concrete discovery mechanism, several possible approaches are mentioned in the literature.
[c] Several approaches are possible, but no commitment to a specific method has been made.
[d] The proposed method, rule logic-based discovery, has major drawbacks as discussed in section 8.4.
[e] Message patterns definition is mixed with the bundling specification, limiting flexibility.

Table 9.1: Comparison of approaches for semantic web services.

descriptions and it is unclear how the preconditions and post-conditions interact with the domain model.

The result of this evaluation is that neither of the methods meet the requirements for enabling semantic web services. As a result we must conclude that no method is currently suitable for realizing the service-oriented architecture vision described in chapter 4. In the next chapter we summarize the results from this report. Additionally we recommend steps that can be taken towards realizing semantic service provisioning.

10 Conclusions and Recommendations

In this chapter we recapitulate the findings of this report with regard to the state of the art in semantics in service-oriented architecture. Additionally we make recommendations for further research. In section 10.1 we present the results of this research. In section 10.2, we compare the ideal situation to the current situation. Based on this comparison, we determine what concrete steps can be taken towards a realizable and desirable result.

10.1 Findings

In this report we have investigated the semantic aspects related to the service-oriented architecture vision. Our goal was to find out what current possibilities exist to enable automatic service discovery and delivery. Our first step was to define the notion of a service. We consider a service to be made up of two parts: a business part and a technology part. The business side of the service concept dictates that a service represents an activity that has a value to the customer. The technological side of the service concept is concerned with delivering this value in an automated way.

We then treated the service-oriented architecture vision and discussed its motivation and promises. The main promise of the service-oriented architecture is that information system support is separated in components. These components advertise their services in a centralized directory. When an application needs a certain service, this service is looked up in the directory and used automatically. The benefit of organizing IT support in this way is that it is easy to modify, replace and introduce components without having to change all applications.

Web service technology is widely regarded as a means to realize service oriented architectures. We have investigated if web service technology is usable for this purpose. We found that web service technology only supported syntactic descriptions and ignored the semantic aspect of service descriptions. This means that web service technology by itself cannot support automated discovery or delivery.

This conclusion led us to investigate what aspects play a role in service provisioning in service-oriented architectures. We built a conceptual framework that recognizes the structures and processes that surround service provisioning. Several concerns related to service description, communication and delivery were identified. We further researched and evalu-

ated approaches towards solving the essential problems of ontologies and discovery of services.

Based on our conceptual framework, we were able to derive requirements for methods that strive to enable semantic service provisioning. Three of such methods are proposed in the literature: OWL-S, WSMO and SWSF. We evaluated these methods with the aid of our framework and the requirements.

The conclusion of the evaluation is that the proposed methods do not meet the requirements. We identified three major problems with the proposed methods. Firstly, discovery of services is not done based on the transformation of the world achieved by the service. Secondly, no plausible solution for the problem of heterogeneous data representations is given. Thirdly, no working implementations have been demonstrated.

Consequently, we must conclude that it is not possible to realize the entire service-oriented architecture vision with the proposed methods for semantic web services. However, in the next section we introduce a viable strategy for realizing a useful subset of the service-oriented architecture vision.

10.2 Recommendations

Knowing that, at this time, it is not possible to describe and build services in such a way that they can be fully automatically found and used, it becomes relevant to investigate what parts of the semantic service-oriented architecture can be realized in the near future. In this section we investigate what useful results can be achieved with current technology. Afterward we look at the steps that can be taken towards realization of the full service-oriented architecture vision.

Realizable Goals

An important realization is that integrating business processes between companies requires the integration of data models as well. Processes can only be integrated if all parties agree about the meanings of the terms used. A shared data model is therefore a good starting point for a semantic service-oriented architecture, as it fulfills the role of the ontology in the framework introduced in chapter 6.

We recommend that companies that want to partake in an industry-wide semantic service-oriented architecture start with jointly developing a shared data model of their operating domain. The data model should include the essential abstractions and the instance data that must be shared. Sharing a common data-model and its instance data can be realized by utilizing distributed relational database technology. This technology is attractive because it is mature and well understood.

The shared data model can be built by integrating the different company specific schemas or by adopting an already existing standardized model. In the telecom world, the SID data model (see section 7.4) can be used for this purpose.

Components that offer services can be built on top of this shared data model. Their behavior can be documented using the transaction logic formal notation (transaction logic is discussed in section 7.3) or an easier to use and more informal language such as UML activity diagrams.

If services are described using this method then their fundamental characteristics are described. With this information it will be easier for application developers to determine what a service does. This will make it easier to make use of a service. As a result, the time required to link applications and components will be reduced and costs can be saved.

We recommend that IT-dependent industries investigate the possibilities of developing such a distributed data model in which operations can be offered. Integrating services will still be a manual activity, but since services can be described in a formal way it will be clear what each service does, thereby reducing costs.

Future Goals

The most important future goals for semantic service-oriented architectures are those of automated application service discovery and invocation. We think that the best way to offer this functionality is to describe the services using transaction logic. Using this language, preconditions and post-conditions can be modeled and therefore the transformation that the service achieves in the world is known. Transaction logic can also be used as a declarative programming language. The most important future goal is to realize a full implementation of transaction logic on top of a distributed relational database system. When this implementation exists services can be discovered based on the transformation they achieve in the world.

Another important remaining task is that of representing, negotiating and monitoring service level agreements. Initially, the best way to offer support for service level agreements could very well be to include industry wide standardized service level packages in the data model. Negotiation of service level agreements would then become unnecessary. In the future, research in multi-agent systems might enable software systems to autonomously negotiate favorable deals. This kind of research is discussed in [Wooldridge, 2002, pp. 129–163].

We conclude that although the technology for building semantic service-oriented architectures has not been fully realized yet, it is possible to make important steps right now. The first step companies should make is to prepare for the semantic service-oriented architecture by developing a domain model of their industry. This shared model will be a stepping stone for when all the required technology becomes available.

Bibliography

[Baader and Nutt, 2003] Baader, F. and Nutt, W. (2003). Basic description logics. pages 43–95.

[Baida et al., 2003] Baida, Z., Akkermans, H., and Gordijn, J. (2003). Serviguration: towards online configurability of real-world services. In Sadeh, N. M., Dively, M. J., Kauffman, R. J., Labrou, Y., Shehory, O., Telang, R., and Cranor, L., editors, *ICEC*, pages 111–118. ACM.

[Balzer et al., 2004] Balzer, S., Liebig, T., and Wagner, M. (2004). Pitfalls of OWL-S: a practical semantic web use case. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 289–298, New York, NY, USA. ACM Press.

[Batini et al., 1986] Batini, C., Lenzerini, M., and Navathe, S. B. (1986). A comparative analysis of methodologies for database schema integration. *ACM Comput. Surv.*, 18(4):323–364.

[Battle et al., 2005] Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., Mcilraith, S., Mcguinness, D., Su, J., and Tabet, S. (2005). Semantic web services framework (SWSF) overview. Technical report.

[Berners-Lee, 2001] Berners-Lee, T. (2001). The semantic web. *Scientific American*, 284(5):28.

[Börger and Stärk, 2003] Börger, E. and Stärk, R. (2003). *Abstract State Machines : A Method for High-Level System Design and Analysis*. Springer.

[Brickley and Guha, 2004] Brickley, D. and Guha, R. V. (2004). RDF vocabulary description language 1.0: RDF-Schema. W3c recommendation, World Wide Web Consortium.

[Congram and Epelman, 1995] Congram, C. and Epelman, M. (1995). How to describe your service: An invitation to the structured analysis and design technique. *International Journal of Service Industry Management*, 6(2):6–23.

[de Bruijn et al., 2005] de Bruijn, J., Lausen, H., Krummenacher, R., Polleres, A., Predoiu, L., Kifer, M., and Fensel, D. (2005). The web service modeling language WSML. WSML final draft, Digital Enterprise Research Institute. `http://www.wsmo.org/TR/d16/d16.1/v0.2/`.

[Duke et al., 2005] Duke, A., Davies, J., and Richardson, M. (2005). Enabling a scalable service-oriented architecture with semantic web services. *BT Technology Journal*, 23(3):191–201.

[Elenius et al., 2005] Elenius, D., Denker, G., Martin, D., Gilham, F., Khouri, J., Sadaati, S., and Senanayake, R. (2005). The OWL-S editor – a development tool for semantic web services. *Lecture Notes in Computer Science*, 3532:78–92.

[Fallside, 2001] Fallside, D. C. (2001). XML-Schema part 0: Primer. Technical report.

[Fensel and Bussler, 2002] Fensel, D. and Bussler, C. (2002). The web service modeling framework WSMF. *Electronic Commerce Research and Applications*, 1(2):113–137.

[Gennari et al., 2003] Gennari, J. H., Musen, M. A., Fergerson, R. W., Grosso, W. E., Crubzy, M., Eriksson, H., Noy, N. F., and Tu, S. W. (2003). The evolution of protégé: An environment for knowledge-based systems development.

[Gozdecki et al., 2003] Gozdecki, J., Jajszczyk, A., and Stankiewicz, R. (2003). Quality of service terminology in IP networks. *Communications Magazine, IEEE*, 41(3):153–159.

[Grimm et al., 2004] Grimm, S., Motik, B., and Preist, C. (2004). Variance in e-business service discovery. In *Semantic Web Services: Preparing to Meet the World of Business Applications*.

[Guarino and Giaretta, 1995] Guarino, N. and Giaretta, P. (1995). Ontologies and knowledge bases: Towards a terminological clarification. In Mars, N. J. I., editor, *Towards Very Large Knowledge Bases*, pages 25–32. IOS Press, Amsterdam.

[Guizzardi, 2005] Guizzardi, G. (2005). *Ontological Foundations for Structural Conceptual Models*. PhD thesis, University of Twente.

[Hepp, 2006] Hepp, M. (2006). The true complexity of product representation in the semantic web. In *Proceedings of the 14th European Conference on Information Systems*.

[Horrocks et al., 2004] Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B., and Dean, M. (2004). SWRL: A semantic web rule language combining OWL and RuleML. W3c member submission, World Wide Web Consortium.

[Keller and Ludwig, 2003] Keller, A. and Ludwig, H. (2003). The WSLA framework: Specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management*, 11(1):57–81.

[Keller et al., 2004] Keller, U., Lara, R., Polleres, A., Toma, I., Kifer, M., and Fensel, D. (2004). WSMO web service discovery. WSMO Working Draft.

[Kifer, 2005] Kifer, M. (2005). Requirements for an expressive rule language on the semantic web. In *Rule Languages for Interoperability*. W3C.

[Kifer et al., 2004] Kifer, M., Lara, R., Polleres, A., Zhao, C., Keller, U., Lausen, H., and Fensel, D. (2004). A logical framework for web service discovery. In *ISWC 2004 Workshop on Semantic Web Services: Preparing to Meet the World of Business Applications*, volume 119, Hiroshima, Japan. CEUR Workshop Proceedings.

[Kifer and Lausen, 1989] Kifer, M. and Lausen, G. (1989). F-Logic: A higher-order language for reasoning about objects, inheritance, and scheme. In *SIGMOD Conference*, pages 134–146.

[Kotler and Armstrong, 2005] Kotler, P. and Armstrong, G. (2005). *Principles of Marketing*. Prentice Hall, 11th edition edition.

[Lovelock, 1992] Lovelock, C. H. (1992). *Managing Services: Marketing, Operations, and Human Resources*. Prentice Hall College Div.

[Manola and Miller, 2004] Manola, F. and Miller, E. (2004). RDF Primer, W3C Recommendation. Technical report.

[Martin, 2004] Martin, D. (2004). OWL-S: Semantic Markup for Web Services. Technical report. `http://www.daml.org/services/owl-s/1.0/owl-s.html`.

[Martin et al., 2004] Martin, D. L., Paolucci, M., McIlraith, S. A., Burstein, M. H., McDermott, D. V., McGuinness, D. L., Parsia, B., Payne, T. R., Sabou, M., Solanki, M., Srinivasan, N., and Sycara, K. P. (2004). Bringing semantics to web services: The OWL-S approach. In Cardoso, J. and Sheth, A. P., editors, *SWSWPC*, volume 3387 of *Lecture Notes in Computer Science*, pages 26–42. Springer.

[McGuinness and van Harmelen, 2004] McGuinness, D. L. and van Harmelen, F. (2004). OWL web ontology language overview. W3c recommendation, World Wide Web Consortium.

[McIlraith and Son, 2002] McIlraith, S. and Son, T. (2002). Adapting golog for composition of semantic web services. In *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning*, pages 482–496.

[Microsoft Corporation, 2006] Microsoft Corporation (2006). What is .NET?

[Milham, 2004] Milham, D. (2004). etom – public B2B business operations map (bom) application note c: An initial proposal for the scope and structure of ict business transaction.

[Oren et al., 2004a] Oren, E., Wahler, A., Schreder, B., Balaban, A., and Zaremba, M. (2004a). Demonstrating WSMX - least cost supply management. In *Proceedings of the Workshop on WSMO Implementations*.

[Oren et al., 2004b] Oren, E., Zaremba, M., and Moran, M. (2004b). Overview and Scope of WSMX. WSMO Working Draft v01.

[Orlowska et al., 2003] Orlowska, M. E., Weerawarana, S., Papazoglou, M. P., and Yang, J., editors (2003). *Service-Oriented Computing - ICSOC 2003, First International Conference, Trento, Italy, December 15-18, 2003, Proceedings*, volume 2910 of *Lecture Notes in Computer Science*. Springer.

[Parasuraman et al., 1984] Parasuraman, A., Zeithaml, V. A., and Berry, L. L. (1984). *A Conceptual Model of Service Quality and Its Implications for Future Research (Report No 84-106)*. Marketing Science Inst.

[Preist, 2004] Preist, C. (2004). A conceptual architecture for semantic web services. In *International Semantic Web Conference*, pages 395–409.

[Preist et al., 2005] Preist, C., Cuadrado, J. E., Battle, S., Grimm, S., and Williams, S. K. (2005). Automated business-to-business integration of a logistics supply chain using semantic web services technology. In Gil, Y., Motta, E., Benjamins, V. R., and Musen, M. A., editors, *International Semantic Web Conference*, volume 3729 of *Lecture Notes in Computer Science*, pages 987–1001. Springer.

[Reichgelt, 1991] Reichgelt, H. (1991). *Knowledge Representation: An Ai Perspective (Tutorial Monographs in Cognitive Science)*. Ablex Pub.

[Roman and Scicluna, 2006] Roman, D. and Scicluna, J. (2006). Ontology-based choreography of WSMO services. WSMO Working Draft.

[Singh and Huhns, 2005] Singh, M. P. and Huhns, M. N. (2005). *Service-Oriented Computing : Semantics, Processes, Agents*. John Wiley & Sons.

[Sowa, 1983] Sowa, J. F. (1983). *Conceptual Structures: Information Processing in Mind and Machine (Systems Programming Series)*. Addison-Wesley.

[Stojanovic and Dahanayake, 2005] Stojanovic, Z. and Dahanayake, A. (2005). *Service-Oriented Software System Engineering Challenges and Practices*. Idea Group Publishing.

[Strassner et al., 2003] Strassner, J., Fleck, J., Huang, J., Faurer, C., and Richardson, T. (2003). TMF white paper on NGOSS and MDA.

[Sun Microsystems, 1999] Sun Microsystems (1999). Simplified guide to the java 2 platform, enterprise edition.

[Thiadens, 2005] Thiadens, T. (2005). *Manage IT! : Organizing IT Demand and IT Supply*. Springer.

[Trienekens et al., 2004] Trienekens, J. J. M., Bouman, J. J., and Zwan, M. V. D. (2004). Specification of service level agreements: Problems, principles and practices. *Software Quality Control*, 12(1):43–57.

[Uschold and Grüninger, 1996] Uschold, M. and Grüninger, M. (1996). Ontologies: principles, methods, and applications. *Knowledge Engineering Review*, 11(2):93–155.

[van der Aalst, 2002] van der Aalst, W. M. P. (2002). Making work flow: On the application of petri nets to business process management. In Esparza, J. and Lakos, C., editors, *ICATPN*, volume 2360 of *Lecture Notes in Computer Science*, pages 1–22. Springer.

[Voss et al., 1985] Voss, C., Armistead, C., Johnston, B., and Morris, B. (1985). *Operations Management in Service Industries and the Public Sector: Text and Cases*. John Wiley & Sons.

[Weerawarana et al., 2005] Weerawarana, S., Curbera, F., Leymann, F., Storey, T., and Ferguson, D. F. (2005). *Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall PTR.

[Wooldridge, 2002] Wooldridge, M. (2002). *Introduction to MultiAgent Systems*. John Wiley & Sons.

[Yang et al., 2003] Yang, G., Kifer, M., and Zhao, C. (2003). Flora-2: A rule-based knowledge representation and inference infrastructure for the semantic web. In Meersman, R., Tari, Z., and Schmidt, D. C., editors, *CoopIS/DOA/ODBASE*, volume 2888 of *Lecture Notes in Computer Science*, pages 671–688. Springer.

List of Abbreviations

API: Application Programming Interface

B2B: Business to Business

CBD: Component Based Development

COM: Component Object Model

CORBA: Common Object Request Broker Architecture

DSL: Digital Subscriber Line

EJB: Enterprise Java Beans

epBOM: eTOM public B2B Business Operations Map

eTOM: Enhanced Telecom Operations Map

HTTP: Hyper Text Transfer Protocol

IETF: Internet Engineering Task Force

ISP: Internet Service Provider

IT: Information Technology

ITU/ETSI: International Telecommunication Union / European Telecommunications Standard Institute

MDA: Model driven architecture

MOF: Meta Object Facility

OO: Object Oriented

OWL: Web Ontology Language

OWL-S: Web Ontology Language for Services

RDF: Resource Description Framework

RSA: Cryptographic algorithm named after its inventors: Rivest, Shamir, Adleman.

SADT: Structured Analysis and Design Technique

SID: Shared Information/Data Model

SLA: Service Level Agreement

SoA: Service-oriented Architecture

SOAP: Previously: Simple Object Access Protocol. This acronym was deemed misleading and therefore currently SOAP doesn't have an acronym expansion anymore.

SWRL: Semantic Web Rule Language

SWSF: Semantic Web Service Framework

SWSL: Semantic Web Service Language

SWSO: Semantic Web Service Ontology

UDDI: Universal Description, Discovery and Integration

URI: Uniform Resource Identifier

UML: Unified Modeling Language

WS-BPEL: Business Process Execution Language for Web Services

WSDL: Web Service Description Language

WSLA: Web Service Level Agreement

WSMF: Web Service Modeling Framework

WSMO: Web Service Modeling Ontology

WSML: Web Service Modeling Language

WSMX: Web Service Modeling Execution Environment

XML: Extensible Markup Language

XML-Schema: Meta-modeling language to define a concrete XML based language

OWL Description Logic Discovery

```
<?xml version="1.0"?>

<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:owl="http://www.w3.org/2002/07/owl#"
 xmlns="http://www.owl-ontologies.com/telecom.owl#"
 xml:base="http://www.owl-ontologies.com/telecom.owl">
 <owl:Ontology rdf:about=""/>
 <owl:Class rdf:ID="connectivity_offer">
  <owl:equivalentClass>
   <owl:Class>
    <owl:intersectionOf rdf:parseType="Collection">
     <owl:Restriction>
      <owl:hasValue
       rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
       1000
      </owl:hasValue>
      <owl:onProperty>
       <owl:FunctionalProperty rdf:ID="speed"/>
      </owl:onProperty>
     </owl:Restriction>
     <owl:Restriction>
      <owl:someValuesFrom>
       <owl:Class rdf:ID="DSL"/>
      </owl:someValuesFrom>
      <owl:onProperty>
       <owl:ObjectProperty rdf:ID="technology"/>
      </owl:onProperty>
     </owl:Restriction>
     <owl:Restriction>
      <owl:onProperty>
       <owl:DatatypeProperty rdf:ID="delay"/>
      </owl:onProperty>
      <owl:hasValue
       rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
       20
      </owl:hasValue>
     </owl:Restriction>
```

```
    </owl:intersectionOf>
   </owl:Class>
  </owl:equivalentClass>
 </owl:Class>
 <owl:Class rdf:ID="match_cr_dsl_co">
  <owl:equivalentClass>
   <owl:Class>
    <owl:intersectionOf rdf:parseType="Collection">
     <owl:Class rdf:about="#connectivity_offer"/>
     <owl:Class>
      <owl:complementOf>
       <owl:Class rdf:ID="connectivity_request"/>
      </owl:complementOf>
     </owl:Class>
    </owl:intersectionOf>
   </owl:Class>
  </owl:equivalentClass>
  <rdfs:subClassOf
   rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
 </owl:Class>
 <owl:Class rdf:ID="WiMax"/>
 <owl:Class rdf:ID="connectivity_request_dsl">
  <rdfs:subClassOf>
   <owl:Restriction>
    <owl:someValuesFrom rdf:resource="#DSL"/>
    <owl:onProperty>
     <owl:ObjectProperty rdf:about="#technology"/>
    </owl:onProperty>
   </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf
   rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
 </owl:Class>
 <owl:Class rdf:about="#connectivity_request">
  <owl:equivalentClass>
   <owl:Restriction>
    <owl:hasValue
     rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
     1000
    </owl:hasValue>
    <owl:onProperty>
     <owl:FunctionalProperty rdf:about="#speed"/>
    </owl:onProperty>
   </owl:Restriction>
  </owl:equivalentClass>
 </owl:Class>
 <owl:Class rdf:ID="match_cr_co">
  <owl:equivalentClass>
   <owl:Class>
    <owl:intersectionOf rdf:parseType="Collection">
     <owl:Class rdf:about="#connectivity_offer"/>
     <owl:Class>
      <owl:complementOf rdf:resource="#connectivity_request"/>
     </owl:Class>
```

```
    </owl:intersectionOf>
   </owl:Class>
  </owl:equivalentClass>
 </owl:Class>
 <owl:ObjectProperty rdf:about="#technology">
  <rdfs:domain>
   <owl:Class>
    <owl:unionOf rdf:parseType="Collection">
     <owl:Class rdf:about="#connectivity_request_dsl"/>
     <owl:Class rdf:about="#connectivity_offer"/>
    </owl:unionOf>
   </owl:Class>
  </rdfs:domain>
  <rdf:type
   rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"
  />
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
   This property points at a concrete technology that is
   used for realizing the connection.
  </rdfs:comment>
 </owl:ObjectProperty>
 <owl:DatatypeProperty rdf:about="#delay">
 <rdf:type
  rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
   Average packet delay in miliseconds between the customer
   and the ISP network.
  </rdfs:comment>
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
 </owl:DatatypeProperty>
 <owl:FunctionalProperty rdf:about="#speed">
 <rdfs:range
  rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
   The throughput of the connection in kilobits per second
   (kbps).
  </rdfs:comment>
  <rdf:type
   rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain>
   <owl:Class>
    <owl:unionOf rdf:parseType="Collection">
     <owl:Class rdf:about="#connectivity_offer"/>
     <owl:Class rdf:about="#connectivity_request"/>
    </owl:unionOf>
   </owl:Class>
  </rdfs:domain>
 </owl:FunctionalProperty>
</rdf:RDF><!— Created with Protege (with OWL Plugin 2.1, Build 284)
http://protege.stanford.edu —>
```

List of Figures

List of Tables