Institut für Informatik der Technischen Universität München

Search Result Management System (SerumS) - An Approach for Efficient and Consistent Web Services Brokering

Quang Cua Cao

Vollständiger Abdruck der von der Fakultät der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Univ.-Prof. Dr. Johann Schlichter
Prüfer der Dissertation:	1.	Univ.-Prof. Dr. Eike Jessen, em.
	2.	Univ.-Prof. Dr. Martin Bichler

Die Dissertation wurde am 14.12.2006 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 02.02.2007 angenommen.

Abstract

The key concept of Web services is to make both inter- and intra-application integration possible in a way that the integration does not depend on the platform or implementation of the service customer or provider. Since the information about Web services and their providers is usually managed in a central UDDI[1] registry and the present UDDI standard does not offer a mechanism for the automatic actualization of the Web service information, it is often inaccurate or outdated[1]. This work presents a concept for solving these problems. Moreover, it allows to optimize the Web service application process by adopting a Pull/Push technique and Publish/Subscribe Model based search and management system for Web services. Furthermore, the system searches proactively for new Web services and selects the appropriate service for the service customer. This improves significantly the reaction time and the performance of the Web service application process.

The following main contributions build up the core of the work:

- The current UDDI standard is extended by a Publish/Subscribe mechanism
- Service customers can use the Publish/Subscribe mechanism to register themselves for certain Web services
- Reduction of response time for service customers by automatic searching for new Web services by the service broker (in our case SerumS)
- Improved correctness and completeness of the search result
- Notification about new Web services and changes to the service customers through the service provider
- Avoidance of inconsistent data by automatic update of the Web service information

The aim of this work is to use the "Publish-Subscribe Notification for Web services"[2] specification for realizing the Publish/Subscribe mechanism, so that the result can be considered as a replacement for a UDDI registry. The proof of concept (PoC), which results from this work, has in addtion to the traditional functionalities of a UDDI registry also the features mentioned above.

[1] Universal Description Discovery and Integration

Zusammenfassung

Das Schlüsselkonzept von Web Services liegt darin, sowohl Unternehmens-übergreifende als auch -interne Anwendungsintegration auf die Art und Weise zu ermöglichen, dass die Integration weder von der Plattform noch von der Implementierung des Service Customer oder Provider abhängt. Da die Informationen zu den Web Services und deren Provider in einer zentralen UDDI Registry verwaltet werden und der aktuelle UDDI Standard keinen Mechanismus zur automatischen Aktualisierung der Web Service Informationen besitzt, sind diese sehr oft ungenau und "out-of-date"[1]. In dieser Arbeit wird ein Konzept zur Lösung dieser Probleme vorgestellt, welches darüber hinaus den Web Service Anwendungsprozess durch den Einsatz eines auf Pull/Push-Technik und Publish/Subsribe-Modell basierten Such- und Managementsystems optimiert. Außerdem sucht das System proaktiv nach neuen Web Services und wählt für den Service Customer den passenden Service. Dies verbessert significant die Reaktionszeit und die Performanz des Web Service Anwendungsprozesses.

Die folgenden Hauptbeiträge bilden den Kern der Arbeit:

- Der aktuelle UDDI Standard wird um den Publish/Subscribe Mechanismus erweitert
- Service Customer können den Publish/Subscribe Mechanismus benutzen, um sich für bestimmte Services zu registrieren.
- Reduktion der Antwortzeit für Service Customer durch automatisches Suchen nach neuen Web Services durch den Service Broker (in unserem Fall SerumS)
- Verbesserte Korrektheit und Vollständigkeit des Suchergebnisses.
- Die Service Customer werden von den Service Provider über neue Web Services und ihre Änderungen benachrichtigt.
- Vermeidung von inkonsistenten Daten durch automatisches Update der Web Service Informationen.

Das Ziel dieser Arbeit ist das Benutzen der "Publish-Subscribe Notification for Web services"[2] Spezifikation für die Realisierung des Publish/Subscribe Mechanismus, so dass das Ergebnis als Ersatz für die UDDI Registry betrachtet werden kann. Das "Proof of Concept (PoC)" besitzt zusätzlich zu den traditionellen Fuktionalitäten einer UDDI Registry auch die gerade erwähnten Neuerungen.

Acknowledgement

First of all, I would like to thank Univ-Prof. Dr. Eike Jessen and Univ.-Prof. Dr. Martin Bichler for supervising this thesis and giving me the full freedom to work on it. I especially appreciate their many suggestions and help during my research. I like to thank the colleagues at "The chair of Internet-based Information Systems (IBIS)", particularly Adrian Paschke, for many precious and fruitful discussions.

I would also like to thank Dr. Manfred Jobmann for his discussions, and in particular for helpful questions and suggestions which have improved the quality of this work.

Finally, I like to thank again my first supervisor Univ-Prof. Dr. Eike Jessen, Mcs. Mohammad Khaleghi of the "Chair for Theoretical Computer Science and Foundations of Artificial Intelligence", Bilen Emek Abali and the colleagues of the chair "Network Architectures" for the proof-reading of this thesis.

Contents

List of Figures

Listings

List of Tables

Chapter 1

Introduction

1.1 Distributed system architecture and application resource management

In distributed systems application components are developed by their service provider and finally deposed at an appropriate location, where they can be accessed by a service client[1]. It is necessary for the service provider to inform service clients, where and how (e.g. by which technical media) they can find the application interfaces. Generally speaking, it concerns a problem of the resource localization or formally a problem of asymmetrical distribution of application resources, from which undesired efficiency problems for the service user result in using desired services. This straightly represents a main problem in Web services, because an efficient localization and administration of the Web service information is necessary in order to find the application interfaces and establish efficient use of them. Figure 1.1 shows the model of traditional distributed application systems and the problem associated with them.

The objects involved in a distributed application can be divided in two groups:

1. *Service information-owning entities*:
   They are *passive* because they do not perform actions, rather they only react to inquiries performed by other entities like requesting for certain resources from a service client
2. *Service information-using entities*:
   They are *active* because they have to perform appropriate actions to get necessary service resources.

Figure 1.2 shows the relation between the service resources and the activities of the application entities involved in the application process. The relation between the elements of Figure 1.2 can be explained better with the matrix depicted in Table 1.1

[1]In this work we use the term "service client" for a software component, "service user" for a human actor and "service customer" in an e-commerce context. Analogously, we say "server" for a *technical* component, otherwise the term "service provider" is used for an *human* actor or an institution

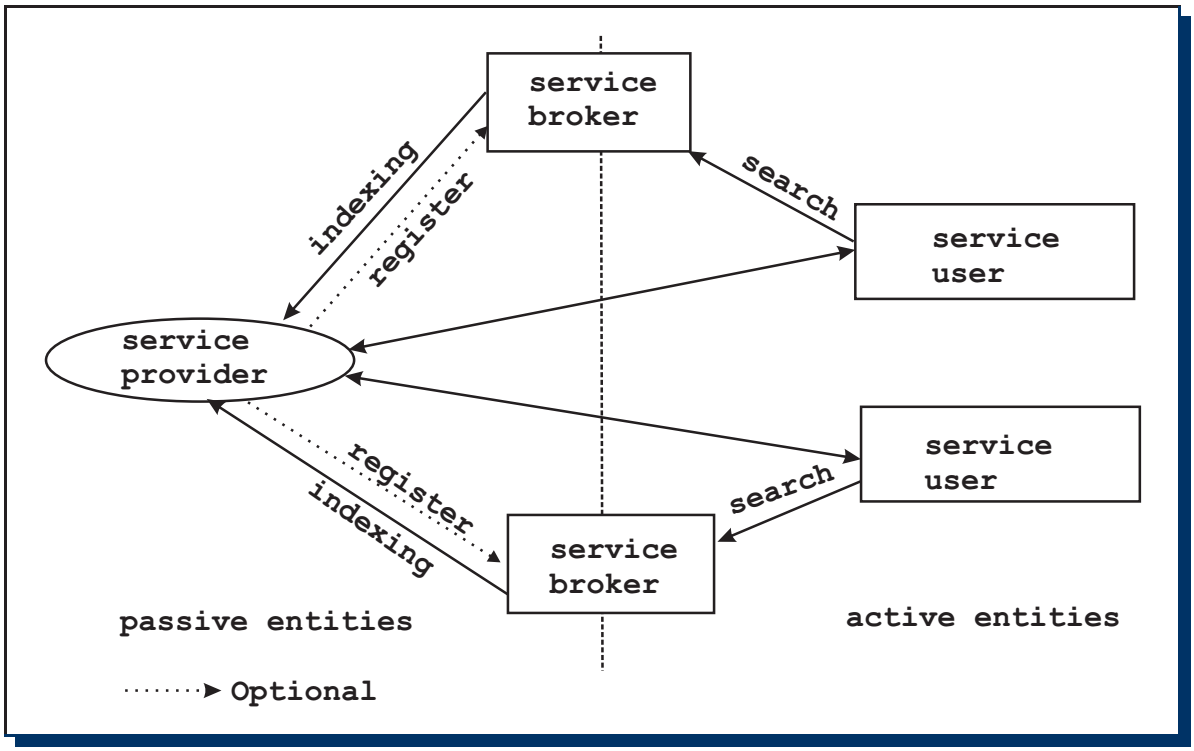

Figure 1.1: Model of a traditional distributed application

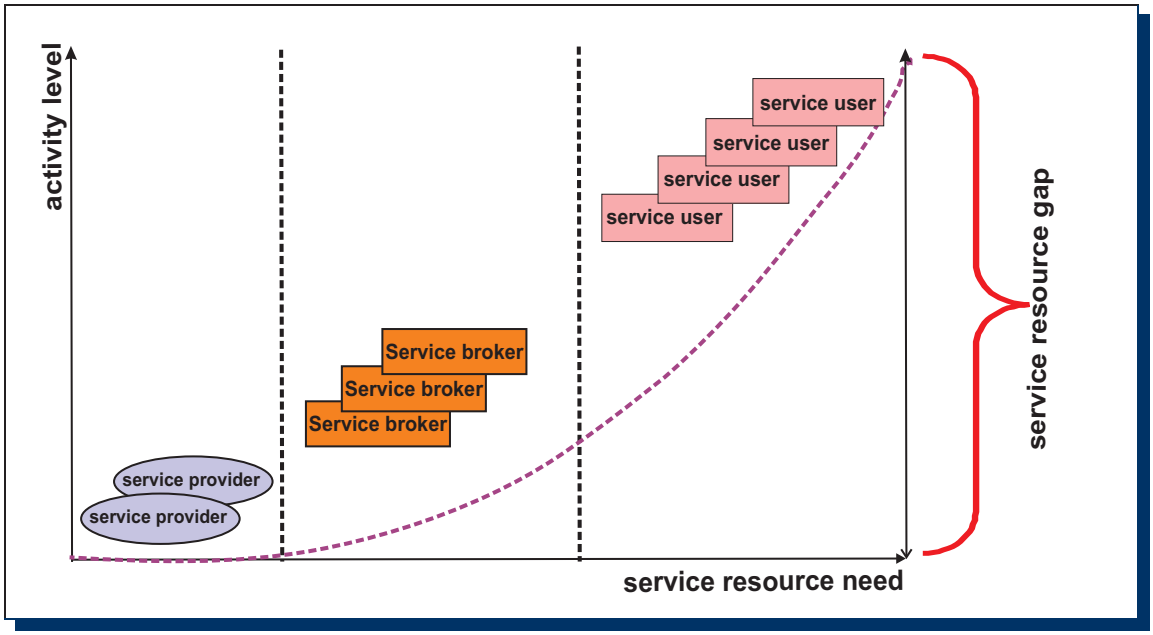

Figure 1.2: Relation between service resources and the activity level of the entities

Service entities	Activity level	Need for service resources	Service resource gap
Providers	very low	none	none
Brokers	low	low	low
Users	high	high	high

Table 1.1: Service resource and activity level matrix

The matrix shows that the need for service resources is very high for the service users. The reason is the unavailability of prior knowledge about the existence of the service providers and their services. The high demand for the service resources requires some actions from a service user to find them and hence leads to a high activity level. Activity level is the work which a service user spends for service search. Generally speaking, we are dealing with the service resource gap issue, which represents the set of the missing service information needed by a service user for using a certain service. This service resource gap results from the natural distribution of the service resources and the need of service resources at the side of the service user. Actually, service resource gap is the basic problem from which other issues regarding the performance and efficiency of the execution of the service process result. In the next section we examine more concretely how it comes to the service resource gap issue. Also, we point out that service resource gap is a general problem of the current distributed application architecture and it needs to be solved.

1.2 Push-Pull technique vs. Pull technique

1.2.1 Pull technique

Most applications (whether they are standalone or distributed) are based on an "one way" architecture, where the client has to perform all the necessary work steps in order to achieve a certain result. In local or standalone applications it is not necessary for the client to retrieve the appropriate information to use the application functionality. More complicated are distributed application systems consisting of a number of different clients and server software components, which are at different places and are connected through a communication medium, so that remote operation is possible. The communication medium can be a network section within a Local Area Network (LAN) or a Wide Area Network (WAN). Figure 1.3 shows the basic cycle of a distributed application and the elementary communication processes via an activity diagram. In this scenario we assume that the client does not have any information about the location of the server and its services in advance.

Figure 1.4 shows more details about the service process and points out the relationship between the involved objects via a sequence diagram. The diagram illustrates the basic process model of the most distributed application systems today. The whole process consists of four elementary steps:

1. *Searching for necessary service information:*
   A client sends a search request to a service broker for information, which is necessary to

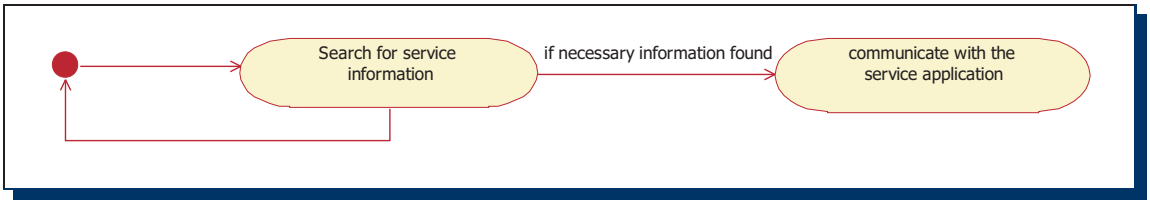

Figure 1.3: Search cycle of the distributed application process

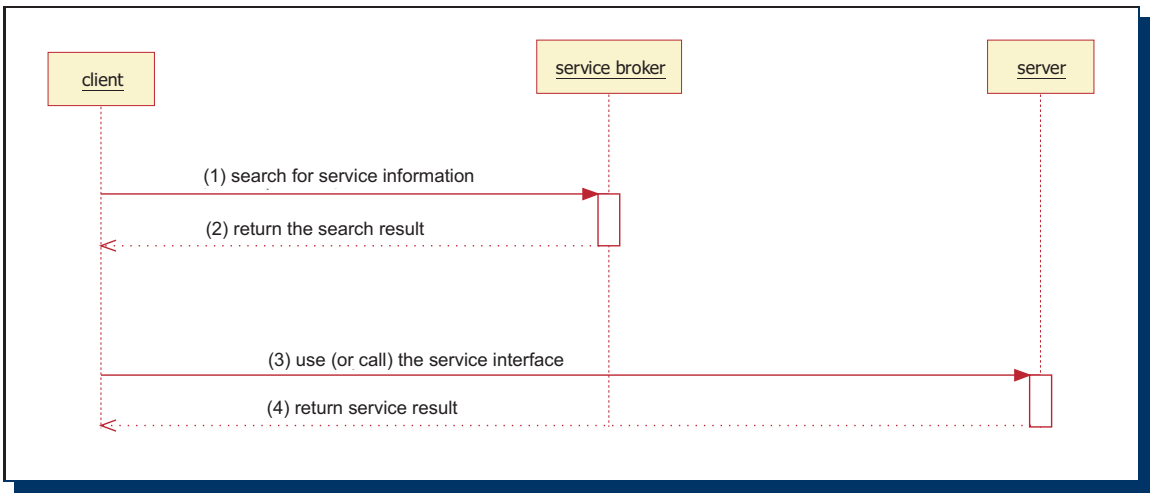

Figure 1.4: Process and involved objects of a distributed application

find and to use the service

2. *Obtaining the service information from a service broker:*
   As result of the search process, the client can obtain some service information, from which it chooses the appropriate one to use.

3. *Calling the service interface:*
   The client calls the service (interface) at the address of the server.

4. *Obtaining the service result from the server:*
   The server can return some results to the client if they exist and/or if the client explicitly requested to return a result.

A typical case of such processes takes place when someone is using a search engine to find certain service or general information. An example with "www.google.com" as service broker is demonstrated in Figure 1.5.

As one can take from the three diagrams (Figure 1.3, 1.4 and 1.5), the client has to be active from the beginning of each application process. It first has to find out whether there is any service at all to satisfy its need. Then it has to explore from where and how it can access the service. Thus, by this architecture the client has to do the major part of the whole application process; the key word in this context is *"search for service information"*. Instead of *"search for*

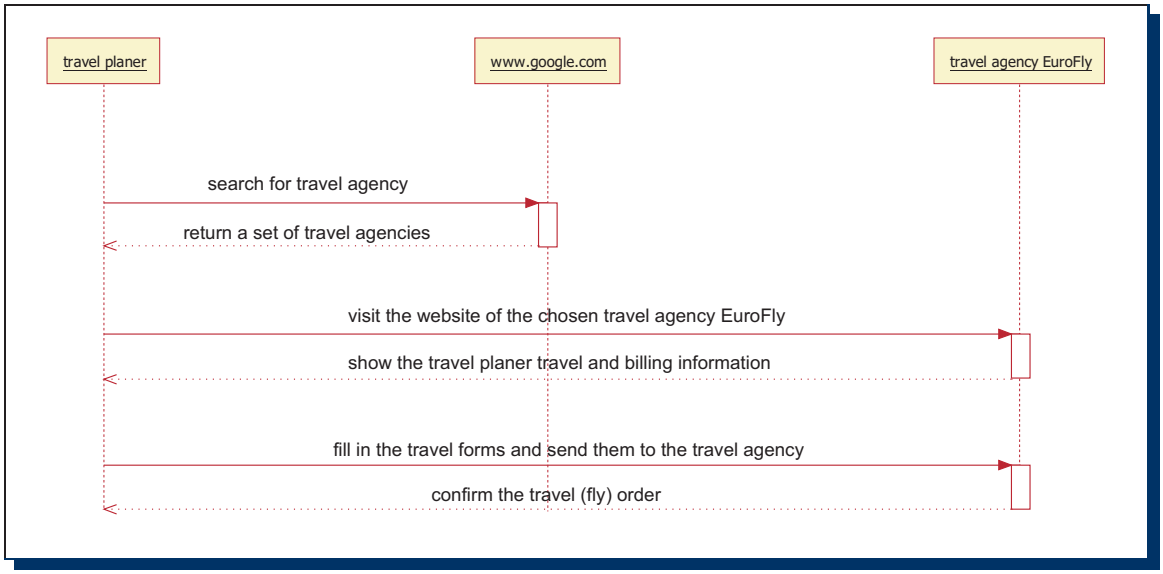

Figure 1.5: "www.google.com" as service broker for travel planning

service information" we also can say: *"pull the service information"*. It means that the active part of the distributed application system has to pull the necessary information to itself, e.g. request it in order to call or use the application functionality of the remote part(s). Thus, "Pull technique" based applications suffer from the following disadvantages:

- The effort and load within the application process is not shared symmetrically among the involved entities.
- The client part has the highest activity level
- The application information needed by the client has to be searched "ad hoc" when the application has to be performed and thereby extends the service execution time.
- the service information is not up-to-date, if it is cached by the client and changed meanwhile on the server.

Hence, it is necessary to consider alternative ways which can avoid these disadvantages. This leads to the "Pull-Push" technique, which will be discussed in detail in the next section.

1.2.2 Pull-Push technique

Besides the predominant number of distributed applications based on the Pull technique, there are also many existing applications based on the Pull-Push technique. The most well-known application system based on this technique is a "news group", to which a user can register himself to get *automatically* news articles from one or many news servers. Another example is an email application system. An email application system works like a news group with the

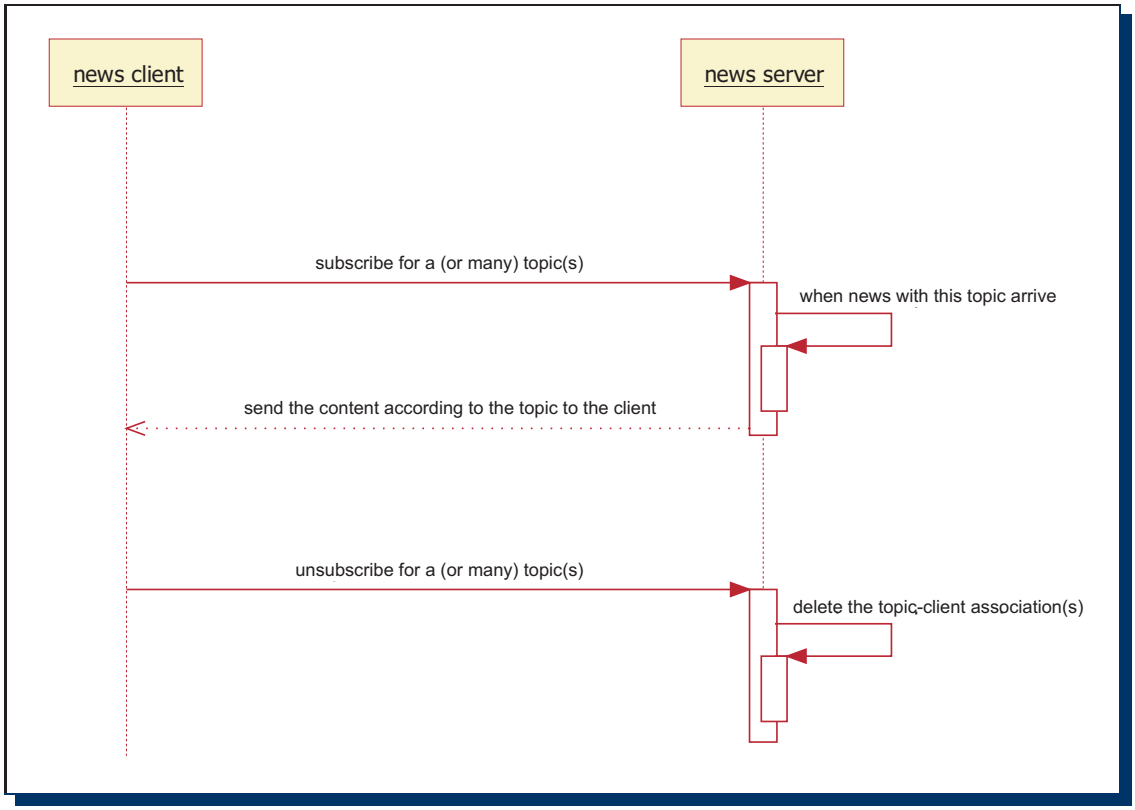

Figure 1.6: Process model of news and email application systems

difference that the receiver of an email is not the whole group but is always explicitly determined by the sender. However, both news group and email system work as in Figure 1.6.

The basic process model of applications based on the Pull-Push technique is shown in Figure 1.7. The main difference between a Pull and a Push-Pull based application is the possibility for a client to register itself at the service broker for certain services. Due to this, a service broker or in some circumstances the server itself can inform the interested clients (subscribers) about new relevant events.

Thus, the Pull-Push technique offers the following advantages:

- The search for services by the client can be omitted
- The client is informed in time about new (interested) services
- the client gets only the service information according to its registered (subscribed) topic(s) and hence the data redundancy can be limited.

But the most important advantage of the Push-Pull technique is the possibility of an application broker or server to *inform* (or *notify*) a client about changes on existing services so that the client can access the service interfaces without any failure or raising any exception. This is one of the main aspects which belongs to the core of this thesis and will be discussed concretely and with

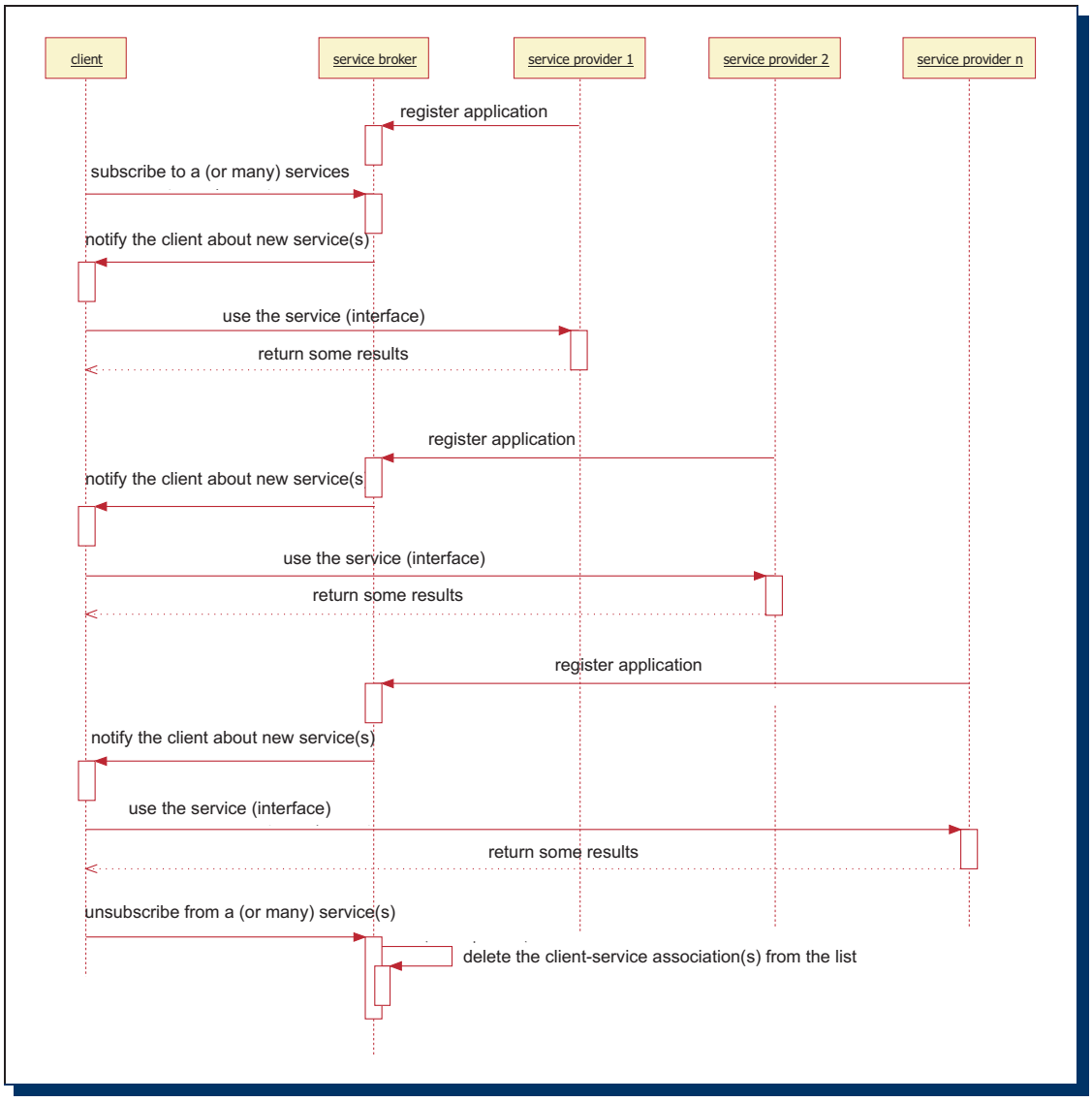

Figure 1.7: Basic process model of Push-Pull technique based applications

more details in the following chapters. Figure 1.8 summarizes the core differences between the Pull and Push-Pull technique.

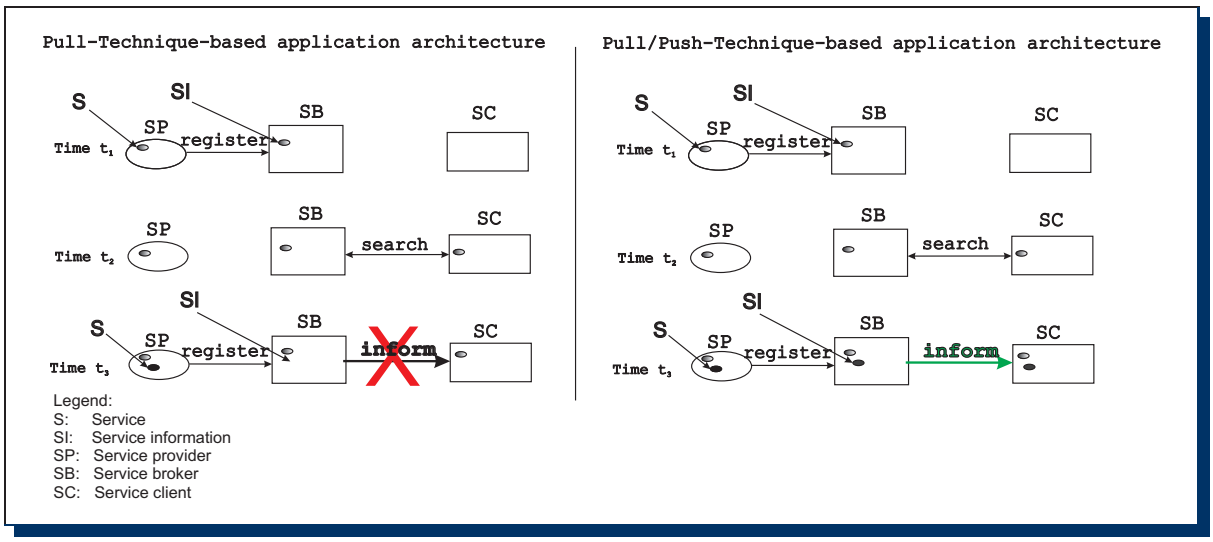

Figure 1.8: Push-Pull technique vs. Pull technique

1.3 Motivation

> "Companies are looking for enhanced capabilities that will help them not only more easily create and use web services, but also help them manage and secure web services within a SOA..." [4]

In the previous section we have discussed general problems within a distributed application environment, which arise as a result of the asymmetric distribution of the application resources. The client suffers as an "active entity" in order to achieve the information before it can use the service. The process model of the distributed application architecture and its problem can be translated to the Service Oriented Architecture (SOA), on which Web services are based. We assume that the reader is already familiar with the SOA concept, but for the sake of completeness we will give a short overview of it. Although there is not any generally acceptable definition of SOA today[5], in the article [6] one finds a detailed discussion about SOA along with the following definition:

> [SOA is] "an application architecture within which all functions are defined as independent services with well-defined invokable interfaces which can be called in defined sequences to form business processes"

From the above definition, the following key aspects are typical for the SOA concept:

- All application functions are defined as *services* with well defined interfaces to communicate with other components.

- All services are *independent*. They are designed as "black boxes"; one service neither knows nor cares how the other one performs its function. Only the result returned by the service is relevant.

- *Well-defined* invokable interfaces which can be called *within sequences* of a defined business process. This enables execution of complex business processes among different many service providers.

The following section points out the relation between both the distributed application architecture and the SOA. It shows how the mentioned Push-Pull technique can be used to solve the problems in the current Web services infrastructure. Figure 1.9 shows the Web service environment with an UDDI service broker as an example application system based on the SOA [7]:

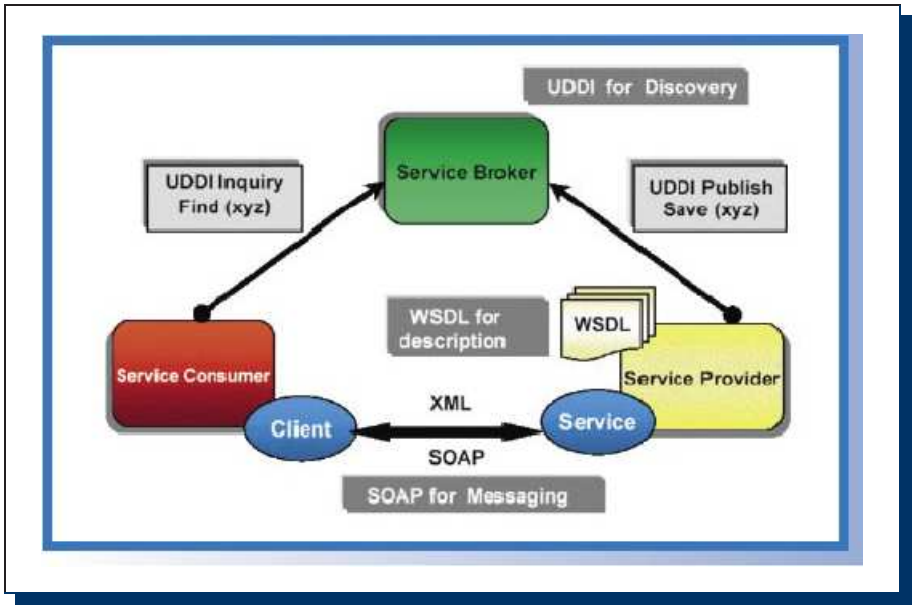

Figure 1.9: Example of an application based on the Service Oriented Architecture (SOA)

Unlike traditional distributed applications, which often use a company's own proprietary protocol to enable the communication between the application components, Web services use standardized communication protocols with appropriate message description languages. But this only contributes to the independence aspect so that both service providers and service clients do not care about the usability of the service interfaces. The targets which have to be achieved by both the distributed application architecture and especially the SOA are:

- Ability to control the service data and their usage in the broker

- Ability of service providers to control their service data

- Ability to hold Web service information always up-to-date

- Avoidance of performance-related problems

These four requirements represent the core thematic of this thesis. The main strategy to fulfill the requirements is to apply the "Publish-Subscribe Notification for Web services" specification[2], which is based on the Push-Pull technique, to construct a system which can replace a traditional UDDI registry. Before doing this, we need some basic understanding of SOA and Web services. Hence the purpose of the next chapter is to introduce into these topics and provide some necessary vocabulary for further chapters.

1.4 Methodology and structure of the thesis

After introducing the general problem of the current distributed application architecture and presenting a possibility to handle the problem by applying the Push-Pull technique instead of the Pull technique, which is dominant in the distributed and service oriented applications today, we will give a comprehensive overview about Web services in chapter 2. In the chapter 3 the main issues in Web services mentioned in 1.3 are concretized. The discussion of these issues shows which problems generally exist but are not considered seriously by the public and outlines the importance of this thesis. The chapter 4 presents the current evolutions around the topic of this thesis and shows to what extent they are contributing to solve the problems mentioned in the previous chapter. The chapter 5 clarifies the idea of the thesis and describes the method of the solution by applying the "Publish-Subscribe Notification for Web services" specification as a Push-Pull mechanism. This chapter is closed with a Proof of Concept (PoC) containing details of the implementation. The chapter 6 describes the validation of the PoC regarding the performance of the elementary operations and is accompanied by the "Quantitative Modelling" and a simulation to estimate the response time and load capacity of SerumS. The last chapter 7 summarizes the core contributions of this thesis and gives a look-out to future works. Figure 1.10 depicts the structure of the thesis.

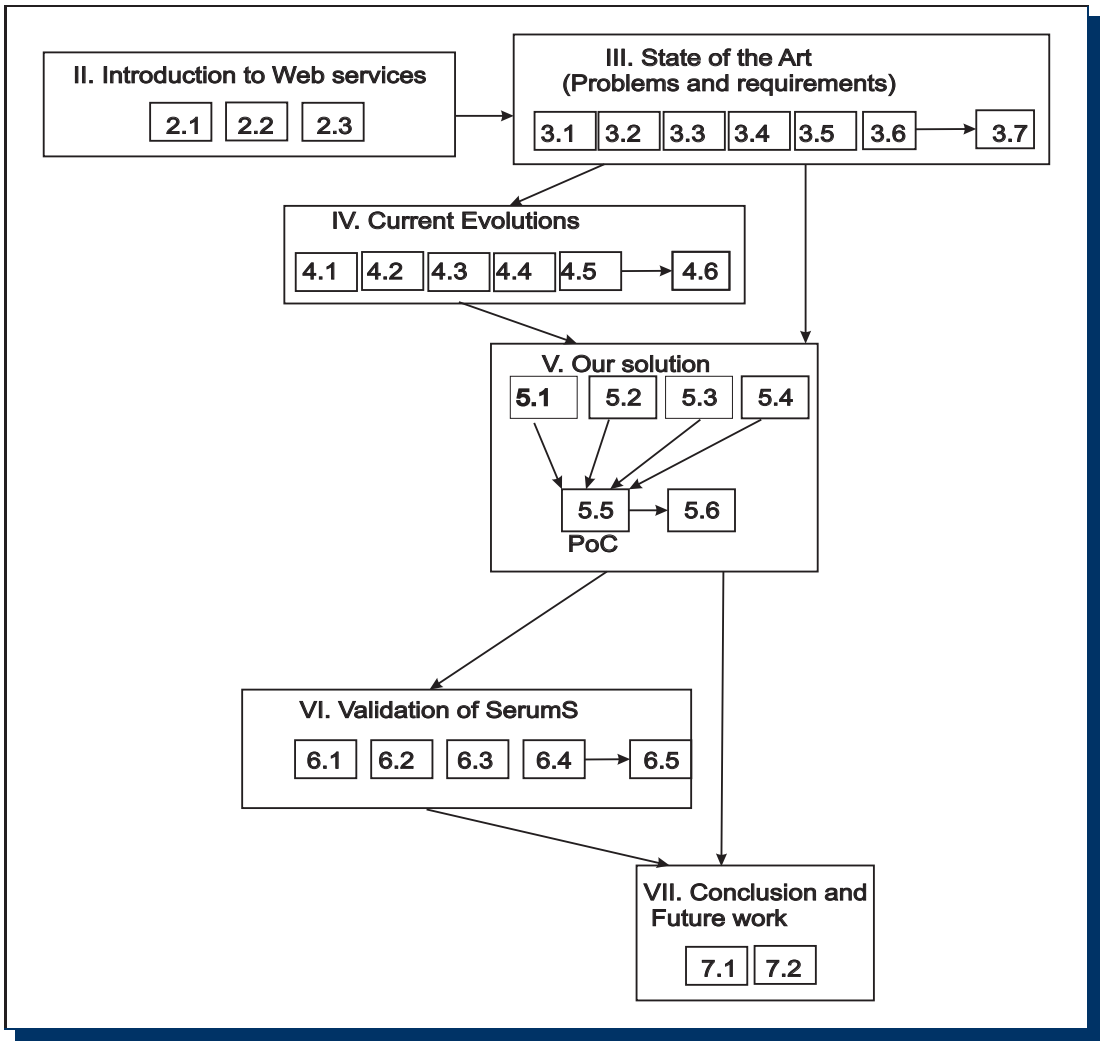

Figure 1.10: Structure of the dissertation

Chapter 2

Introduction to Web services

This chapter introduces the terminology of Web services which is needed as a basis for the further chapters. First, we consider some well-known definitions of Web services and explain our own notion of a Web service. An example Web service used by the company "Amazon"[1] is supposed to illustrate the definitions and terminologies and to show how Web services generally work.

2.1 Definition

The World Wide Web Consortium (W3C)[2] gives the following definition of Web services:

> "A Web service is a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML based messages exchanged via Internet-based protocols"[8]

This definition determines a Web service more technically through standardized message languages and communication protocols. According to the definition above, a Web service (or its service interface) must be described in a well known language which allows automatic interpretation and discovery by a software agent, based on a standardized communication protocol (and not on a company's proprietary one).

Unlike the definition of the W3C consortium IBM gives a more technical definition of Web services:

> "A Web service is an interface that describes a collection of operations that are network- accessible through standardized XML messaging. A Web service is described using a standard, formal XML notion, called its service description. It covers all the details necessary to interact with the service, including message formats (that detail the operations), transport protocols and location. The interface hides

[1] http://www.amazon.com
[2] http://www.w3.org

> the implementation details of the service, allowing it to be used independently of the hardware or software platform on which it is implemented and also independently of the programming language in which it is written. This allows and encourages Web Services-based applications to be loosely coupled, component-oriented, cross-technology implementations. Web Services fulfill a specific task or a set of tasks. They can be used alone or with other Web Services to carry out a complex aggregation or a business transaction."[9]

The first sentence of the IBM's definition is not in accordance with the official W3C's one and is strictly speaking too narrow and not correct. However, the second part describes objectively the technical details of Web services which are relevant in the context of developing Web service applications. The most important outline in this definition is the hiding of the implementation details of a service and the intention to make service developers free from binding to a programming language or platform. Hence the service applications can be created more independently by the developers, and everyone can choose the programming language or tool he is familiar with. Additionally, one does not need to care about the format of the messages which are exchanged between the service application components. The other important property of Web services in this definition is the possibility to combine multiple Web services by different service providers, located at more than one server, to a single one. As a consequence, a whole business transaction, which comprises a set of Web services, can be performed. With BPEL4WS (Business Process Execution language for Web Services)[10] it is possible to determine explicitly the execution of a combined business transaction.

Now we construct our own definition by combining the important properties from the two previous definitions:

> *A Web service is a software component, which is identified and accessed by a unique URI. The URI in general refers to the address of the service provider which describes the service by using a standardized XML based specification language, e.g. Web services Description Language (WSDL)[11] and can be identified and used by a service customer's software agent. One important property which characterizes Web services and makes them different from other existing services is the integration of intra- and extra-company applications from any platform.*

2.2 How are Web services implemented?

In this chapter we discuss the technical aspects of Web services and explain how they are implemented. Altogether, this chapter is supposed to answer the following questions:

a) Which communication protocol is used between the Web service entities (service broker, service client, server)?

b) What is the syntax of messages, which are exchanged as communication protocol elements between the Web service entities?

c) How do service client and server communicate with the service broker (aka. UDDI registry)?

d) How are service interfaces described (in terms of description language and description structure)?

To answer these questions, we take as a Web service example scenario the real service provider "www.amazon.de" (briefly called "Amazon"). Indeed, apart from the services which are available via the Internet using the traditional Internet protocol "http" (known as http://www.amazon.com), Amazon also offers the same services as Web services known as "Amazon E-Commerce Service"[12] described by an appropriate WSDL document. In this example we assume that the service provider Amazon has published its "Amazon E-Commerce Service" by the SAP UDDI node[13] and a service customer "BookCustomer" uses this service to order some book objects. The whole work process from the deployment of the "Amazon E-Commerce Service" until calling the service interfaces by the "BookCustomer" is illustrated in Figure 2.1.

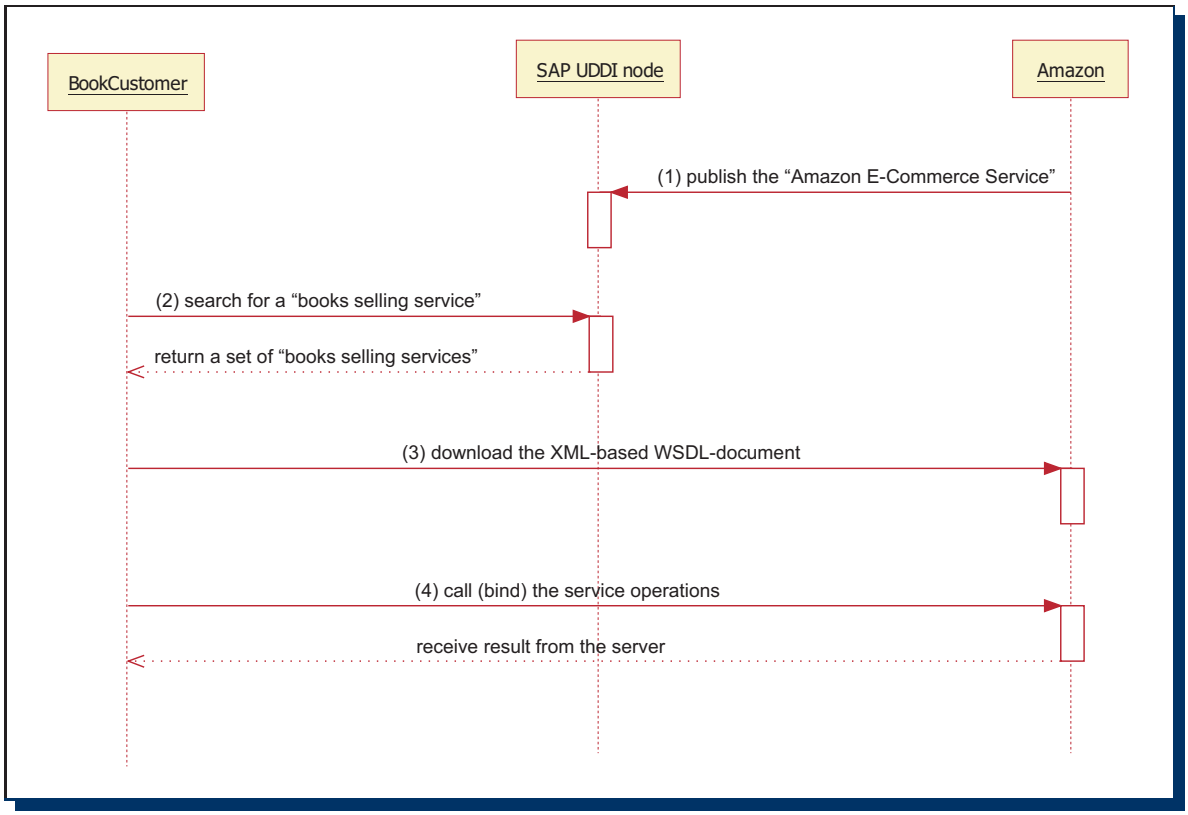

Figure 2.1: Process of the Amazon E-Commerce Service

(1)*Publish the "Amazon E-Commerce Service":*

To make the Web service "Amazon E-Commerce Service" available for the customers, the service provider Amazon publishes it to the SAP UDDI node using the "publish"-interface which is a part of the UDDI-API. The whole "publish" interface consists of 4 atomic operations described as followed:

i. An Amazon operator sends the SOAP-based "save_business" messages (see Listing 2.1) to the UDDI registry to register a "BusinessEntity". We additionally assume that the Amazon operator has already received the necessary authorization information which is

required for all operations to the UDDI registry and it is declared in the "<authInfo>" tag.

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/
  soap/envelope/">
  <soapenv:Body>
    <save_business generic="2.0" xmlns="urn:uddi-org:api_v2">
      <authInfo>A7309120-4FAF-11DA-A536-CC20AD5F81BA</authInfo>
      <businessEntity businessKey="">
        <name>www.amazon.com</name>
          <description>
            Amazon E-Commerce Service,
            book selling service
          </description>
          <contacts>
            <contact useType="Customer Support">
              <personName>Amazone</personName>
              <phone>0123456789</phone>
              <email>webservices@amazon.com</email>
            </contact>
          </contacts>
        </businessEntity>
      </save_business>
    </soapenv:Body>
</soapenv:Envelope>
```

Listing 2.1: The UDDI "save_business" operation message format

With the "save-business" operation message the service provider Amazon publishes the contact and descriptive information about its business. As a result the Amazon operator will obtain a "BusinessKey" which is used for the further steps.

ii. The next message (see Listing 2.2), which belongs to the "publish" operation and is sent to the UDDI registry, contains a "save_tModel" element. Within a "save_tModel" message a service provider can specify technical information about the type of the service "<categoryBag>" which is identified with an unique key number.

iii. Another important element is the "<overviewURL>" which contains the physical address of the technical description of the service (in this case a WSDL document).

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/
  soap/envelope/">
  <soapenv:Body>
    <save_tModel generic="2.0" xmlns="urn:uddi-org:api_v2">
      <authInfo>A7309120-4FAF-11DA-A536-CC20AD5F81BA</authInfo>

        <tModel tModelKey="">
          <name>Amazon E-Commerce Service</name>
          <description xml:lang="en">
            interface for Web service searching and
            ordering Amazon's book items
          </description

          <overviewDoc>
            <description xml:lang="en">
```

```
17                The service's WSDL document
18              </description>
19              <overviewURL>
20                http://webservices.amazon.com/AWSECommerceService/
21                AWSECommerceService.wsdl
22              </overviewURL>
23            </overviewDoc>
24
25            <categoryBag>
26              <keyedReference tModelKey="UUID:C1ACF26D-9672-
27                4404-9D70-39B756E62AB4"
28                  keyName="uddi-org:types" keyValue="wsdlSpec"/>
29              <keyedReference tModelKey="UUID:DB77450D-9FA8-
30                45D4-A7BC-04411D14E384"
31                  keyName="Internet related services"
32                   keyValue="007406"/>
33            </categoryBag>
34          </tModel>
35
36        </save_tModel>
37      </soapenv:Body>
38    </soapenv:Envelope>
```

Listing 2.2: The UDDI "save_tModel" operation message format

The "keyName" (line 28) and "keyValue" (line 32) within the "<categoryBag>" elements define the type of the service offered by the service provider. This information is necessary for service clients to find the service.

iv. The last step is to perform the "save_service" operation (see Listing 2.3). The service provider uses the information obtained in the last steps to describe one or many services.

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/
3     soap/envelope/">
4     <soapenv:Body>
5         <save_service generic="2.0" xmlns="urn:uddie-org:api_v2">
6           <authInfo>
7             A7309120-4FAF-11DA-A536-CC20AD5F81BA
8           </authInfo>
9
10          <businessService
11            businessKey="A8A11FD0-4FA9-11DA-A7C8-8ED3B4EC3B3D "
12            serviceKey="">
13            <name>Amazon E-Commerce Service</name>
14            <description>
15              some detailed descriptions for the service
16            </description>
17            <bindingTemplates>
18              <bindingTemplate bindingKey="">
19                <accessPoint URLType="http">
20                  http://soap.amazon.com/onca/
21                    soap?Service=AWSECommerceService
22                </accessPoint>
23                <tModelInstanceDetails>
24                  <tModelInstanceInfo tModelKey="uuid:DB77450D-9FA8-
25                    45D4-A7BC-04411D14E384">
26                  <instanceDetails>
```

```
27                    <overviewDoc>
28                      <overviewURL>
29                        http://webservices.amazon.com
30                          /AWSECommerceService/AWSECommerceService.wsdl
31                      </overviewURL>
32                    </overviewDoc>
33                  </instanceDetails>
34                </tModelInstanceInfo>
35              </tModelInstanceDetails>
36            </bindingTemplate>
37          </bindingTemplates>
38        </businessService>
39
40      </save_service>
41    </soapenv:Body>
42  </soapenv:Envelope>
```

Listing 2.3: The UDDI "save_service" operation message format

The relation between the "<BusinessEntity>", "<BusinessService>" and "<Service_tModel>" elements, which are also called "UDDI data types", is illustrated in Figure 2.2:

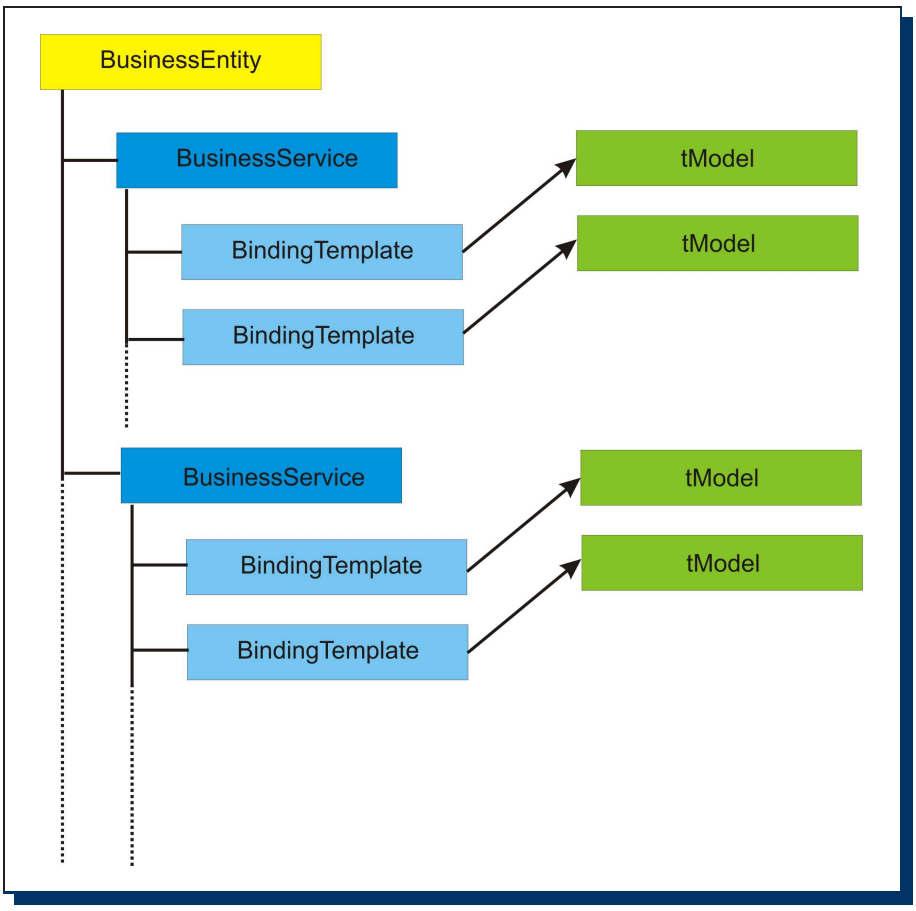

Figure 2.2: Relation between the UDDI data types

- A "<BusinessEntity>" element represents a business object and may own one or

many “<BusinessService>” elements, which represent the concrete service offered by the service provider.

- A “<BusinessService>” element may own “<BindingTemplates>” elements with a “<tModel>” element which specifies the technical details about how a service client can use the service.

(2) *Search for a “book selling service”:*

This step has to be performed by the service client and consists of the following steps:

i. Getting all relevant Web service information about “book selling service” (see appendix A).

The basic idea of the code example is to look for a “books selling service” through the “<categoryBag>” element which specifies the service type. If any services are found, the name and the description (line 39 in appendix A), especially the content of the “<overviewURL>” (line 48 in appendix A), will be printed out for the user to download the WSDL document in order to use the Web service interfaces. Due to the code line 39 in appendix A we obtain the textual information:

```
<name>Amazon E−Commerce Service</name>
<description>some detailed descriptions for the service</description>
```

and at line 48 the content of the “<overviewURL> element and the address, where the WSDL document is located:

```
http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl
```

ii. Obtaining relevant information to use the Web service (interfaces), in our case how to search and order books.

(3) *Download the XML based WSDL document:*

This task has to be performed by the service client to obtain the information about the Web service

(4) *Call the service operations:*

It contains the following tasks:

i. The service client has to determine the “<service>” element within the WSDL document. The Amazon WSDL document “AWSECommerceService.wsdl” contains one such element:

```
<service name=”AWSECommerceService”>
  <port name=”AWSECommerceServicePort”
    binding=”tns:AWSECommerceServiceBinding”>
    <soap:address
```

```
        location="http://soap.amazon.com/onca/
          soap?Service=AWSECommerceService"/>
    </port>
  </service>
```

ii. looking for the "<binding>" element to determine the transport protocol supported by the service provider:

```
  <binding name="AWSECommerceServiceBinding"
    type="tns:AWSECommerceServicePortType">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>
                .
                .
                .
  </binding>
```

Given the "<service>" and "<binding>" elements, a service client knows that the service is accessible at the address "http://soap.amazon.com/onca/soap?Service=AWSECommerceService" and the transport protocol is "soap/http" (aka. "soap over http") which is specified at "http://schemas.xmlsoap.org/soap/http".

Relying on the example of "Amazon" the questions a), b) and c) at the beginning of this chapter (see chapter 2.2) have been answered so far. In the next part we discuss question d) ('*how is the service interface described?*") by looking deeper into a WSDL document. Appendix B shows the structure of a WSDL document and appendix C gives a concrete example. A WSDL-document contains the following information:

- declarations of name spaces used within the WSDL document, e.g. see line 4 in the appendix C
- declarations of the data types used in the document

  Beside the simple data types which are already defined in [14] like String and Integer, a service provider can define other complex data types to be used in the WSDL document, e.g. see line 10 in the appendix C. A complex data type can also point to a data element which is defined in an external document.
- declaration of "<message>" elements

  A "<message>" element corresponds to a "parameter" within a function or method in a programming language and is used in one or many "<portType>" element(s), see line 30 in appendix C.
- declaration of "<portType>" elements as signature of the "<messages>" elements

  A "<portType>" element corresponds to the signature of a function or method in a programming language and can have "<input>" or "<output>" elements if required.
- declarations of "<binding>" and "<service>" elements

All elements described here can be used in "<binding>" elements which in turn are used in the "<service>" elements. The advantage of a "<binding>" element is the possibility for a service client to "bind" dynamically an interface at runtime. For example at line 30 of appendix C, the service provider gives the service customer the possibility to bind the service interface with the transport protocol "soap/http" which is defined at "http://schemas.xmlsoap.org/soap/http". If the service provider asks to change this protocol to a secure one he can replace "http://schemas.xmlsoap.org/soap/http" by "http://schemas.xmlsoap.org/soap/https". The capability of "dynamic binding" is one of the advantages of the WSDL concept compared to other architectures which cannot handle dynamic interface binding without cumbersome changing of the structure of the interface description, e.g. new compiling of skeleton components for a CORBA[3] application.

2.3 Résumé

In this chapter we have discussed the technical aspects of Web services. With the Web service example "Amazon" one can see how complex a whole SOA based application process is. Opposed to other traditional distributed application environments, there is a lot of overhead caused by using XML despite of its own advantages. It is necessary that both client and server should handle the requests and responses as quickly as possible, especially, if many requests are sent at same time to the server. Therefore, it is necessary to make the execution of the Web service application process more efficient by avoiding unnecessary communication processes. In the next chapter we make the context, in which the general consequences of such communication issues become visible, more concrete by presenting the main problems which are solved within this thesis.

[3]Common Object Request Broker Architecture

Chapter 3

State of the Art - General issues

Service providers develop their Web services and publish the associated information in a central repository, the so-called UDDI registry, so that it can be found and used by service clients. The information about a Web service is normally present in form of a XML based WSDL document. The information originating from a WSDL document is laid down in the UDDI registry according to the mapping pattern[15] shown in Figure 3.1.

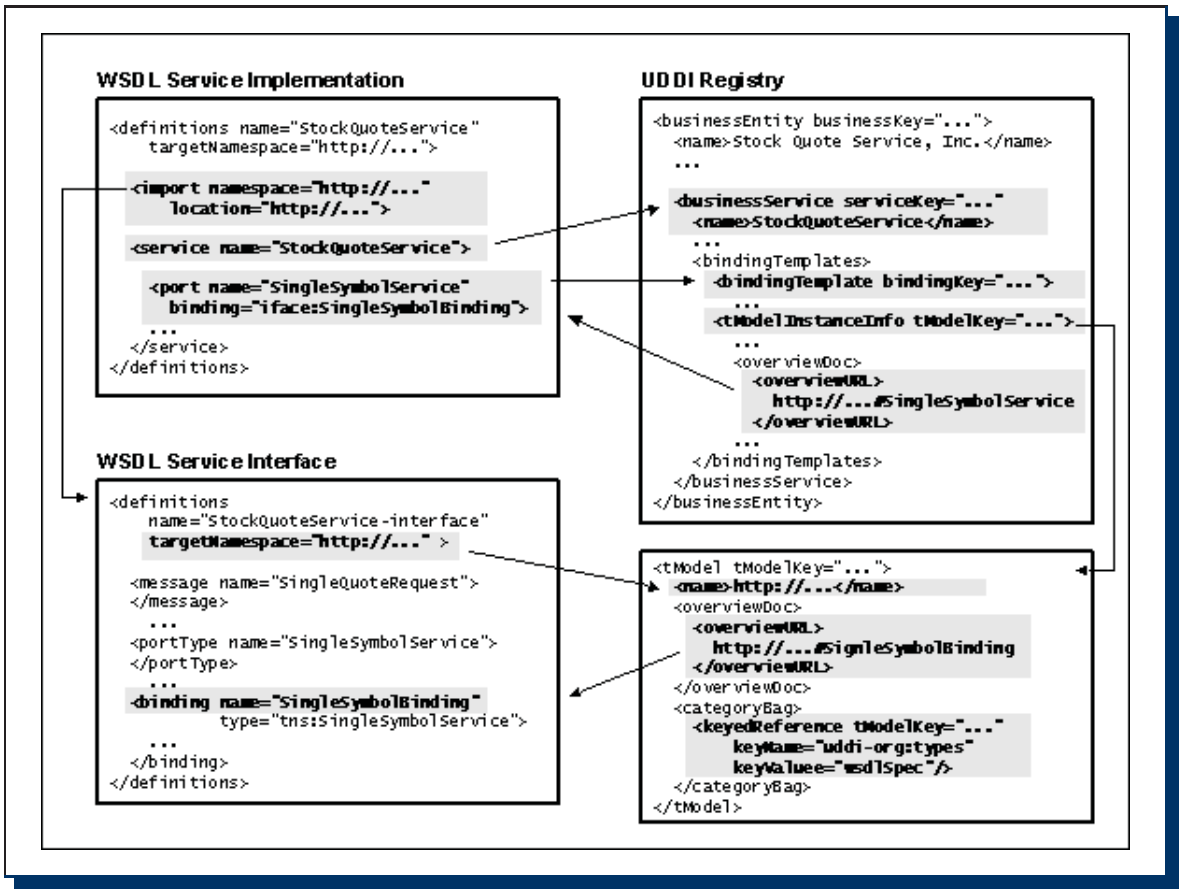

Figure 3.1: WSDL-UDDI Mapping Schema

This approach seems to be systematic and meaningful at first glance, however, there are many aspects which have a negative effect on the execution process of the Web services application and are discussed in the following section.

3.1 Different implementations of UDDI Business Registry node (UBR node)

3.1.1 Non-uniform usability

Today, there are many different UBR nodes provided for real business as well as for test purposes. Some of well known companies which are hosting UBR nodes are Microsoft[1], IBM[2], SAP[3]. Although the UBR implementations are conform with the UDDI specification[16], there is no uniform usability regarding the GUI-based frontend client software. Generally speaking, every UBR node has its own user interface which has to be first studied carefully by the user before he can use the functionality to publish Web services and/or search for them. As examples, Figures 3.2 and 3.3 show the user interface of two different UBR nodes.

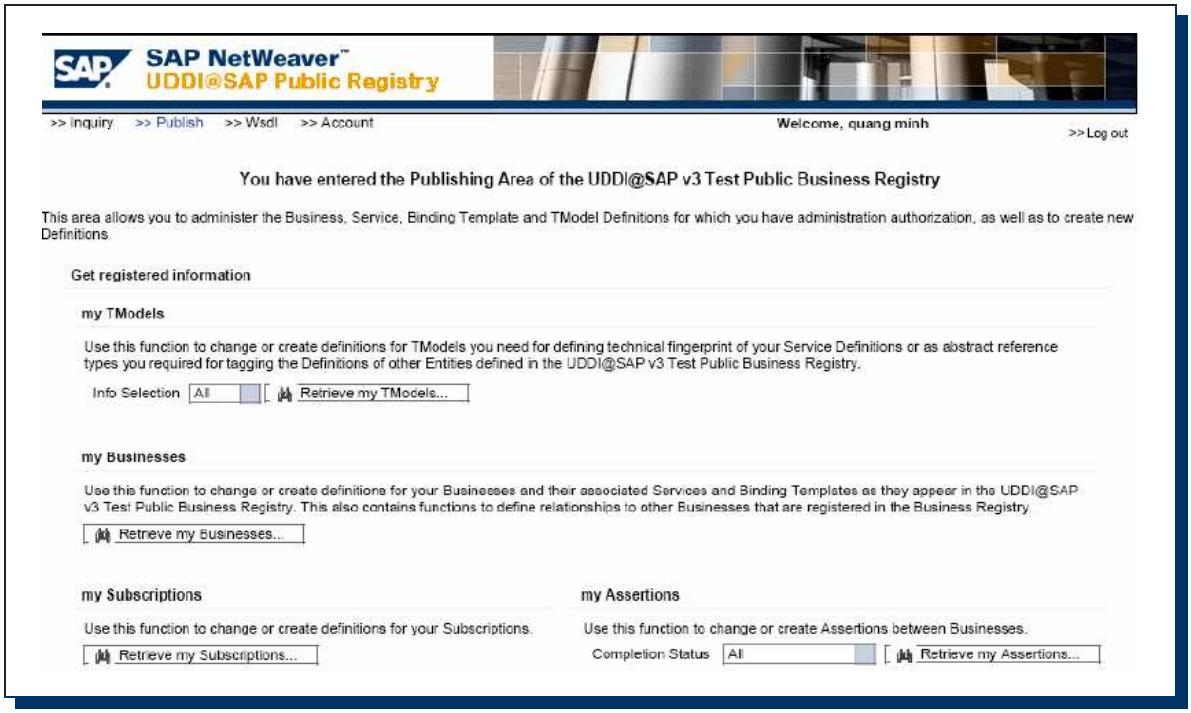

Figure 3.2: The browser-based user interface of the SAP Test Public Business Registry[13]

These two Figures clearly show how different the GUIs and the terminologies used by each UBR node operator are. While in the first case a complex user interface with a separated representation of the “inquiry()” and “publish()” functionality is provided, the user interface of XMethods[4] is

[1] http://uddi.microsoft.com
[2] http://uddi.ibm.com
[3] http://uddi.sap.com
[4] http://www.xmethods.net

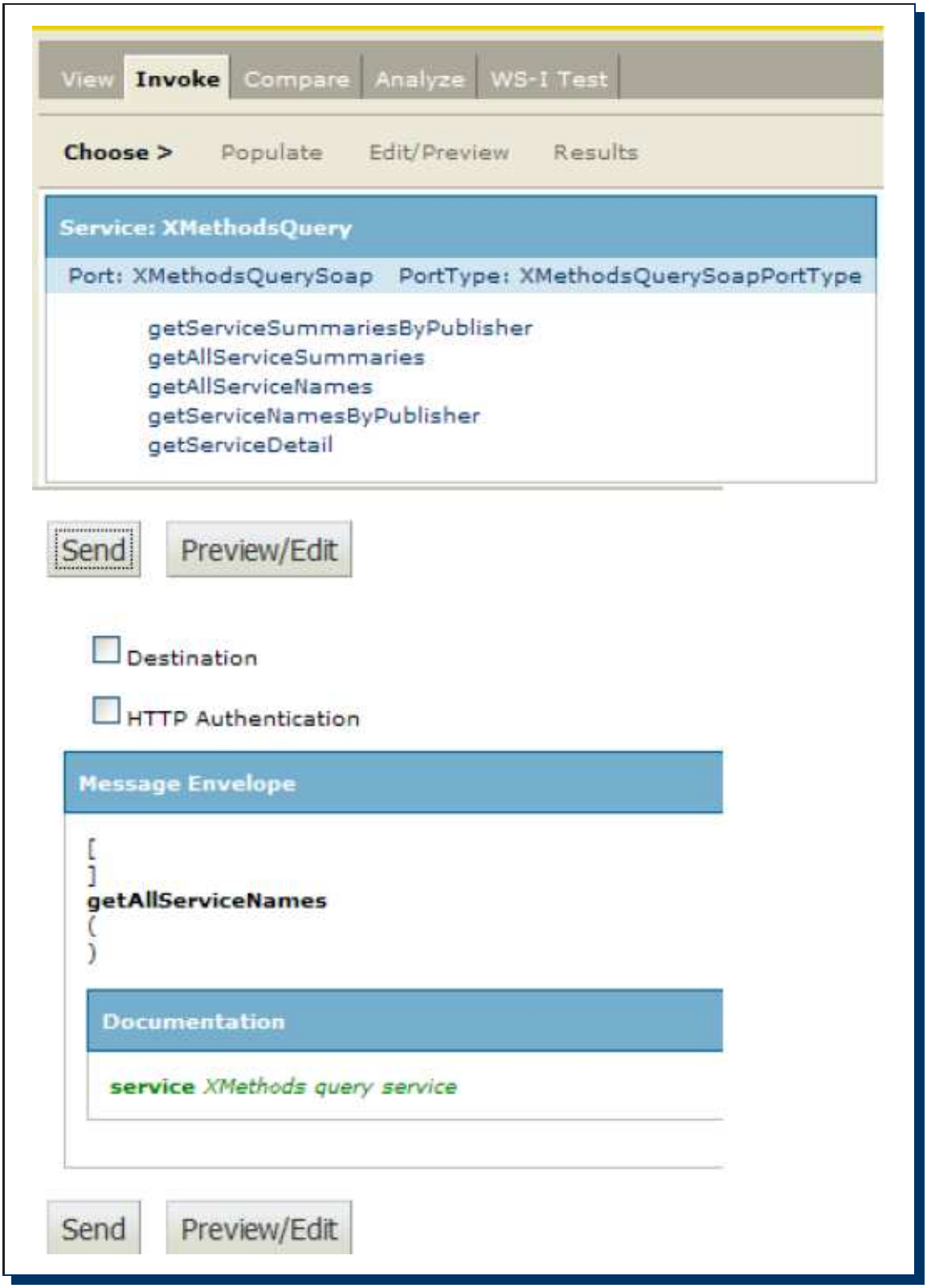

Figure 3.3: The browser-based user interface of the XMethods Query Service

presented as a “mixed form” of the whole UDDI-API. Someone who is familiar with the second user interface still needs to learn how to use the SAP UDDI query service and vice versa.

3.1.2 Distribution of Web service definitions over many UBRs

Today, a service provider has a wide range of different UBR nodes to chose for publishing his Web services. This leads to the fact that the Web services are spread over many UBR nodes, hence a service user will not get an optimal search result as he wishes and he is often forced to visit more than one UBR node to obtain an “accurate” and “complete” (see chapter 6.2) search result.

3.2 Inability to control the Web service data in the UDDI registry

“The UDDI - The Weather report”[17] shows, that “48% of the production UDDI registry (tModels tested only) contains links which are unusable. These pointers include missing, broken or inaccurate information”. Generally, the problem of an UDDI registry is that it represents a kind of “newsgroup”, in which everyone can place information about his business. Although every service provider is identifiable by an unique key within the “<BusinessEntity>” element, thoughtless operations like repeated changing or replacing of the service information can lead to an inconsistent state of the whole Web service content within the UDDI registry. Other examples of such operations are duplicates of “serviceKey” elements, incorrect references or even “data anomalies”, which are characterized by no longer unique “serviceKey” or “<tModelKey>” within a “<BusinessEntity>” element. Furthermore, there are many entries in an UDDI registry which do not contain Web service data, but HTML based references pointing to normal webpages. The Section “Category Research Summary” in the “UDDI - The Weather report” [17] with the appropriate graphical statistic data in Figures 3.4 and 3.5 confirms this statement.

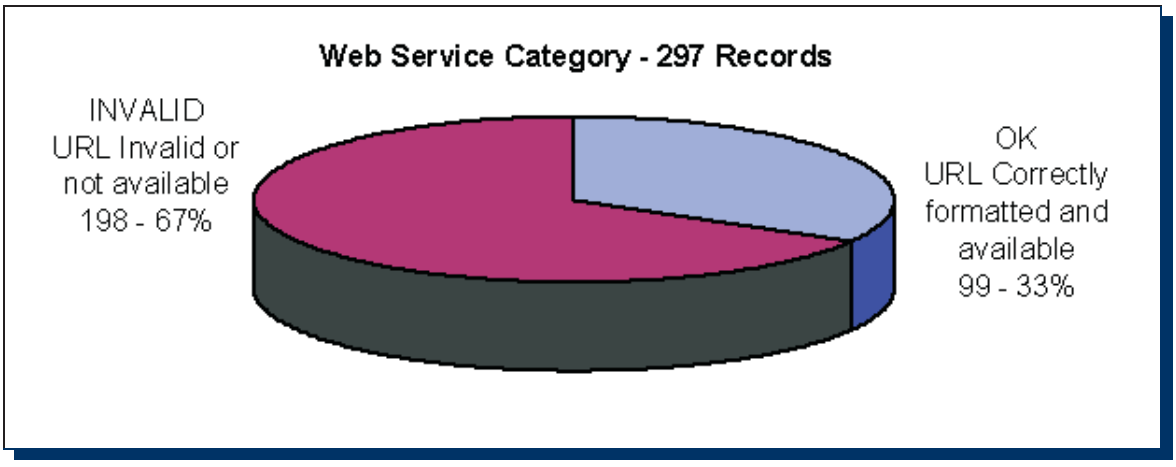

Figure 3.4: URL problem in UDDI registries

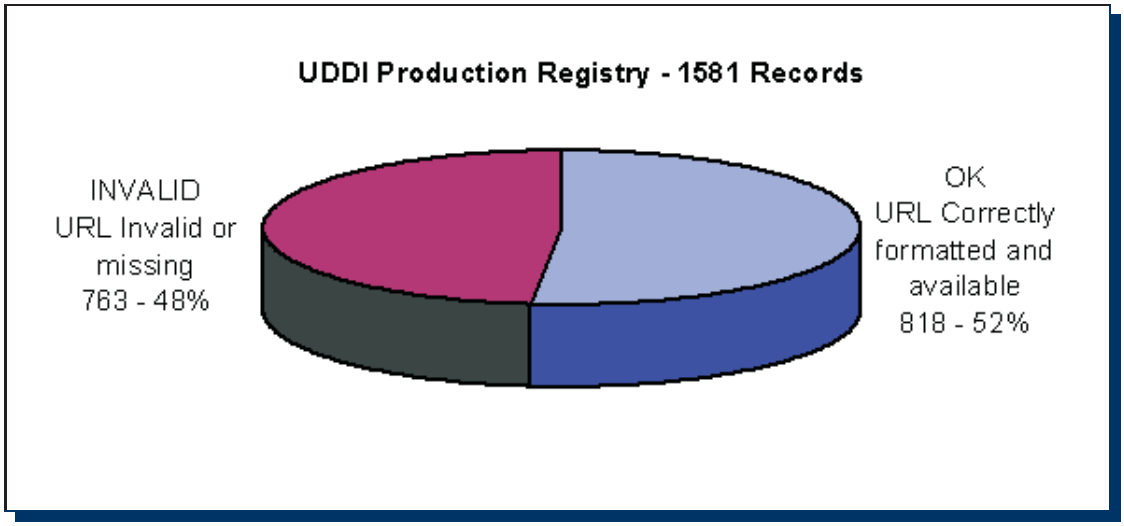

Figure 3.5: URL problems in UDDI the production registries

3.3 Non-authorised use of Web service data (against the intention of service providers)

Not only UDDI operators want to have control over the Web service data in their UBR node, but also service providers often want to protect information about their Web services and the company itself against unauthorised users. Indeed, there is a possibility of abusing the Web service information by a third party, e.g. a service provider can use the logo, contact information and description of a well-known and reliable company to advertise its own business. If a user is searching for that business, he gets both companies as results, but the second one has provided a different address within the "<overviewURL>" element which refers to a bad service. In worst case the customer uses this service and is betrayed because of bad service quality or high service price or confidentiality rupture. Hence a mechanism, which allows the service providers to specify options how their Web service data should be treated, would provide more flexibility and security when exposing the data to a third party. Thus, authentification and authorization are important and there are many approaches available, e.g. authorization through Certificates or Access Control List (ACL). This thesis also presents a mechanism to restrict the use of the services by offering service providers the possibility to specify appropriate parameters in their profile.

3.4 Limited ontology (vocabulary) for representing service information in the UDDI registry

The information about the service providers and their Web services are managed by the UDDI registry using the "UDDI data types" (see chapter 4). It is a combination of technical data from the Web service description and basic information about the service provider and its business. However, the ontology defined in the UDDI standard to represent the service content

is limited. The article "A Model for Web Services Discovery With QoS"[18] states that "The other shortcoming of the current UDDI model is that it limits the service discovery to functional requirements only". Therefore, service users either do not get enough information about the provider's business or do not get a satisfactory search result. For example, it would not be possible for a service user to search for certain Web services which are newer than a given date. On the other hand, there are limitations for service providers specifying more meaningful information for their services like specifying parameters to restrict the usage of their services. This thesis provides an approach to solve this issue by adopting service customer and provider profiles which contain an additional and meaningful vocabulary to give customers and providers more options to deal with the Web service information (see the example profiles in the chapter 6.2.0.2.2).

3.5 Timeliness and consistency issues

The general desire of a service provider who offers a Web service is to keep the data refering to the offered service always up-to-date. In order to ensure this, after each change of a service the associated information should be updated as fast and effectively as possible. Thus, such services which are changed frequently, require higher flexibility and efficient handling during the execution of the change operation so that the service client can always use the correct version of the service application without any technical problem. The article "Versioning of Web services, Solving the problem of maintenance"[19] shows that this problem cannot be solved effectively by the UDDI registry .

3.6 Performance related issues

The concentration of the Web service information in a UDDI registry forces service users to contact it in order to get the necessary information for using the service. The procedure *"search for Web service information and then use the Web service"* must be repeated each time if the service user wants to keep the service information always up-to-date and did not cache it from an earlier operation; this causes the following disadvantages:

- *Possibly long response time on the broker side:*
  This problem can arise, if many requests are made by the service customers to the registry at the same time. Thus, a long waiting time is possible, which can have a negative effect on the execution process of the service application. Current measurement shows, that an UDDI registry needs more than one second for a search request (see chapter 6.2.1.6).

- *Unfavorable net traffic at operation time on the service customer's side:*
  Sometimes service customers also get problems with their own local network. Especially, if a service customer needs a service at a time when his network is loaded by accesses of his users. As a consequence the service customer will not be provided immediately with the necessary service information. With our solution, the broker sends *proactively* the Web service information to the service customer so that the customer does not need to search for it (see chapter 5).

- *Big effort for filtering the search result because of non-relevant data:*
  Another problem with searching the Web service information is data irrelevance. This results due to the state of data in the UDDI registry and/or due to the search query sent by the service clients, if it is faulty or imprecise. The consequence is the large portion of data which are of no interest and must be segregated "manually" by the service customer. The effort for the segregation process is often much more than for the execution of the service application itself, and it delays the service execution.

- *No possibility for service customers to register themselves for desired services:*
  In addition, service customers do not have the possibility of registering themselves to be informed about changes of the Web services, e.g. technical changes or business condition changes like contract regulations. In general, this leads to the Publish/Subscribe model, which is not yet supported by the current UDDI standard. We will introduce an innovative concept which improves the correctness and the completeness of the search result (see chapter 5.3).

3.7 Résumé

The issues with a UDDI registry based on the current UDDI standard described in this chapter urgently need a comprehensive solution which has to consider the functional as well as the performance related problems. Before we present our solution, it is meaningfull to examine existing works on this topic. The next chapter gives an overview about the current evolutions regarding this topic and shows that they are hardly applicable for the solution of our problem.

Chapter 4

Current Evolutions

Currently the Web services community does not pay much attention to the solution of the described problems. Instead, the focus is on ontological metadata specifications and in particular on the semantical aspects in order to specify Web services in a machine-readable and interpretable way. Thus, the automatic interpretability of the non-functional aspects associated with a Web service like "Quality of Service (QoS)" or "Service Level Agreement (SLA)", which describes the mutual acknowledgment between both business partners to get or pay penalties in case of unfulfilled agreements, is actually considered to be more important than the management and retrieval of the Web service information. The following section describes the current evolutions by presenting the state of research in this problem domain; however, they do not contribute directly to solve the issues mentioned in the previous chapter.

4.1 Ontology Web Language for Web services (OWL-S)

4.1.1 Concept

OWL-S[20] (formerly DAML-S) is a set of markup language constructs which can be used by a service provider to describe the properties and capabilities of its Web services in an unambiguous and computer-interpretable form. With OWL-S it is possible that the Web service tasks including discovery, execution, interoperation, composition and execution monitoring can be performed in an automated way[21]. The main goal of using OWL-S is to give Web services better semantics so that beside "which" and "where" also the question *"how the Web service application works"*, e.g. what the execution steps of the application do and under which conditions the execution is allowed, is answered. Since a Web service can be described due to information by the OWL-S ontology more clearly and the data about a service provider are specified more precisely (e.g. by "Service Profiles") the risk of a mistake or "data inconsistency" (see chapter 3.5) is smaller. With OWL-S used for specifying the application process flow and also the conditional constructs, certain agreements about technical and content related aspects between service customers and the service providers can be negotiated before using the service. With such constructs within a "ServiceModel" shown in Listing 4.1 the service customer is better informed about what he has to expect during the execution of the service application process.

```
1  <process:AtomicProcess rdf:ID=''Purchase''>
2
3  ...
4
5    <process:hasInput>
6      <process:Input rdf:ID="IDNummer"/>
7    </process:hasInput>
8
9    <process:hasResult>
10     <process:Result>
11       <process:hasResultVar>
12         <process:ResultVar rdf:ID="TimeLimit">
13           <process:parameterType rdf:resource="&time;
14             #TemporalEntity"/>
15         </process:ResultVar>
16       </process:hasResultVar>
17       <process:inCondition expressionLanguage="&expr;#KIF"
18         rdf:dataType="&xsd;#string>
19          (and (cuurent-value (time-limit ?IDNumer)
20            ?TimeLimit)
21              (>= ?TimeLimit ?purchaseAbo))
22       </process:inCondition>
23     </process:Result>
24   </process:hasResult>
25
26 ...
27
28 </process:AtomicProcess>
```

Listing 4.1: Conditional information within a ServiceModel in OWL-S

In the data structure shown in Listing 4.1 the service client must specify an ID number (line 6) as an "<Input>" parameter, which is verified by the server for accessing the service. An expiry time (line 12) for using the service is assigned to the ID number of the service customer. The "<process:inCondition>" tag contains instructions to verify the validity of the ID number of the service customer and of the service time limit (line 19). With such semantic constructs in OWL-S, the question *"how the service is used"* is also answered. The more important aspect of OWL-S in the context of this thesis is the partition of a whole OWL-S document in three different parts, namely "ServiceProfile", "ServiceModel" and "ServiceGrounding". Hence, it promotes a better management of the Web service information. Figure 4.1 shows the structure of the OWL-S-based Web service specification.

The partition of an OWL-S based Web service document alleviates the service provider by structuring the service information, e.g. updating of the service provider's information only needs to modify the part "Service Profile" residing in the "profile.owl" document.

4.1.2 Conclusion

With the partition of an OWL-S document into different logical parts a certain level of "efficient management" of the Web service information is achieved. However, this approach does not respect other issues like if some information on the service provider is changed or if the service provider wants to restrict the use of his Web services to certain users or user groups.

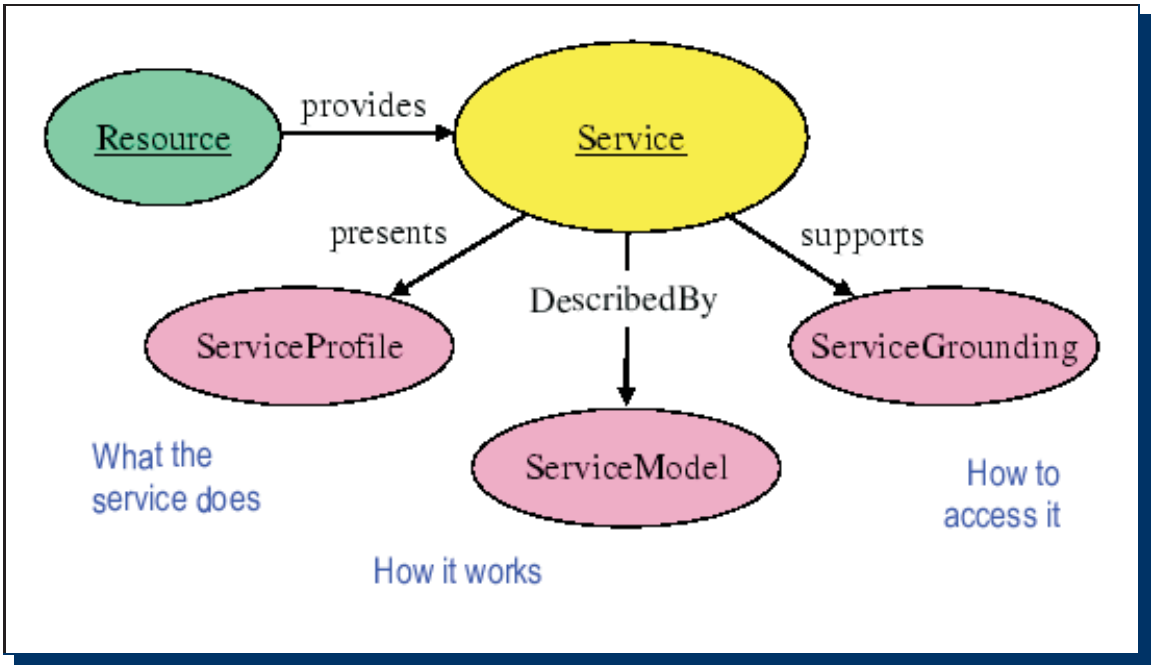

Figure 4.1: Structure of the OWL-S-based Web services description[20]

In the next section we consider another already existing way for efficient management of Web service information by linking the Web service information directly from Web-based resources like HTML documents.

4.2 Web services Inspection Language(WSIL)

4.2.1 Concept

"WSIL[22] is an XML document format to facilitate the discovery and aggregation of Web service descriptions in a simple and extensible fashion. While similar in scope to the Universal Description Discovery and Integration (UDDI) specification, WSIL is a complementary, rather than a competitive, model to service discovery"[23]. In contrast to a service entity (presented by the "BusinessEntity" element) within an UDDI registry, a WSIL document does not contain any functional information about a Web service, it rather presents descriptive information which has to be read and interpreted by human to learn more about the service.

Generally the use of WSIL is based on the existing relationship between business partners[24], which know each other and their addresses, so that a direct use of a Web service without an UDDI registry as service broker is possible. The service customer knows the Web service address of the service provider described in a WSIL document through an Internet-URL like "http://[domain of service provider]/[optional server context path]/inspection.wsil". A WSIL document itself is XML based and consists of the elements specified in Listing 4.2.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <inspection xmlns="http://schemas.xmlsoap.org/ws
3    /2001/10/inspection/">
4
5    <abstract>
6      Acme Industries Public Web Services
7    </abstract>
8
9    <service>
10     <name>Store Finder Service</name>
11     <abstract>
12       A service to perform a geographical search of Acme
13       store locations.
14     </abstract>
15     <description referencedNamespace=
16       "http://schemas.xmlsoap.org/wsdl/"
17       location="http://example.org/services/storefinder.wsdl">
18     </description>
19   </service>
20
21   <link referencedNamespace="http://schemas.xmlsoap.org/
22     ws/2001/10/inspection/"
23       location="http://example.org/services/ecommerce.wsil">
24     <abstract>
25       Acme Industries Public e-Commerce Services
26     </abstract>
27   </link>
28 </inspection>
```

Listing 4.2: Example of a WSIL document[25]

- The "<abstract>" element (line 5) contains a short description of the Web service.

- The "<service>" element (line 9) contains beside other sub-elements the link to the ".wsdl" document specified by a full URL.

- There are one or many optional "<link>" element(s) (line 21) with a location-attribute referring to other WSIL documents

Listing 4.3 shows how a WSIL document can be used within the header of an HTML web page and Figure 4.2 illustrates how WSIL works in the practice .

```
1  <!doctype html public "-//W3C//DTD HTML 4.01 Transitional//EN">
2  <html>
3    <head>
4      <meta ...>
5      <meta name="serviceInspection" content="http://www.example.com/
6        inspection.wsil>
7    </head>
8
9     <body>
10      ...
11    </body>
12 </html>
```

Listing 4.3: WSIL document within an HTML web page

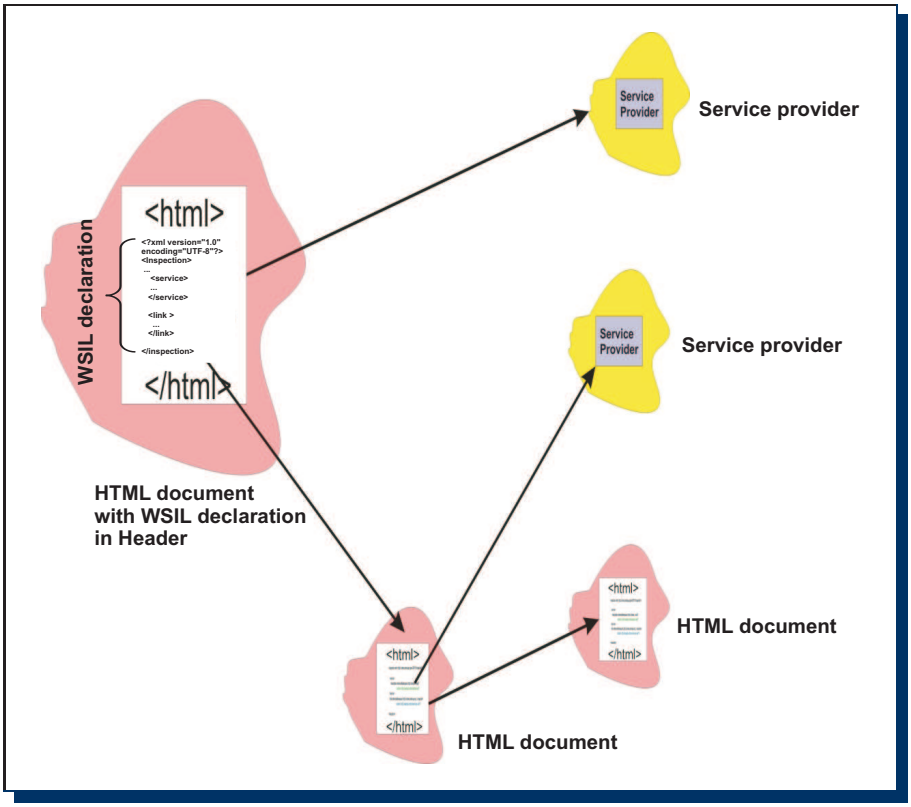

Figure 4.2: Use of a WSIL document

The WSIL content within the header of an HTML document either links directly to a WSDL document residing at the service provider or to other HTML documents containing further WSIL information.

In the example above we assume that the WSIL document is named as “inspection.wsil” and is placed directly under the URL “www.example.com” (line 8,9). A WSIL document can also be placed in any document type with a proper structure and meta data, which only needs to be found and interpreted by a software agent.

The use of WSIL releases the Web service programming model from the inherent dependence on the concept of the UDDI. With WSIL, a decentralized management of Web services information on the side of the service provider becomes possible. This enables also a direct communication between service customers and service providers. The substantial flexibility of WSIL is that it can be associated with HTTP based documents, which can be easily found and interpreted by a webcrawler, e.g. web search engine. This can be realized by the service provider by placing the appropriate <meta> tag within a HTTP document[25]. WSIL is also extensible to other specifications or standards like WSDL and UDDI. It means that within a WSIL document one cannot only refer to URL based resources but also directly to service points (aka. service interfaces) or to information within an UDDI registry. Listing 4.4 demonstrates the use of WSIL in combination with the WSDL and UDDI specifications.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <inspection xmlns="http://schemas.xmlsoap.org/ws/2001/10/inspection/"
3   xmlns:wsilwsdl="http://schemas.xmlsoap.org/ws/2001/10/inspection/wsdl/"
4   xmlns:wsiluddi="http://schemas.xmlsoap.org/ws/2001/10/inspection/uddi/">
5
6    <abstract>Acme Industries Public Web Services</abstract>
7
8    <service>
9      <name>Store Finder Service</name>
10       <abstract>A service to perform a geographical search of Acme
11                 store locations.</abstract>
12       <description referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
13                 location="http://example.org/services/storefinder.wsdl">
14       <wsilwsdl:reference>
15         <wsilwsdl:referencedService
16             xmlns:ns1="http://example.org/services/storefinder.wsdl">
17             ns1:StoreFinder<wsilwsdl:referencedService>
18       </wsilwsdl:reference>
19       </description>
20   </service>
21
22   <link referencedNamespace="urn:uddi-org:api">
23       <abstract>Acme Industries Public e-Commerce Services</abstract>
24       <wsiluddi:businessDescription
25                 location="http://example.org/uddi/inquiryapi">
26         <wsiluddi:businessKey>3C9CADD0-5C39-11D5-9FCF-BB3200333F79
27                   </wsiluddi:businessKey>
28         <wsiluddi:discoveryURL useType="businessEntity">
29           http://example.org/uddi?3C9CADD0-5C39-11D5-9FCF-BB3200333F79
30         </wsiluddi:discoveryURL>
31       </wsiluddi:businessDescription>
32   </link>
33
34 </inspection>
```

Listing 4.4: Combination of WSIL with WSDL and UDDI specifications [25]

In the example we are using the extension element "<wsilwsdl:reference>" (line 14) to point directly to the "StoreFinder" interface of the service provider "example.org". The "<wsiluddi:businessDescription>" element (line 24) is used to refer directly to the business information in an UDDI registry.

4.2.2 Conclusion

One can consider WSIL as a kind of "Resource Simple Syndication (RSS)" for representing and delivering Web services in a compact way. For a service customer a WSIL document serves as a reference pointing to Web services without being dependent on the existence of an UDDI registry. This approach has the advantage that a service provider itself can manage the information about his Web services and keeps them always up-to-date. Hence this solution addresses the timeliness aspect of the Web services information (see chapter 3.5) and enables an optimal management of the Web service information at the service provider. However, WSIL presupposes that the business partners mutually know each other and that one has to first find the right document containing the WSIL link in order to reach the final WSIL document. Furthermore the service

provider has to update the links in one or many affected HTML web pages if the addres of any WSIL or the WSDL document is changed. Hence for an automation of the Web services application process in the actual sense, WSIL does not present an adequate solution under a long-term perspective.

4.3 Two-Level UDDIs

4.3.1 Concept

The idea behind this approach is to divide the UDDI registries in to two types[26]:

i. Sandboxes:

They are used for development purposes and a wide range of users have the right to work with them. Service developers can register their services and execute operations in the sandboxes whenever they wish without caring about affecting the registry. After a service is properly tested and is considered as ready for the production, the developer can notify the UDDI system administrator to migrate it to the so-called "Production registries".

ii. Production registries:

They are specially assigned to system administrators with limited right and are used for the Web services which are on the production-ready phase. System administrators first check for the stage of the maturity and the quality of the Web services from the sandboxes to see whether they fulfil the policies or production specification of their organization, and then move them in the (final) production registries[1]. The relation between "sandboxes" and "production registries" is illustrated in Figure 4.3.

With the separation of the sandboxes from the production registries an UDDI registry operator has more control over how the registry is used. Both developers and system administrators benefit from flexibility and autonomy, because they can work independently from each other with their appropriate access rights given by the system administrator. On one side, developers have the freedom to test their products as they want and can be sure that the other services are not affected. On the other side, with the right limitation system administrators do not have to care about the non-authorised operations executed by other (not allowed) users, hence they can always hold the production registries in a "clean" status.

4.3.2 Conclusion

The use of "two level UDDI registries" contributes to the solution of "uncontrollability of the Web services" issues; however this approach still has the following disadvantages:

- The separation of the two types of UDDI registries requires a well designed plan from the UDDI site operator considering which Web service version should be offered in "sandboxes", and which in "production registries"

[1]It is clear that the system administrator can only check the correctness of the Web services up to a reasonable level but not the whole functionality of every service because the set of the results is too big

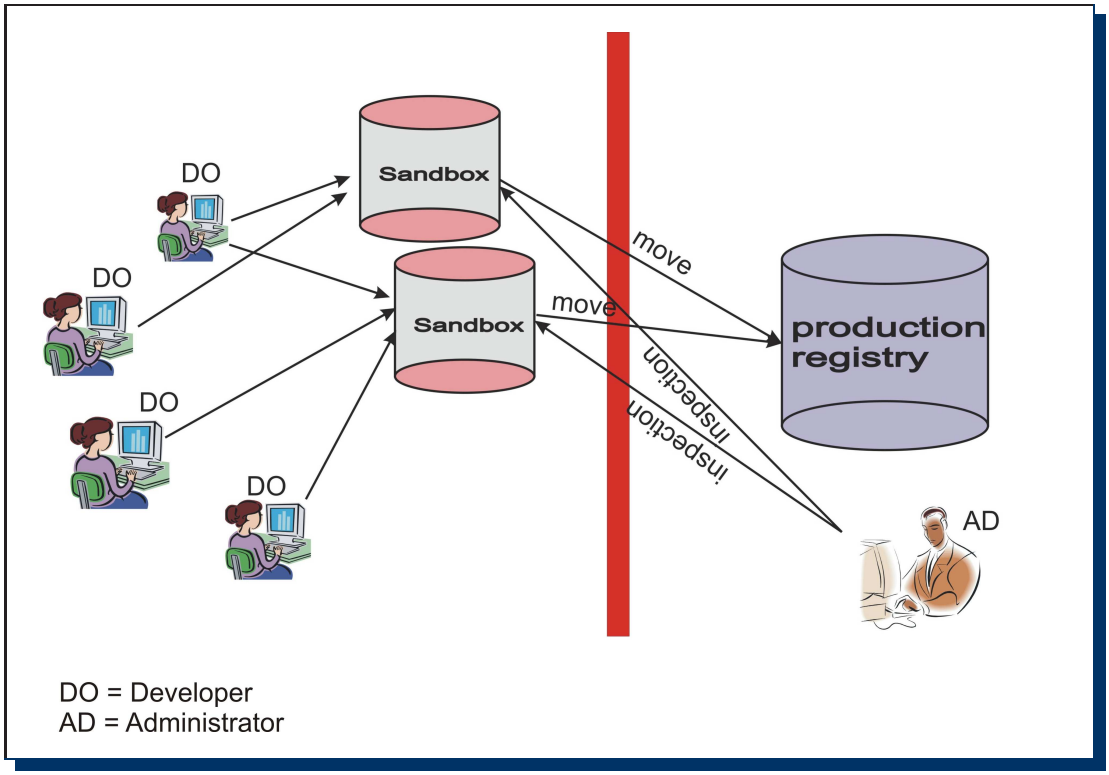

Figure 4.3: Two-Level UDDI registries

- More UDDI registry instances also result in higher cost. Beside the financial aspect there is also more complexity and effort necessary to put the whole infrastructure into operation.
- The maintenance of separated system is more complex and more fault-prone.

Despite the mentioned contribution of this approach to our initial problems, it hides some enterprise-critical disadvantages which have to be handled in an appropriate way. Hence this solution is not suitable for our problem and we have to look for a better one.

4.4 Combination of Peer-to-Peer (P2P) and SOA

4.4.1 Basic

> "A pure peer-to-peer network does not have the notion of clients or servers, but only equal peer nodes that simultaneously function as both "clients" and "servers" to the other nodes on the network. This model of network arrangement differs from the client-server model where communication is usually to and from a server.[...]"[28]

In addition, every peer manages itself in a way that special server instances as "mediators", which have to forward the inquiries and answers, can be avoided. The "symmetry" plays a

substantial part regarding both the distribution of roles and the resource administration of the individual peers, which differentiates P2P from other architectures.

From this view we use the following implied properties of P2P to combine it with the SOA to address the consistency and timeliness issues mentioned in the chapter 3.

a) every peer in the network environment can act as a communication peer, thus a sender and receiver and forwarder of any data

b) every peer can itself hold and manage the service information needed by the application

The next question is how do we apply P2P on SOA to construct a solution for our problems. To reduce the complexity, we divide the answer for this question into two parts: In the first part "concept" we construct the approach and the process by presenting the technologies and the way how P2P and SOA are combined with each other to build a Web services environment to address our problems. The second part "implementation" demonstrates this idea with a practical example by using the JXTA[27] as a P2P framework and shows finally to what extent this "combination" approach actually is applicable for the solution of our problems.

4.4.2 Concept

Figure 4.4 shows the network infrastructure in a traditional SOA-based Web services environment, where an instant direct communication between service providers and service clients is not possible, because the address of the providers is not *a priori* known to the clients. The other issue is that all the information needed by the service clients to use the services is concentrated on the service broker(s) (in our case UUDI registries). Hence we have an "asymmetrical" communication model resulting from the roles assigned to the involved service entities in a Client-Server relation:

a) Service providers (SP) are Clients (C) to the Service brokers (SB)

b) Service customers (SC) are clients to the SBs

c) SC are Clients to the SPs if SC access their services

Ideal is a Web services environment where every SC can always get the Web service information directly and on time from the SPs and also manage it for its own use without help of a SB. Under this assumption we would have a "symmetrical" communication between all involved service entities. P2P can realize this concept, where SB(s) can still work as super node(s) for searching Web service information. Hence, the SOA-based Web services environment in Figure 4.4 can be reconstructed to a SOA-P2P-based Web services environment like in Figure 4.5. where each SC knows each SP and SB so that a direct, immediate and bidirectional exchange of any messages, e.g. notifications of new Web services or changes of existing Web services to SC, can be performed without any "third party" or additional time overhead. Altogether a P2P-SOA based Web service architecture would bring the following capabilities:

a) Every host acts as a Client as well as a Server

b) Direct communication between SCs and SPs is possible (the intermediate role of the broker between SC and SP is no longer necessary)

c) Every host manages the Web service information itself

d) Web service information is available at different hosts

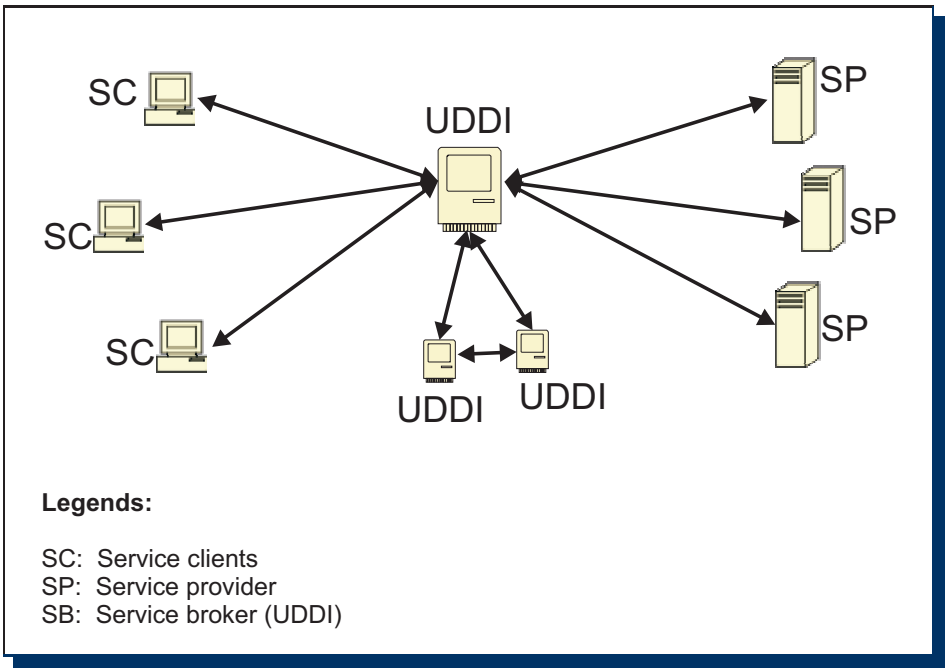

Figure 4.4: Conventional SOA-based Web services environment

The mentioned capabilities cause the following advantages in contrast to a traditional SOA-based Web services architecture with central broker:

- The "single point of failure" problem through a SB can be avoided, because SCs can get the service information directly from the SPs
- Since published service information is available on different peers, every SC can get it directly from the nearest SC or SP (e.g. via multicasting) which has this Web service information
- SPs can use the "peer-to-peer" property to send every kind of messages to the Clients

To realize this concept with these advantages the paper "Scalable and Ontology-Based P2P Infrastructure for Sematic Web Services"[29] already presents an approach which uses a Hypercube overlay topology based P2P infrastructure to organize and search Web services in an efficient manner. This paper describes how peers are organized into an Overlay Hypercube Graph so that the longest search path can be limited to $\frac{1}{N-1} \cdot \sum_{i=1}^{\log_b N} \frac{(b-1)^{\log_b N-i+1}}{(log_b N-i)!} \cdot \prod_{j=0}^{\log_b N-1}(i+j) \approx 0.5 \cdot \log_b N < N$, where N is the total number of the (host) nodes and is also the longest path in a traditional P2P network which has to be traversed within a search process in worst case.

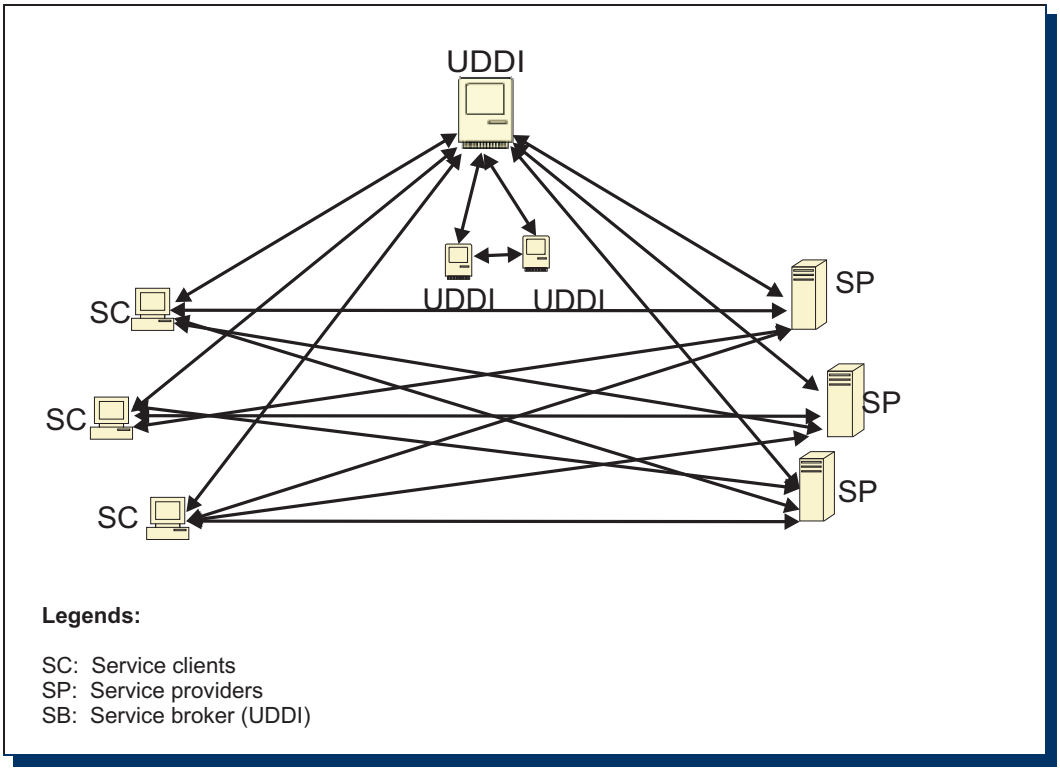

Figure 4.5: SOA-P2P based Web services environment

In general one can conceptually construct a network infrastructure combining the advantages of P2P and SOA together and optimize it in an appropriate way to limit the traffic load generated by the application operations, e.g. sending or forwarding a request for a resource. However, there are also other criteria affecting the whole performance of the application process, e.g. organizing the peers within the Hypercube topology based P2P network causes an additional complexity of $O(log_b N)$ for reconstructuring the Hypecube topology if a peer comes to or leaves from the network.

There are also other issues which are not visible at the conceptual constructing phase but rise during the realization (implementation) and operation time of the network. The next part describes such aspects which are not concerned or not visible during the conception of the P2P-based Web service network but cause import issues regarding the network management and application performance at the network operation time. We do this by giving an example (section 4.4.3) and pointing out the problems related to it (section 4.4.4).

4.4.3 Implementation

The previous section describes the concept of how to combine P2P with SOA to construct a Web services environment with a “symmetrical” communication model, where every entity has the capability to act as a client as well as a server. In this section we take a closer look into the technical aspects of this combination by considering an existing example based on

JXTA, a Java-xbased P2P framework and GLUE[30], a Web service framework. In the short article “Peer to Peer in Theorie und Praxis, Ad-hoc-Web-Services durch P2P-Technologien”[31] we find a JXTA-GLUE based realization of a P2P-SOA-based Web service application. Listing 4.5 shows the source code of the “PeerChainApplication”. The process of the application contains the main steps residing in the “public static void main()” method (line 37).

```
1  public class PeerChainApplication {
2  private Set foundPeers =
3  Collections.synchronizedSet(new HashSet());
4  ...
5    public void discoveredPeer(String wsdlURL) {
6      if (foundPeers.add(wsdlURL))
7        System.out.println("discovered new peer=" + wsdlURL);
8    }
9
10   private void startPeerChaining() throws RegistryException
11   {
12     PeerChainService peerChain =
13       PeerChainService)Registry.bind(myWSDLUrl,
14         PeerChainService.class);
15     peerChain.chainPeers(new ArrayList());
16     // start chain with empty list
17   }
18
19   private DiscoveryService startJxta()
20     throws PeerGroupException {
21     returnPeerGroupFactory.newNetPeerGroup()
22       .getDiscoveryService();
23   }
24
25   private PeerAdvertisement
26     createAdvertisement(String wsdlURL) {
27     PeerAdvertisement adv =
28     (PeerAdvertisement)AdvertisementFactory
29       .newAdvertisement(
30     PeerAdvertisement.getAdvertisementType());
31     adv.setName("Java Spektrum P2P Example");
32     adv.setDescription(wsdlURL);
33     // description is web service URL
34     return adv;
35   }
36
37   public static void main(String args[]) throws Exception {
38     int port = 8004; // default port
39     ...
40     HTTP.startup("http://localhost:" + port + "/P2P");
41     // start GLUE HTTP server
42     PeerChainApplication app = new PeerChainApplication();
43     // publish peer chain web service on GLUE HTTP server
44     app.myWSDLUrl =
45     PeerChainServiceImpl.publish(app, port) + ".wsdl";
46     // publish JXTA advertisement that contains
47     // peer chain web service url
48     DiscoveryService discoSvc = app.startJxta();
49     discoSvc.addDiscoveryListener(new Listener(app));
50     PeerAdvertisement adv =
51     app.createAdvertisement(app.getWSDLUrl());
52     new Publisher(adv, discoSvc).start();
53     while (true) { // start peer chaining periodically
```

```
54        Thread.sleep(app.randomInt(MAX_SLEEP) * 1000);
55        if (!app.foundPeers.isEmpty()) app.startPeerChaining();
56        }
57     }
58  }
```

Listing 4.5: Example "PeerChainApplication" based on JXTA-framework

The "PeerChainApplication" consists of the following statements:

- Publishing a Java-Object as a Web service with GLUE (line 45), while a URL to the WSDL document is generated
- Notifying the other peers about the new Web service (line 48): starting of JXTA and generating a "PeerAdvertisement", which contains the URL to the WSDL document known from the previous step. Starting a separate thread which continually publishes the generated "PeerAdvertisement" in a certain time interval.
- Registration of a "Listener" for arriving discovery events (line 49); extracts the "PeerAdvertisement" from it and checks whether the "PeerAdvertisement" contains an URL to a WSDL document, and if any exists, a call to the Web service interface can be performed.

The "PeerChainApplication" is a simple example of the original idea of how a peer can act as service entity in a client as well as in a server role. In this example a peer can publish Web service advertisements and also listen to arriving advertisements and perform a call to the Web service referenced by a WSDL-URL. An "Advertisement" publish operation in this example sends the same information about the Web service to all peers, for what a multicasting approach can be used. In this manner we have a kind of "self-management" of the Web service information by the peers, hence it is not necessary to have a service broker like an UDDI registry, and if a peer crashes, the published Web service information could still be found on other Peers. The concrete implementation of the corresponding Java class based Web service objects can be found in the denoted literature or in the appendix D.

The "PeerChainApplication" is an example to show how P2P can be combined with Web services to publish and search service information without a service broker. Other P2P-SOA based applications which do not concentrate on retrieving service information, but rather concern the possibilities for efficient Web service composition, are "Binding- and Port-Agnostic Service Composition using a P2P SOA"[32] and "SwinDeW-B: A P2P Based Composite Service Execution System with BPEL"[33]. The first work presents an approach for executing composed Web services with dynamic hosts which do not need to have a public, static IP, which is usually required within a Web service environment. The second example addresses the weakness of the current workflow engines, which are based on centralised Business Process Execution Language for Web Services (BPEL)[10], regarding performance, scalability and shows how P2P based decentralised Web services engines can be created to solve these problems.

The three presented P2P-SOA based applications have demonstrated the possibility of combining of Web services with P2P. Applying the combination can not only be used for retrieving Web service information but also for executing of Web service processes. However, each approach brings besides advantages also disadvantages, e.g. the application mentioned in [32] can cause "single points of failure" by using the "Service Proxies" and both applications [32] and [33]

have to deal with communication overheads resulting from processing costly XML messages and multicast operations.

4.4.4 Problems

In general, all the communication processes performed between the peers within a JXTA network environment are based on the protocols described in the JXTA Protocols Specification [34]. Among protocols [35] the "Endpoint Routing Protocol (ERP)" is used by the peers to find a certain host node in the JXTA network environment. A peer technically uses the IP-Multicasting as discovery mechanism to detect the others in the same sub-network[31]. The short article "Web Services and Peer-to-Peer Computing"[36] points out the following issues rising from this context:

i. Network-related issues

(a) Bandwidth and scalability issue

The use of P2P and Web services based applications may slow down the network connections and hamper the core business activities, especially if these applications are designed for content retrieval operation, in our case for a specific Web service, because in a worst case all the peers in the network have to be visited until the entry is found. Thus a lot of network bandwidth is used which can affect other applications. The IP-Multicasting as discovery mechanism is only applicable for a small and manageable sub-network. In case the company's network grows and its application system becomes more distributed, searching for a certain information may take a long time and the network itself may mostly be busy with search processes.

One can construct a hybrid P2P network in a Wide Area network (WAN) with a supernode as central point where service customers and service providers can register themselves with a profile so that a service customer can be notified about new services and can contact directly the service provider. However, this approach could encounter the following issues:

- *Service customer could not be found*
  This case can happen if a match between the customer's and provider profiles occurs and the supernode sends the appropriate service information to the service client. Assume that this service client has left the P2P network meanwhile and it has an other IP address after rejoining the network.
- *Service provider could not be found*
  Analogue to the previous issue a service client could not find the service provider from an earlier session if it has another IP address meanwhile.

To overcome these issues service clients and service providers can notify the supernode about their status, especially if they leave and join the network. This approach can avoid the issues mentioned above, but would cause considerable additional load for the network because a WAN is an open system and the number of the hosts is big and unpredictable. The consequence is that the supernode is loaded with handling status messages from the hosts and may not have capacity for notifying service customers about new Web services. On the other hand, the network will be overloaded with traffic caused by sending status messages between service providers, supernode, service clients and backwards. Altogether, the following issues can arise during the network operation for a classical LAN:

- **Long searching time for service information**
  In worst case, a peer has to visit all other peers to find a service and needs $o(\log_a N)$, where a is the lowest number of the neighbours of any peer in the network.
- **High network load**
  Besides the traffic caused by search and management operations, the network is heavily loaded with transports of different types of messages. With Web services, service provider peers have to publish their services to service customer peers and if any service is changed, notify messages have to be released in order to keep the service information at the service customers up-to-date. The transport of these messages will generate unavoidable traffic for the network and as this network size increases and becomes more distributed, it may be affected by poor and slow customers connections[36].
-

(b) (Network Address Translation) NAT and Firewall issue
"The JXTA Technical Overview"[27] mentions the general issue of executing P2P operations to peers behind a NAT gateway or a firewall. In most cases, in order to enable the communication between peers through a firewall a prior consultation with system administrators is necessary to let the communication traffic through (e.g. opening a special incoming port at the firewall or gateway or using a special "tunnel" mechanism). This point indeed presents a complement to the basic idea of SOAP as a lightweight message exchange protocol designed to be independent from any type of network infrastructures. Normally a SOAP based Web service application would use a HTTP-enable port to communicate with others, but in the case of a P2P network this will not be possible without an appropriate modification or reconfiguration of the network.

ii. Security issues:

"The most important feature of the P2P based application - decentralized, distributed - is also its weakest link"[36], and dynamic membership is also one of the features being weakly linked. This article points out that a P2P system could be only strongly safe if it is closed. However, if one has chosen a P2P based environment, he also wants that the peers should have the freedom regarding their dynamic existence. That means each peer can come and go as it wants and the network environment and (in our case) the distributed Web service application should not be affected negatively in any case. But this is also the general weak point of a P2P system, because every host node can be replaced with any other, which can treat the network environment (e.g. through IP-spoofing to imitate a host identity) and the application system in a malicious way.

iii. Complex architecture and difficult maintenance

In General the architecture of a P2P based application is more complex than a standard distributed one because of the following reasons:

(a) Lack or complex realization of a security mechanism to hold the system to be safe, which is already mentioned before.

(b) The host nodes (peers) involved in the distributed application system must be organized and managed in an appropriate way to make them work as a closed system. That means the application should not be affected due to the geographical distance or underlying platforms of the host nodes.

(c) A P2P based Web services environment requires complex and costly maintenance work to ensure a stable application operation. Firstly, the application system must identify the entering peers and check whether they are legitimate the system, and after any peer has left the network, the network environment still works (e.g. it does not crash because of "single point of failure") and the functionality of the distributed application is not affected. Secondly, the resources have to be replicated in a consistent way that they are up-to-date and always available. Last but not least, in an open P2P based application system failures are more difficult to locate and fix.

4.5 Résumé

Despite the reasonable concept and its advantages there are too many issues emerging during the concrete implementation and application operation of a P2P based Web service application system. In an open environment - say for business-to-business integration (B2Bi) - this combination approach is not reasonable due to the mentioned problems before; especially, it is not applicable for an open enterprise-critical system, where a high level of security and availability of the service end points are essential and necessary. However, a combination of both architectures is recommended, if the following conditions are fulfilled:

- **Restricted set of peers:**
  The set of the hosts is restricted and predictable, so that the load in worst case is bearable for the network .

- **Well organised network and good multicasting:**
  The network must be well organised in a manner that a multicast operation can reach quickly relevant services.

- **Web services with low change rate:**
  We assume that service provider peers have to propagate new services and changes of existing services to the service customer peers. Section 4.4.4 already shows that the transport of the messages caused by such activities can slow down the network in a considerable measure.

- **Consistency of the Web service information:**
  It is typical for a P2P network that the same resource is available at more than one host node. This is a good property regarding replicating the Web service information in our context. However, service information at any host node could be out-of-date if the affected service is changed. In this case, if any peer searches for the service, it could get the false information and never reach the final service interface or enter in a "exception" state. Hence, every peer must hold the service information for only certain reasonable time limit. After that, the peer either marks the affected information with additional "out-of-date" flags or searches for new information from the original service provider peer.

- **The network security is warranted**
  As mentioned before, a P2P network is easily exposed to malicious attacks because of its dynamic property. Especially the papers "Die vermeintliche Robustheit von Peer-to-Peer Netzen"[37] and "Defending the Sybil Attack in P2P Networks"[38] show how a P2P network can be attacked and abused. Thus, solid network management with appropriate

tools and strategies against unwished attacks is very important in order to keep the network secure and stable.

4.6 Requirements for a new solution

The present evolutions contribute in each case only indirectly and partially to the solution of the problems mentioned in the chapter 3. They offer advantages, which result from the application of a suitable ontology or a meaningful allocation of the Web service information. The approach presented in this work attacks directly the problem using the following approaches:

i. A service customer does not search himself for service information, but gets it automatically from the service broker (*pro-activity type I*) on the basis of a subscription

ii. The service broker not only waits for the service information, but also looks itself "proactively" for new services (*pro-activity type II*)

iii. A service provider sends changes of his profile and service data to the service broker

iv. The service broker informs service customers about changes of the service information on the service providers (Notification) (*pro-activity type III*)

v. Improved completeness and correctness of the search result

The next section presents the concept of a *Search Result Management System for Web services* and shows how it can fulfill these requirements.

Chapter 5

Our Solution - The Search Result Management System (SerumS)

5.1 Introduction

Today, a service provider can place information about its Web service in form of a WSDL document, which can be found by the search engine "Google"[1] like a normal HTML web page; SerumS integrates this search function of "Google" as a part of its whole functionality. In addition, we assume that a service provider assigns an UNSPSC[39]-based <categoryBag> identifier to its Web service, so that SerumS can determine exactly the service type of a service provider. Based on these assumptions we construct the architecture of SerumS as a Web service broker, which can be used as a replacement for an UDDI registry.

5.2 Architecture

This part presents the inner architecture of SerumS with the interfaces of the system for communicating with service customers and service providers and for handling the Web service content found on the Internet. The description of the architecture is closed with the workflow for processing the "subscribe" and "publish" requests and the pro-active search.

5.2.1 System interfaces

Figure 5.1 presents the architecture of SerumS and shows the relation between its components.

SerumS essentially consists of the following components:

i. Interfaces for search and management of Web services and profiles of the service customers and providers

   The search functionality of SerumS compares the profile of a service customer with the

[1] http://www.google.com

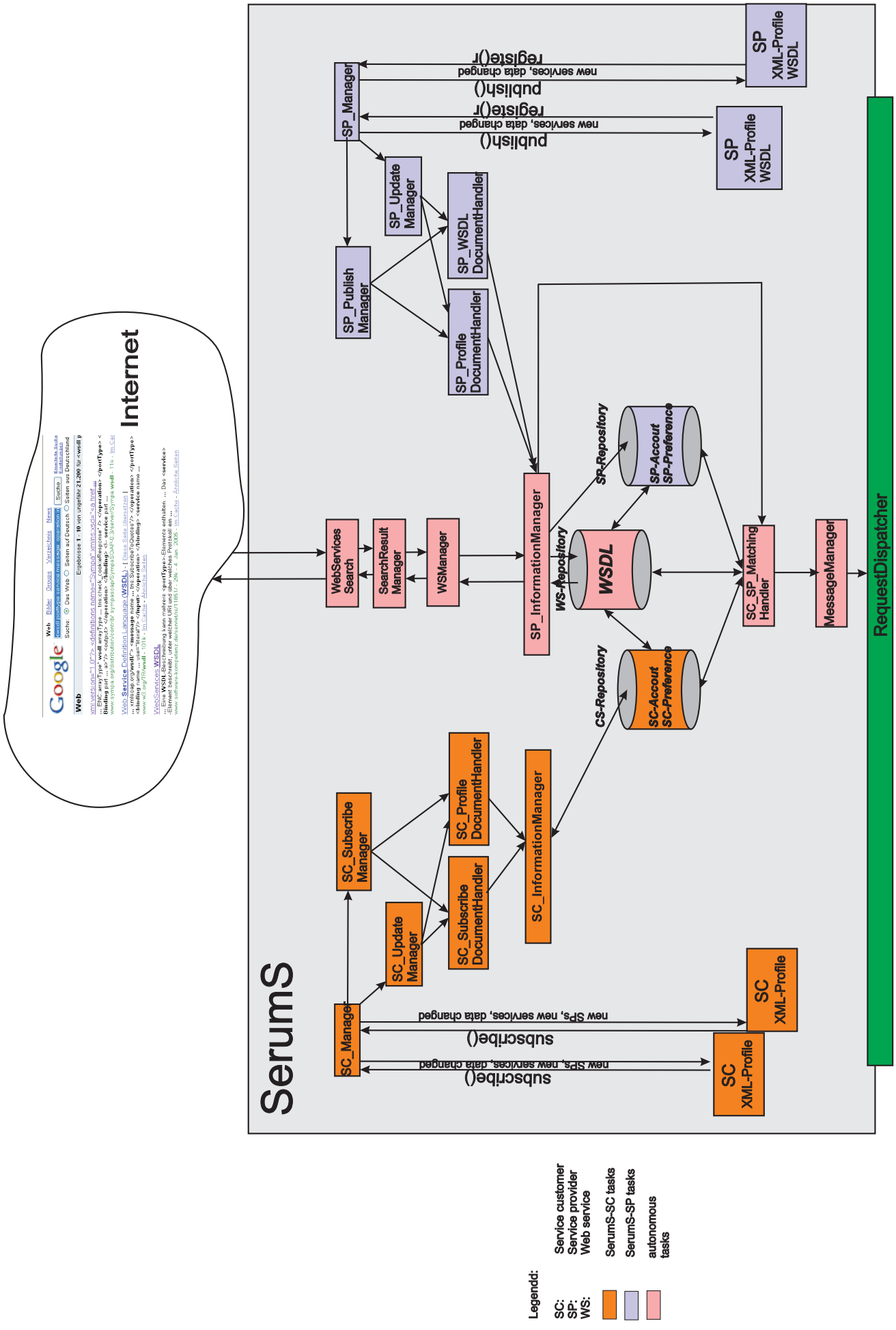

Figure 5.1: Architecture of SerumS

Web services and the profiles of the service providers so that (in the case of matching) the service customer is notified. The whole functionality consists of the following tasks:

(a) Search in the Internet for new Web services using the Google Web service based search API

(b) Analyze found Web services and prepare them for using and matching with the service customer profiles

(c) Check for the correctness of technical data in WSDL documents (e.g. existence of import-data elements and URL-references)

(d) Check provider profiles and their services to match with the customer profiles

(e) Inform service customers about new Web services and changes of subscribed services

ii. Interfaces to the service customers (SerumS-SC)

They provide the functionality for the interaction between service customers and SerumS, e.g. register routines initiated by service customers, notification about new Web services or changes of existing services and service providers to service customers. In general these interfaces offer the service customers the following possibilities:

(a) Service customers can request for topical Web services

(b) Service customers can register themselves for certain Web service(s)

(c) Service customers can send SerumS an additional profile specifying more parameters to search for and/or match with Web services and provider profiles

(d) Service customers can specify how to be notified of new Web services or changes of existing services (e.g. per Email or by a Web service based "notify" interface)

iii. Interfaces to the service providers (SerumS-SP)

With them service providers can register themselves and publish their services to SerumS. The notification about changes of the Web service data is performed by the SP_UpdateManager. These interfaces give the service providers the following functionality:

(a) Service providers can publish their services to SerumS

(b) Service providers can specify more information/technical data to the Web service via a provider profile (e.g. expiration date time of service, users/user group constrain)

(c) Update of Web service information

(d) Update of provider profile

5.2.2 SerumS workflow

In this sub-chapter the whole task workflow of SerumS is described. We distinguish between SerumS-SC, SerumS-SP and autonomous task workflows. Due to the similarity of the first two, their workflow is described together. The autonomous task is responsible for interacting with the Internet and is described separately.

5.2.2.1 SerumS-SC and SerumS-SP workflow

Figure 5.2 shows how a request from a service customer or service provider is processed through the whole system via a collaboration diagram. The important steps are executed by the following components:

- SerumS-SC
  - RequestDispatcher:

    If the system receives a request, the RequestDispatcher checks for the type of the request whether it is a request from a service customer or service provider. If the request is of type SC_Request it will be forwarded to the SC_Manager.
  - SC_Manager:

    It checks whether the request is a "New subscribe" or an "Update" operation. In case of "New subscribe" operation the SC_SubscribeManager is called, otherwise the content of the request is forwarded to the SC_UpdateManager.
  - SC_SubscribeManager and SC_UpdateManager:

    They check for the correctness of the document included in the request and call the responsible component (SC_ProfileDocumetHandler or SC_SubscribeDocumentHandler) for processing the data in the document and finally pass them to the SC_InformationHandler, which makes them persistent with the help of XMLBeans.
  - SC_SP_MatchingHandler

    The last task of SerumS is to call the SC_SP_MatchingManager for performing the comparison of existing Web services and profiles of service providers with the subscribe and profile documents of this (new) service customer. If the SC_SP_MatchingHandler finds any matching, it uses the MessageManager to send notification to the matched service customers.
- SerumS-SP
  - SP_Manager:

    If the received request comes from a service provider, the SP_Manager is called by the RequestDispatcher. The SP_Manager checks analogue to the SC_Manager the type of the operation ("New publish" or "Service update"). The content of the request is forwarded to the SP_PublishManager or SP_UpdateManager.
  - SP_PublishManager and SP_UpdateManager:

    After checking whether the request document is correct and well-formed SP_WSDLDocumentHandler or SP_ProfileDocumentHandler is called for handling its content. The SP_WSDLDocumentHandler checks the WSDL document for its validity and correctness (e.g. for URL references, imported schemas etc.) and forwards it to the SP_InformationManager, which uses XMLBeans to deserialise the XML content of the WSDL document.
  - SC_SP_MatchingHandler

    The last step of this process is to call the SC_SP_MatchingHandler like in the workflow "SerumS-SC" to perform the matching routine.

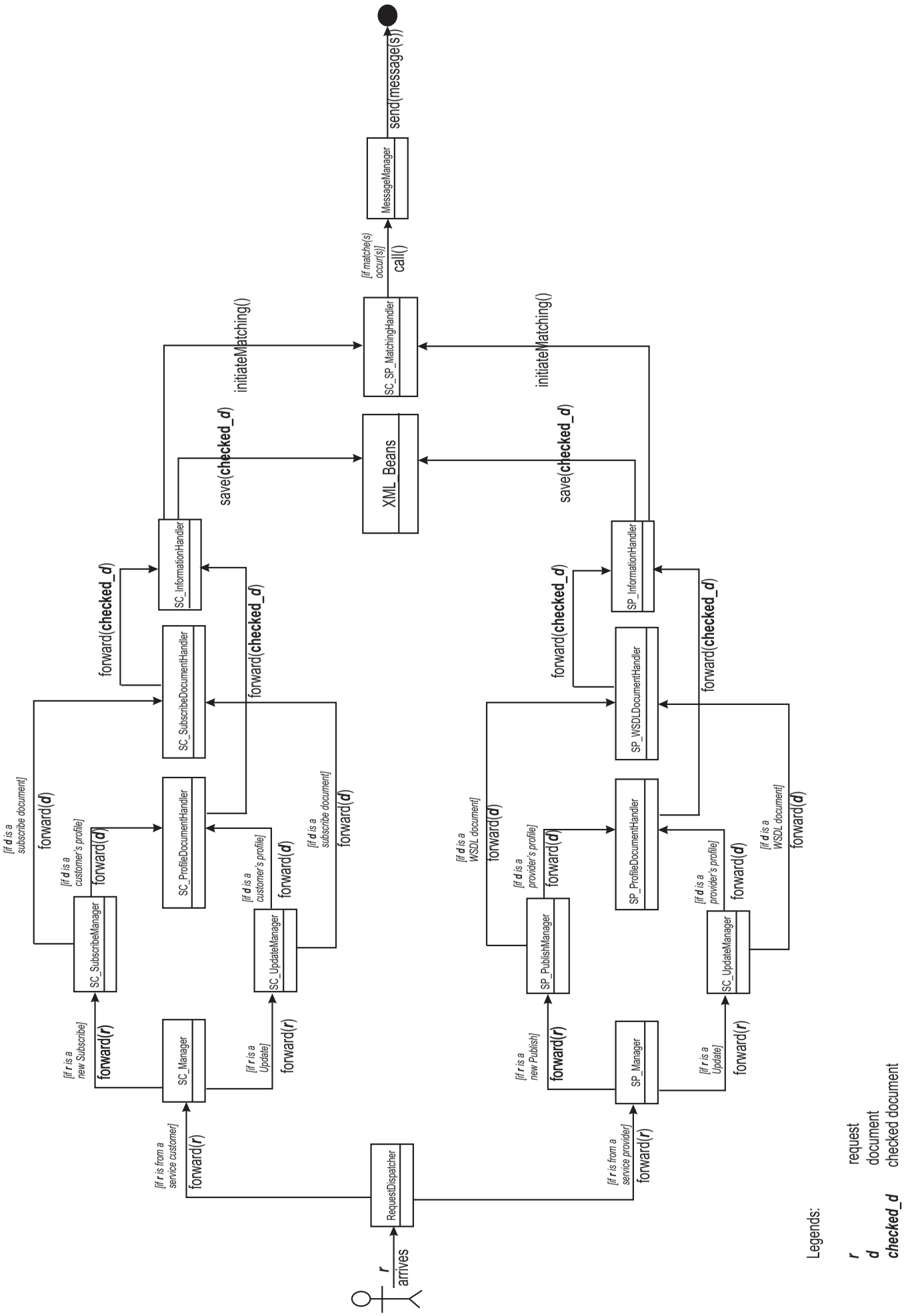

Figure 5.2: Workflow of SerumS

5.2.2.2 Autonomous task workflow

Autonomous tasks are automatically executed by SerumS due to a task plan scheduled by the system administrator. The whole autonomous task workflow consists of searching in the Internet for new Web services and checking routine at known service providers for new services or changes of existing services. Since both the search processes in Internet and checking Web services by the service providers are similar, we only describe the workflow for the automated search in the Internet for new Web services using the Google Web service based search API.
The tasks workflow are executed by the following components:

- WebServicesSearch:
  Due to the system schedule, which will be discussed at the end of this sub-chapter, the WebServicesSearch component generates a search process using the Google Web service based search API. The search API takes a search string (see chapter 5.4.1) as search parameters. The result returned by the Google API is forwarded to the SearchResultManager.

- SearchResultManager:
  It analyses the content of the previous search process and filters out the correct WSDL documents which have to be checked whether they are well-formed and valid. Other works regarding the structure of the WSDL documents also have to be performed by the SearchResultManager before sending them to the WSManager.

- WSManager:
  It is called by the SearchResultManager, which returns the well-formed WSDL documents. An important job of the WSManager is to check the validity of the WSDL documents and their content. After this step the WSManager calls the SP_InformationManager to save the found Web services. The last job of the WSManager is to call the SC_SP_MatchingHandler to perform matching between the new Web services and the service customer profiles and subscribe documents.

As mentioned earlier, the autonomous tasks workflow requires a well designed schedule of the system administrator for the automated search of new Web services in such a way, that the performance of the main jobs (handling the customer's and provider's requestes) is not affected. Generally, a schedule for the autonomous tasks workflow depends on the following aspects:

i. *When* should the autonomous tasks be performed:
   The autonomous tasks should be performed when the load on SerumS is small. This means that SerumS either itself has few tasks in the system (maintain, reorganizing of the Web service contents or checking for the correctness of their content) or there are few requests from the service customers and service providers to be handled by SerumS.

ii. *How often* should the autonomous tasks be performed:
   Actually, we are dealing with Web services which form only a part of the whole services offered by service providers on the world. Web services are generally not easy to develop and maintain so we can assume that the number of new born Web services is low. However, new Web services and changes of them have to be detected and service customers which have subscribed to SerumS have to be informed on time. SerumS only needs to perform the autonomous task one time per day, because a service user who has subscribed to SerumS for new Web services but does not get them immediately, can get them at the beginning

of his work on the next day. In the analytical model in the chapter 6.3.3.2 a calculation is made under this assumption and the result shows that performing the autonomous tasks two times per day does not effect the load of SerumS in any way.

5.3 Comparison of SerumS with a traditional UDDI registry

The previous section has already described SerumS in details. Table 5.1 compares the overall functionality of SerumS with a traditional UDDI registry and shows the advantages and disadvantages of both systems.

Problem	**How UDDI registry works**	**How SerumS works**
Proactive searching for new Web services	A UDDI registry is generally "passive". It means that the registry only waits for service providers publishing their Web services and does not perform any operation in order to get Web services for itself.	SerumS searches periodically for new Web services in Internet which are not published by their providers or not published on time to the system. Indeed, there are a lot of service providers who do not know the address of a UDDI registry or do not publish their services to any registry because they do not have time or technical know how to do that.
Notification of new Web services and changes of relevant Web services information	The service customers are not notified about changes at existing Web services	SerumS notifies service customers about every change of relevant Web services
"Ad-hoc search requests" delay the client, generate unnecessary traffic and affect the customer's network	In general, a service customer has to perform the search operation before it can use a service. "Ad-hoc search requests" delay the service and cause appropriate traffic load which can have negative affect on the customer's network, especially, if there is not any service available and the customer has to perform periodically the same search operation until it has found a service	A service customer can send SerumS a profile with search parameters. Based on this profile document SerumS will notify the customer automatically about new or changed Web services which have been published by the service providers

"Stateless requests" cause performance issue for the UDDI registry ("bottleneck")	Generally, all requests to the UDDI registry are stateless, hence the registry has to handle them immediately and the result has to be sent back to the client before the session gets time out. The registry does not keep the state of the client and it cannot store their requests to handle it at a later time point. Hence there is no cure for a number of requests sent at the same time to the registry.	Since SerumS has both the subscribe and profile document of a service customer, it has enough information to contact the customer whenever it can and especially, if the network traffic condition is favourable. With both documents stored on its site SerumS works "statefully"; this essentially differs from a traditional UDDI registry.
Correctness of the Web service information	There is too much irrelevant or invalid data in the UDDI registry which cannot be used as correct Web service information (e.g. dead links or false links for advertisement purposes)	All data sent by the service providers and customers is XML based and has a valid schema. Based on each schema SerumS checks the received data for their correctness before accepting them. A WSDL document is also checked for its validity so that many "irrelevant" or "invalid" data can be avoided at the beginning.
Protecting the use of the Web service information	Every registered user can have access to every Web service information of every service provider and can misuse them for his own purpose (e.g. using of name or logo of a well-known company for advertising for the own service products)	SerumS offers service providers the possibility to specify the users or user groups which are allowed to see the Web service information. By searching for services or matching between the service and the customer profile SerumS regards the "right-limitation" parameter specified in the provider profile and handles it in an appropriate way, so that only such service users can get the service information which are allowed by the provider.

Big effort for filtering the search result	UDDI registry searches for the Web services matching the criteria specified by the service user. The user has often to deal with a redundant set of Web services to find the appropriate ones.	SerumS combines the parameters from the subscribe and customer profile documents to determine the Web services which fulfil the user's search criteria. Based on this documents, SerumS has more parameters to constrain the search result, which is generally more precise. Thus, the customer will not spend much time for filtering out the desired Web service(s).

Table 5.1: Comparison of SerumS with a traditional UDDI registry

5.4 Technologies and tools

In this section we give an overwiev about the required software and auxiliary tools to realize SerumS. However, the most important tool in our context is the "Publish Subscribe Notification for Web services" specification[2] which forms the core component of SerumS. Hence we will discuss it with more details.

5.4.1 The Search-Engine "Google"

With the seach engine Google[2] one not only can find HTML based webpages, but also Web Service WSDL documents. To make Google find WSDL documents exclusively, it is necessary to specify the search string in a way that only WSDL documents are contained in the search result. A good strategy to do that is to specify as many keywords of a WSDL document as possible in the Google search form. Such search strings like "<wsdl:message <wsdl:portType <wsdl:operation <wsdl:input <wsdl:output" will return a good search result, where nearly all of the found documents are of type "*.wsdl" containing necessary information to access the Web services. Because of the capability of Google to discover Web services, we do not need to construct a new search functionality for SerumS. Since the Google search functionality is also accessible via a Web service based interface, it can be completely integrated into SerumS. However, the search result returned by Google has to be aggregated and prepared by SerumS for further processes.

Listing 5.1 shows the statements for searching new Web services by using the Google Web service based search interface implemented in the Java-class "WSDLFinder.java".

[2] www.google.com

```
1  private GoogleSearchResultElement[] findWSDLDocs() {
2    Vector tmpOverallResult = new Vector();
3
4      for (int i = 0; i < iteration; i++) {
5       GoogleSearch s = new GoogleSearch();
6          s.setKey(ServerConstants.GOOGLE_API_CLIENT_KEY);
7          s.setStartResult(10 * i + 1);
8          s.setMaxResults(maxResult);
9
10     try {
11           if (directive.equalsIgnoreCase("search")) {
12           s.setQueryString(directiveArg);
13       GoogleSearchResult result = s.doSearch();
14
15           GoogleSearchResultElement[] re = result.getResultElements();
16           for (int j = 0; j < re.length; j++) {
17             res += ("<a href=\"" + re[j].getURL()
18               + "\">" + re[j].getTitle() + "</a><br />");
19               if (re[j].getURL().endsWith(".wsdl")) {
20                   tmpOverallResult.addElement(re[j]);
21               }
22             }
23           }
24         }catch (GoogleSearchFault f) {
25           MessagesFactory.printErrors(this, "findWSDLDocs()", f);
26           return null;
27       }
28     ...
29     return overallResult;
30 }
```

Listing 5.1: Google - Web services based searching for new WSDL documents

The search process consists of the following steps:

- create a new instance of the GoogleSearch object (line 5)

- perform the search process (line 13)

- put found WSDL documents in a temporary vector (line 17)

- return the entire result (all found WSDL documents) back to the original application (line 29)

5.4.2 Web Service Framework "AXIS"

The "Publish Subscribe Notification for Web services" specification needs a suitable Web service environment for executing the Subscribe/Notification functionality mentioned before. Subscribe and Notification messages are transported in context of a Web service process within SOAP-Envelopes between a service customer or a service provider and SerumS. For this purpose, the Web Service Framework "AXIS"[3] is used, which offers the necessary interfaces.

[3]http://ws.apache.org

5.4.3 The "Publish Subscribe Notification for Web services" specification

Since the main task of SerumS is realizing the Publish/Subscribe mechanism, it has to propagate reliably all changes of the Web service data to service customers. The Publish/Subscribe functionality is realized by SerumS with the "Publish Subscribe Notification for Web services" specification, which is particularly designed for Web services. Currently, there is not any official Publish/Subscribe Standard adopted by any consortium like W3C or OASIS. The "Publish Subscribe Notification for Web services" is a specification recommended by a number of well-known companies like Fujitsu Laboratories of Europe, Globus, Hewlett-Packard, IBM and SAP AG. So we assume that it will be accepted as a standard in the future. Figure 5.3 shows the general process model of the Publish/Subscribe process.

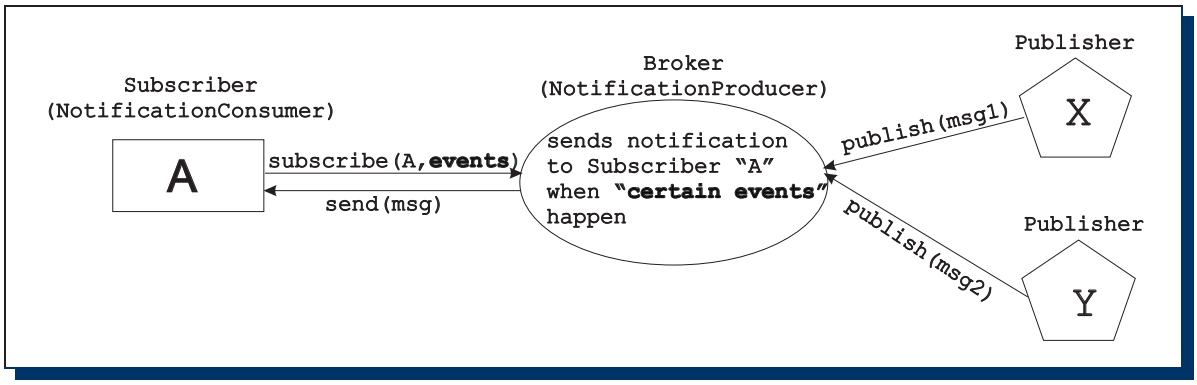

Figure 5.3: Publish/Subscribe process model

Transferred to SerumS, a "Subscriber" represents a service customer who registers itself to SerumS for certain Web services. In the case of matching between a service customer profile and a service provider and/or its Web service, the service customer is notified. In the following the structure of the Publish/Subscribe messages is described:

5.4.3.1 Subscribe Message

Generally, one differentiates between a "Subscribe Message" (in the following named as "SM") and a "Notification Message" (in the following named as "NM"). A subscriber sends a SM with a structure showed Listing 5.2 to a broker (or depending on context directly to a publisher).

```
1  <wsnt:Subscribe>
2
3    <wsnt:ConsumerReference>
4      wsa:endpointReference
5    </wsnt: ConsumerReference>
6
7    <wsnt:TopicExpression dialect = "xsd:anyURI">
8      {any}
9    </wsnt:TopicExpression>
10
11   <wsnt:UseNotify>
12     xsd:boolean
13   </wsnt:UseNotify>?
```

```
14
15    <wsnt:Precondition>
16      wsrp:QueryExpression
17    </Precondition>?
18
19    <wsnt:Selector>
20      wsrp:QueryExpression
21    </wsnt:Selector>?
22
23    <wsnt:SubscriptionPolicy>
24     {any}
25    </wsnt:SubscriptionPolicy>?
26
27    <wsnt:InitialTerminationTime>
28      xsd:dateTime
29    </wsnt:InitialTerminationTime>?
30
31  </wsnt: Subscribe>
```

Listing 5.2: Structure of the Subscribe Message

The important elements of a SM are:

- The end point address of the service customer (line 3)
- The Web service type(s) which the service customer is interested in (line 7)
- The method in which the service customer has to be notified (line 13):

  If the value is “true” then the NotificationProducer (SerumS) must notify the service customer with a NM.
- Matching rule(s) for a NM to be delivered (line 15)
- Application specific rules for a NM to be delivered (line 23)

The Publish/Subscribe specification is designed as a general purpose tool used for a large context. In the next parts we regard the explanation of the specification message elements to our context for subscribing one or more Web service types according to the “United Nations Standard Products and Services Code (UNSPSC)”[39], and we only concentrate on the message elements which are important for our work. A detailed explanation of all message elements can be found in each appropriate section of the specification[2].

The example in Listing 5.3 describes how a service customer can send a SM to SerumS in order to subscribe for a set of Web services.

```
1  <s12:Envelope
2  xmlns:s12=”http://www.w3.org/2003/05/soap−envelope”
3  xmlns:wsa=”http://schemas.xmlsoap.org/ws/2003/03/addressing”
4  xmlns:wsnt=”http://www.ibm.com/xmlns/stdwip
5   /web−services/WS−BaseNotification”
6  xmlns:npex=http://www.producer.org/RefProp
7  xmlns:provider=”http://www.serums.com /provider”>
8
9    <s12:Header>
```

```
10      <wsa:Action>
11        http://www.ibm.com/xmlns/stdwip/web-services/
12          WSBaseNotification/Subscribe
13      </wsa:Action>
14      <wsa:To s12:mustUnderstand="1">
15        http://www.producer.org/ProducerEndpoint
16      </wsa:To>
17    </s12:Header>
18    <s12:Body>
19      <wsnt:Subscribe>
20
21        <wsnt:ConsumerReference
22          xmlns:ncex="http://www.consumer.org/RefProp">
23          <wsa:Address>
24            http://www.serums.com/notificationReceicer
25          </wsa:Address>
26          <wsa:ReferenceProperties>
27            <ncex:NCResourceId>
28            </ncex:NCResourceId>
29          </wsa:ReferenceProperties>
30        </wsnt:ConsumerReference>
31
32        <wsnt:TopicExpression dialect="http://www.ibm.com/xmlns/
33          stdwip/web-services/WSTopics/TopicExpression/simple">
34          provider:unspsc=0000000011,000000018
35        </wsnt:TopicExpression>
36
37        <wsnt:UseNotify>true</wsnt:UseNotify>
38
39        <wsnt:Precondition
40            dialect="http://www.w3.org/TR/1999/REC-xpath-19991116">
41              boolean(provider:continent = "Europe" or
42                provider:continent="America" or
43              provider:continent="Europe" and
44              provider:bindingType="soap")
45         </wsnt:Precondition>
46
47         <wsnt:Selector
48           dialect="http://www.w3.org/TR/1999/REC-xpath-19991116">
49             boolean(ncex:Producer="15")
50         </wsnt:Selector>
51
52         <wsnt:SubscriptionPolicy>
53            The Web services matched to this Subscribe Message
54            may not older then 2 months since its publishing
55         </wsnt:SubscriptionPolicy>
56
57        <wsnt:InitialTerminationTime>
58          2006-12-25T00:00:00.00000Z
59        </wsnt:InitialTerminationTime>
60
61      </wsnt: Subscribe>
62    </s12:Body>
63  </s12:Envelope>
```

Listing 5.3: Example Subscribe Message

In this example we assume that an XML based schema of a provider profile is placed at the address “http://www.serums.com/provider” and the elements “unspsc”, “continent” and “bind-

ingType" are contained in this schema and can be used within a SM. Furthermore, we assume that SerumS is accessible through the URL specified at line 7.

The important content of a SM are:

- The service customer specifies an address needed by the NotificationProducer to send back future messages (line 23) (e.g. SubscribeResponse Message or NM).

- The service customer specifies the Web service type(s) (line 32) which it is interested in. The example uses the UNSPSC-based service types "0000000011" and "0000000018" standing for "Maintenance or repair" and "Oursource" Web services.

- The "<UseNotify>" element (line 37) has the value "true" and indicates that the NotificationProducer must use a NM to deliver the message to the service customer using the address given in the "<Address> element of the service customer profile.

- The values provider:continent="America" or provider:continent="Europe" and provider:bindingType="soap" in the "<Precondition> element (line 39) indicate that the NotificationProducer only delivers a message to the service customer if the matched service provider is from the continent America or Europe and it must support "soap" as "bindingType" (message transfer protocol).

- The content within the "<SubscriptionPolice>" element (line 52) in this example is designed for human interpretation and it indicates that the service customer is only interested for such Web services which are not older than two months since their publishing.

As response to a SM the Subscriber obtains a SubscribeResponse Message as described in Listing 5.4.

```
1  ...
2  <wsnt:SubscribeResponse>
3    <wsnt:SubscriptionReference>
4
5      <wsa:Address>
6        Address of Subscription Manager
7      </wsa:Address>
8
9      <wsa:ReferenceProperties>
10       Subscription Identifier
11     </wsa:ReferenceProperties>
12     ...
13   </wsnt:SubscriptionReference>
14   ...
15 </wsnt:SubscribeResponse>
16 ...
```

Listing 5.4: Structure of the SubscribeResponse Message

- The "<Address>" element (line 5) contains the address of the Subscribe Manager (in our case a part of SerumS), which can differ from the original address of the Server. The Subscriber has to use this address to perform future operations regarding the Subscribe process, e.g. to cancel a subscription or to perform the "GetCurrentMessage" operation (see section 4.3 of [2]).

- The Subscriber gets a unique Subscription Identifier key created by the Subscription Manager (line 9). The Subscriber uses this Subscription key to be identified for the future operations to the Subscribe Manager.

```
1  <s12:Envelope
2  xmlns:s12="http://www.w3.org/2003/05/soap-envelope"
3  xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
4  xmlns:wsnt=
5  "http://www.ibm.com/xmlns/stdwip/web-services/WS-BaseNotification"
6  xmlns:ncex=http://www.consumer.org/RefProp
7  xmlns:provider="http://www.tum-serums.org/provider">
8
9    <s12:Header>
10     <wsa:Action>
11       http://www.ibm.com/xmlns/stdwip/web-services/WSBaseNotification/
12         SubscribeResponse
13     </wsa:Action>
14     <wsa:To s12:mustUnderstand="1">
15       http://www.consumer.org/ConsumerEndpoint
16     </wsa:To>
17   </s12:Header>
18
19   <s12:Body>
20     <wsnt:SubscribeResponse>
21       <wsnt:SubscriptionReference
22           xmlns:npex="http://www.producer.org/RefProp">
23         <wsa:Address>
24           http://www.tum-serums.org/subscribe
25         </wsa:Address>
26         <wsa:ReferenceProperties>
27           <npex:NSResourceId>
28             uuid:8fefcd11-7d3d-66b344a2-ca44-9876bacd44e9
29           </npex:NSResourceId>
30         </wsa:ReferenceProperties>
31       </wsnt:SubscriptionReference>
32     </wsnt:SubscribeResponse>
33   </s12:Body>
34
35 </s12:Envelope>
```

Listing 5.5: Example SubscribeResponse Message

Listing 5.5 shows an example of a SubscribeResponse Message.

- We assume that the Subscribe Manager of SerumS is accessible via the URL “http://www.tum-serums.org/subscribe” (line 24)

- SerumS creates a Universally Unique Identifier (UUID)[43] key for every Subscribe operation and sends it back to the Subscriber (line 28). For future operations the Subscriber has to specify this UUDI-key in every message to identify itself to SerumS.

5.4.3.2 Notification Message

Analogue to a Subscribe Message, a Notify Message (briefly called “NM”) may be sent by a NotificationProducer to a NotificationConsumer if the rules specified in the Subscribe Message are matched. The structure of the NM is illustrated in Listing 5.6.

```
1  ...
2
3  <wsnt:Notify>
4    <wsnt:NotificationMessage>
5      <wsnt:Topic dialect=”xsd:anyURI”>
6        {any}
7      </wsnt:Topic>
8
9      <wsnt:ProducerReference>?
10       wsa:EndpointReference
11     </wsnt:ProducerReference>
12
13     <wsnt:Message>
14       xsd:any
15     </wsnt:Message>
16   <wsnt:NotificationMessage>+
17 </wsnt:Notify>
18
19 ..
```

Listing 5.6: Structure of the Notify Message

The main elements of a NM are:

- The topic matched to the SM (line 5) (in our case the Web service type, which the service customer is interested in)

- The address of the message producer (line 9, to denounce where the message is from)

- The content of the message (line 13), which can be unstructured or XML based.

Regarding SerumS, in the case of matching of the service customer´s subscribed interest with a Web service, a NM is sent back. An example NM is shown in Listing 5.7.

```
1  <s12:Envelope
2  xmlns:s12=”http://www.w3.org/2003/05/soap−envelope”
3  xmlns:wsa=”http://schemas.xmlsoap.org/ws/2003/03/addressing”
4  xmlns:wsnt=
5   ”http://www.ibm.com/xmlns/stdwip/web−services/WS−BaseNotification”
6  xmlns:ncex=”http://www.consumer.org/RefProp”
7  xmlns:npex=http://www.producer.org/RefProp
8  xmlns:provider=”http://www.tum−serums.org/provider”>
9
10   <s12:Header>
11     <wsa:Action>
12       http://www.ibm.com/xmlns/stdwip/web−services/
13         WS−BaseNotification/Notify
14     </wsa:Action>
15     <wsa:To s12:mustUnderstand=”1”>
```

```
16        http://www.consumer.org/ConsumerEndpoint
17      </wsa:To>
18
19      <ncex:NCResourceId>
20        uuid: 8fefcd11−7d3d−66b344a2−ca44−9876bacd44e9
21      </ncex:NCResourceId>
22    </s12:Header>
23
24    <s12:Body>
25      <wsnt:Notify>
26        <wsnt:NotificationMessage>
27
28          <wsnt:Topic
29            dialect=”http://www.ibm.com/xmlns/stdwip/
30              web−services/WSTopics/TopicExpression/simple”>
31            provider:nspsc:0000000011
32          </wsnt:Topic>
33
34          <wsnt:ProducerReference
35            xmlns:npex=” http://www.tum−serums.org/notify”>
36            <wsa:Address>
37              http://www.tum−serums.org/notify
38            </wsa:Address>
39            <wsa:ReferenceProperties>
40              <npex:NPResourceId>
41                uuid:84decd55−7d3f−65ad−ac44−675d9fce5d22
42              </npex:NPResourceId>
43            </wsa:ReferenceProperties>
44          </wsnt:ProducerReference>
45
46          <wsnt:Message>
47            <npex:NotifyContent>
48              <wsdl:definition>
49                 ‘‘content of the WSDL definition’’
50              </wsdl:definition>
51            </npex:NotifyContent>
52          </wsnt:Message>
53
54        <wsnt:NotificationMessage>
55      </wsnt:Notify>
56    </s12:Body>
57  </s12:Envelope>
```

Listing 5.7: Example Notify Message

The example NM has the following content:

- Th Web service type(s) (line 28) matched to the service customer’s SM. In our case the Web service type is shown as an UNSPSC code.

- The Web service address of the notify-interface of SerumS (line 37), from where the message is delivered.

- A WSDL document of the matched service provider is sent as message content to the service customer (line 46).

5.5 Software implementation - Proof of Concept (PoC)

This section describes the technical software aspects of SerumS and gives details about its implementation.

The PoC is developed with Eclipse 3.0[4], an open source graphical-oriented software development environment. As programming language Java 5.1[5] has been chosen for the implementation, because SerumS has to be extensible for future functionality and many frameworks for developing Web service applications are written in Java.

5.5.1 Software modularization and coding

According to the conceptional architecture of SerumS (see 5.2) the whole source code is structured in the following Java-based packages:

- org.oasis-open.docs.wsrf.rw-2.Base-Notification.*

  This source code of this package is generated from the Subscribe/Notification Specification WSDL document "WS-BaseN.wsdl"[6] originated by OASIS using the WSDL2Java tool from the AXIS framework. The "WS-BaseN.wsdl" document contains the interface definitions for realizing the Subscribe/Notification operations. The generated classes are used by SerumS components by extending (in Java "extends") and/or implementing (in Java "implements") them. Basically, the following classes are generated from the WSDL document:

  - Subscribe-components:
    - SubscribeManager.java
    - Subscribe.java
    - SubscribeResponse.java
    - ResumeSubscribe.java
    - PauseSubscribeResponse
    - ResumeSubscribeResponse.java
    - GetCurrentMessage.java
    - GetcurrentMessageResponse.java
  - Notification-components:
    - NoticationProducerRP.java
    - Notify.java

- serums.server.*

  The serums.server.* package contains all necessary components to realize the "Search and management of Web services and profiles of the service customers" functionality (see 5.2). The package in turn contains the following sub-packages:

[4] http://www.eclipse.org
[5] http://www.sun.com
[6] http://docs.oasis-open.org/wsrf/rw-2.wsdl

  - serums.server.management.*

    The package provides components for managing found Web services and notifying the service customers about new Web services or changes of existing services.

  - serums.server.search.*

    The package provides components for searching Web services in the Internet and preparing them for using within the system.

  - serums.server.util

    The package provides additional components for handling WSDL documents.

- serums.provider

  This package contains the components to realize the "Interfaces to the providers" and contains the following sub-packages:

  - serums.provider.operations.*

    The package provides components to handle provider's operations like registering, publishing Web services and notifying Web service changes.

  - serums.provider.util.*

    the package provides some auxiliary components for performing the provider's operations

- serums.customer.*

  Analogue to the serums.provider.* package, the serums.customer.* package provides components to realize the "Interfaces to the service customers" (SerumS-SC tasks). The whole package has the same structure as the serums.provider.* package.

  In addition to the mentioned packages, the following are used for completing the functionality of SerumS:

- serums.client.*

  The package contains GUI based applications for testing the SerumS server functionality and consists of following sub-packages:

  - serums.client.gui.customer.*

    The package contains GUI based components for testing the customer's interfaces, e.g. handling customer's subscribe or registering operation.

  - serums.client.gui.provider.*

    The package contains GUI based applications for testing the provider's interfaces, e.g. handling publish, subscribe or service change notify operation.

  - serums.client.gui.util.*

    The package consists of auxiliary components for the other serums.gui.customer.* and serums.gui.provider.* components.

- serums.common.*

  The package contains all necessary components used overall within the project, e.g. the class XMLDataFactory.java to build XML based messages to send to or to be received by SerumS.

- serums.handler.*

  The package consist of components for handling (operation) requests as SOAP messages sent from a client or provider to SerumS. Two main classes are CustomerOperationsHandler and ProviderOperationsHandler, which receive, analyze a customer or provider SOAP-based operation request and forward it to the appropriate component of SerumS.

- serums.xmltemplates.*

  As mentioned earlier, XMLBeans is used instead of a DBMS for handling the XML based customer or provider profiles. For this purpose, a schema has been defined, from which XMLBeans generates the appropriate Java-class to handle an instance of the schema. The next part "Code example" demonstrates how XMLBeans actually works.

Figure 5.4 depicts the described packages structure and Figure 5.5 shows the GUI interface of the executable application ProviderMainApplication.class from the package "serums.client.gui.provider.*". This application can be used by a service provider to publish a WSDL document to SerumS.

5.5.2 Code examples

In this sub-section some code examples are illustrated to show how service customer and service provider profiles are realized and handled with XMLBeans. In order to get into work with XMLBeans, the following preparing works are necessary:

a) Defining a valid XML schema (in our case a valid XML schema for a customer and provider profile)

   Appendixes E and F show the compact form of the XML schemas of the customer and provider profile.

   Listing 5.8 shows an example of a customer profile according to the customer profile XML schema.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <wstop:Customer xmlns:wstop="customer.profiles.templates">
3
4      <wstop:Contact>
5          <wstop:Description>Customer Profile XML</wstop:Description>
6          <wstop:Phone>017829834579</wstop:Phone>
7          <wstop:Email>cao@in.tum.de</wstop:Email>
8          <wstop:AddressLine>Strassename 3</wstop:AddressLine>
9          <wstop:AddressLine>87659 Garching </wstop:AddressLine>
10         <wstop:AddressLine>Germany</wstop:AddressLine>
11     </wstop:Contact>
12
13     <wstop:Ex-Providers>
14         <wstop:MustHaveContact>true</wstop:MustHaveContact>
15         <wstop:Ex-Providers>
16             <wstop:Continent>Asia</wstop:Continent>
17             <wstop:Continent>Australia</wstop:Continent>
18             <wstop:Country>Argentinia</wstop:Country>
19             <wstop:Country>USA</wstop:Country>
```

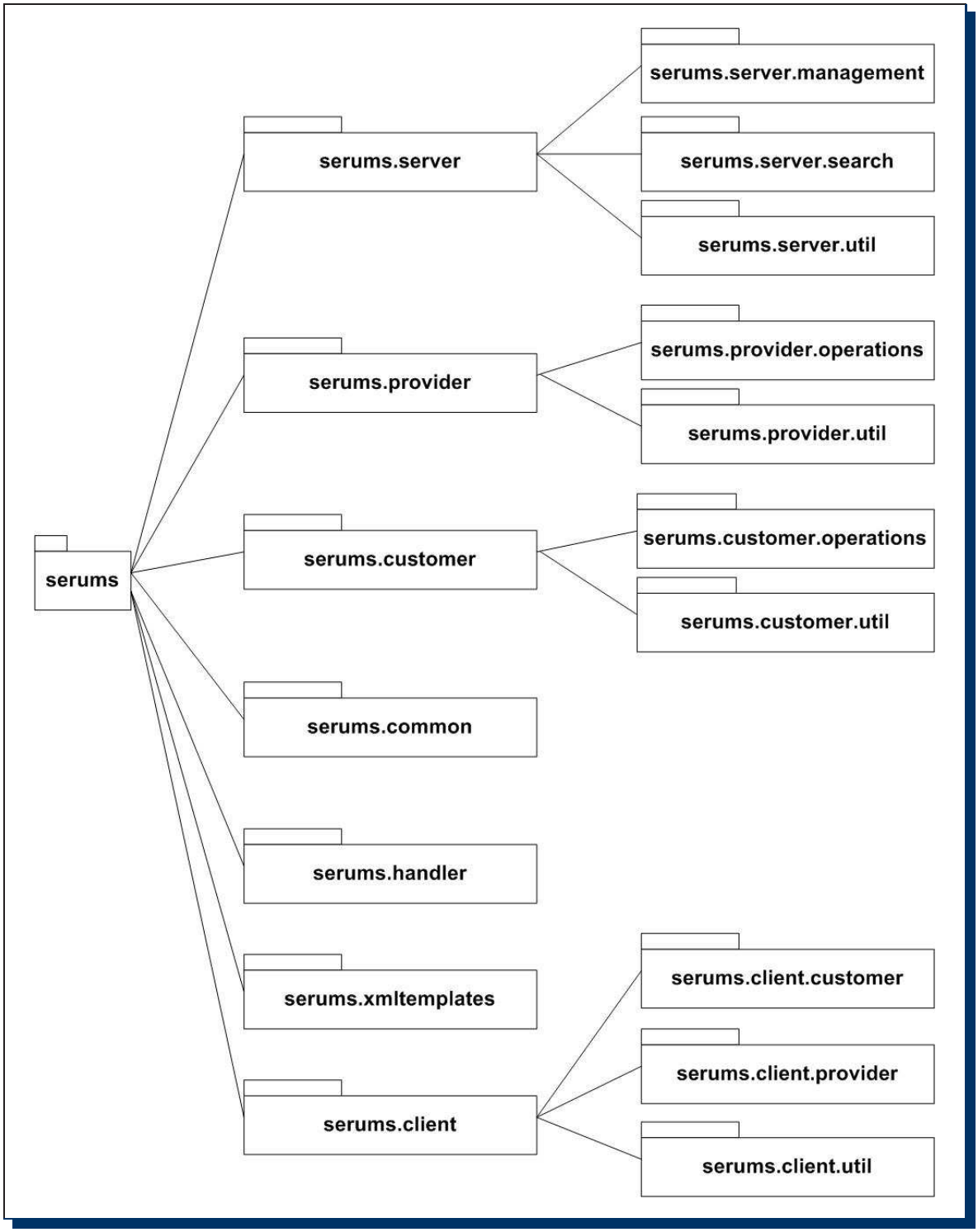

Figure 5.4: SerumS package structure

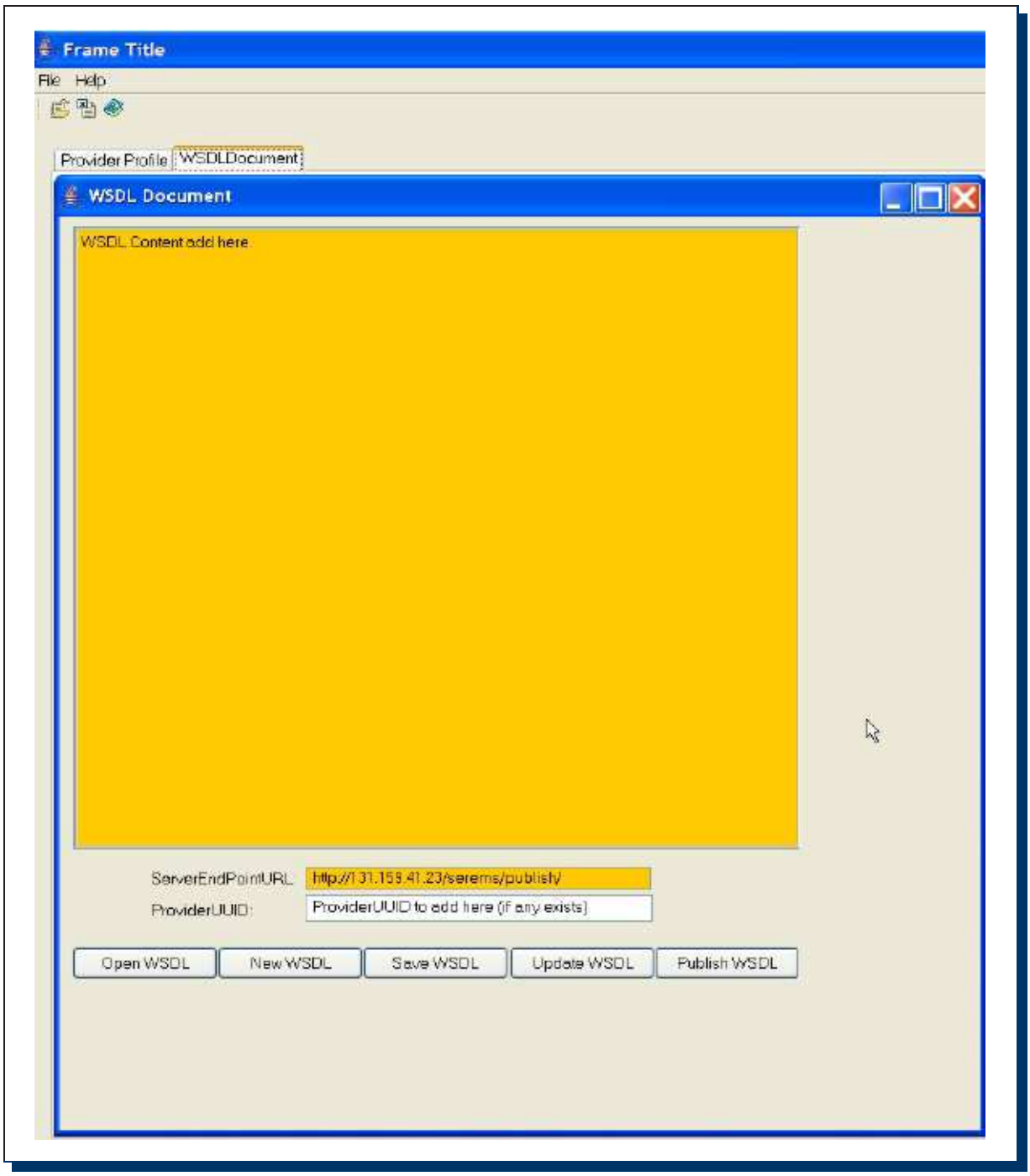

Figure 5.5: The GUI interface for publishing or changing a WSDL document to SerumS

```
20            </wstop:Ex−Providers>
21            <wstop:MustHaveContact>true</wstop:MustHaveContact>
22        </wstop:Ex−Providers>
23
24        <wstop:Service>
25            <wstop:Binding−type>ALL</wstop:Binding−type>
26            <wstop:CategoryBag>t1</wstop:CategoryBag>
27        </wstop:Service>
28
29    </wstop:Customer>
```

Listing 5.8: Example service customer profile

b) Compiling the XML schema into the appropriate Java-class

From the schema XMLBeans generates the appropriate Java-based package hierarchy based on the "targetNamespace", in our case for the provider profile: "templates.profiles.provider.*". This package contains all necessary Java-classes generated by XMLBeans for handling a provider profile document.

c) Handling an XML instance (in our case a well-formed and valid customer or provider profile with real data)

Listing 5.9 shows how a customer profile (as XML document) is converted into an XMLBeans based instance so that it can be used as a Java object with the contents of the customer profile as Java object properties.

```
1
2   ...
3   public static XmlObject getXMLData(String schemaName, File file) {
4                       String schema = new String(schemaName);
5                       XmlOptions opts = getDefaultXMLOptions();
6
7     CustomerDocument customerDoc = CustomerDocument.Factory.
8       parse(new File(XMLDataFactory.TEST_SCHEMA_DIRECTORY
9         + "/" + "customer−profile.xml"),  opts);
10
11    return customerDoc;
12  }
13  ...
```

Listing 5.9: Create an XMLBeans instance from an XML file

The CustomerDocument object is created (line 7) using XMLBeans, which converts an XML based customer profile (located at the file path "XMLDataFactory.TEST_SCHEMA_DIRECTORY + "/" + "customer-profile.xml") in a Java object.

After the XML based customer profile is converted into a Java object, its properties can be accessed with the Java programmer language statements as in Listing 5.10.

```
1   ...
2   CustomerDocument.Customer customer = customerDoc.getCustomer();
3   ContactDocument.Contact contact = customer.getContact();
4   String customerID = contact.getCustomerUUID();
5   String customerDes = contact.getDescription();
```

```
6  String customerStreet = contact.getStreet();
7  ...
```

Listing 5.10: Access the content of the customer profile as Java object properties

In the above example, the contact information of the service customer can be found out with the "get()" methods of the appropriate Java class generated by XMLBeans.

5.5.3 Technical characterization of SerumS

In this sub-chapter the technical specification of Serums and the appropriate equiqments for its operation are described.

- **Lines of Code (LOC):**
  The whole project contains about 75.000 LOC

- **Hardware equipment:**
  CPU: AMD Athlon(TM) XP 2600+ 1.91 GH

  RAM: 1GB

  SerumS requires 55MB RAM for its operation at the beginning and about 140MB during the validation

- **Software equipment:**
  Operating System: SUSE LINUX (v.90) (General Public License (GPL))

  Application Server: Tomcat (v.4.1) (GPL)

  Web service execution environment: AXIS (v.1.2) (GPL)

  Publish/Subscribe Tools: Publish-Subscribe Notification for Web services of OASIS

  Tool for automatic search: Google Web service search API (BSD 2.0 license[7])

- **Validation:**
  SerumS is tested about 24 hours on the mentioned hardware equipment and with randomized requests sent from a test client. SerumS is located on a machine with the IP address 131.159.41.23 within "Das Münchner Hochschulnetz"[8]. All important functionality ("subscribe", "publish", "automatic search for Web services in the Internet", "checking for new Web services and for changes of existing Web services at service providers") has an execution guaranty with at most 10% of the whole testing time. There was no noticeable failure or problem during the validation.

- **Documentation:**
  Documenation is in preparation.

[7] http://de.wikipedia.org/wiki/BSD-Lizenz
[8] http:www.lrz-muenchen.dewirintrode#mwn

5.6 Résumé

This chapter described the architecture of SerumS and presented the tools and technologies, which are necessary for realizing the PoC (SerumS) with the features mentioned in the section 4.6. Via the examples with customer, provider profiles and Subscribe/Notify messages, the concept of “XMLBeans” and the “Publish Subscribe Notification for Web services” specification have been illustrated. The next chapter deals with the evaluation of the implementation by measuring and comparing the performance and effiency of SerumS against a traditional UDDI registry, and demonstrates the robustness of the implementation under load.

Chapter 6

Validation of SerumS

6.1 General Process

The purpose of the validation is to demonstrate the performance of SerumS compared to a UDDI registry based on the traditional UDDI specification. The first part of the validating process is the "Elementary Operations Measurement (EOM)" which evaluates the elementary operations "inquiry()" and "publish()" of the UDDI specification performed by both service entities (service customers and service providers) to a test UDDI registry and SerumS. The time behavior of the "register()" operation, which does not belong to the UDDI specification but which is often needed by the service entities in order to use the registry's functionality, will also be measured. Furthermore, the correctness and the completeness of the search result of SerumS by the "inquiry" operation is also measured and compared to the UDDI registry. The second part of the validation process "Quantitative Modelling (QAM)" models the stationary flow of the task cycle of the service entities. QAM shows the effective load capacity of SerumS compared to a UDDI registry depending on the number of the service entities, the number of Web services and the frequency in which providers publish new services or change existing services and customers search for services. In the last part of the validation process SerumS's functionality is tested under different load configurations. This so-called "Load Capacity Measurement (LCM)" uses the web application test software "NeoLoad"[1] as a load generator to send multiple inquiry requests as a stress test to SerumS. Within the validation, the response time of SerumS for a request while handling a number of virtual users serves as a measure. The response time is the turn-around time from sending the inquiry request until receiving the result returned by the UDDI registry or SerumS. The client and the server machines for the validation process lie in the different network segments, but within the Munich Academic Network. Because the "ping" response time from a test client to the test server is less then 1 ms, the transport time between a client and the server is ignored in the analysis of the result.

The validation steps (EOM, QAM and LCM) have been performed sequentially, and the measurements have been first executed on the test UDDI registry and then on SerumS. However, both EOM and LCM use an environment of their own with an appropriate software and hardware configuration.

[1] http://www.neotys.com/load-testing-tool/neoload.html

6.2 Elementary Operations Measurement (EOM)

6.2.0.1 Purpose of the validation

The following aspects of SerumS are concerned within the EOM:

- *Time for the "register()" operation*:
  The time needed by a service customer or service provider for registering at SerumS
- *Time for the "publish()" operation:*
  The time needed by a service provider for publishing a Web service to the UDDI registry or SerumS
- *Time for the "inquiry()" operation:*
  The time needed by a service customer for searching a certain Web service on the UDDI registry or SerumS
- *"Correctness" of the search result:*
  How correct is the search result returned by SerumS compared to a UDDI registry with different search criteria so that a service customer does not get a non-existing or irrelevant service
- *"Completeness" of the search result:*
  How complete is the search result returned by SerumS compared to a UDDI registry so that a service customer does not miss any existing service matching to its search criteria

6.2.0.2 Validation environment

6.2.0.2.1 Hardware and Software configuration: The communication between the service client, service provider and service broker is in the context of a distributed application, hence in order to obtain a realistic test result the clients and the server are placed on different networks and connected by the Internet. The server and the network are physically configured as followed:

- Server

  CPU: AMD Athlon(TM) XP 2600+ 1.91 GHz, RAM: 1GB
- Network

  Network: Das Münchner Hochschulnetz[2], Network capacity: 2,5Gb/s[3]

Figure 6.1 depicts the hardware and software configuration for the EOM:

- Software at the customer and provider side:

[2]http://www.lrz-muenchen.de/wir/intro/de/#mwn
[3]http://www.lrz-muenchen.de/wir/intro/de/#uni

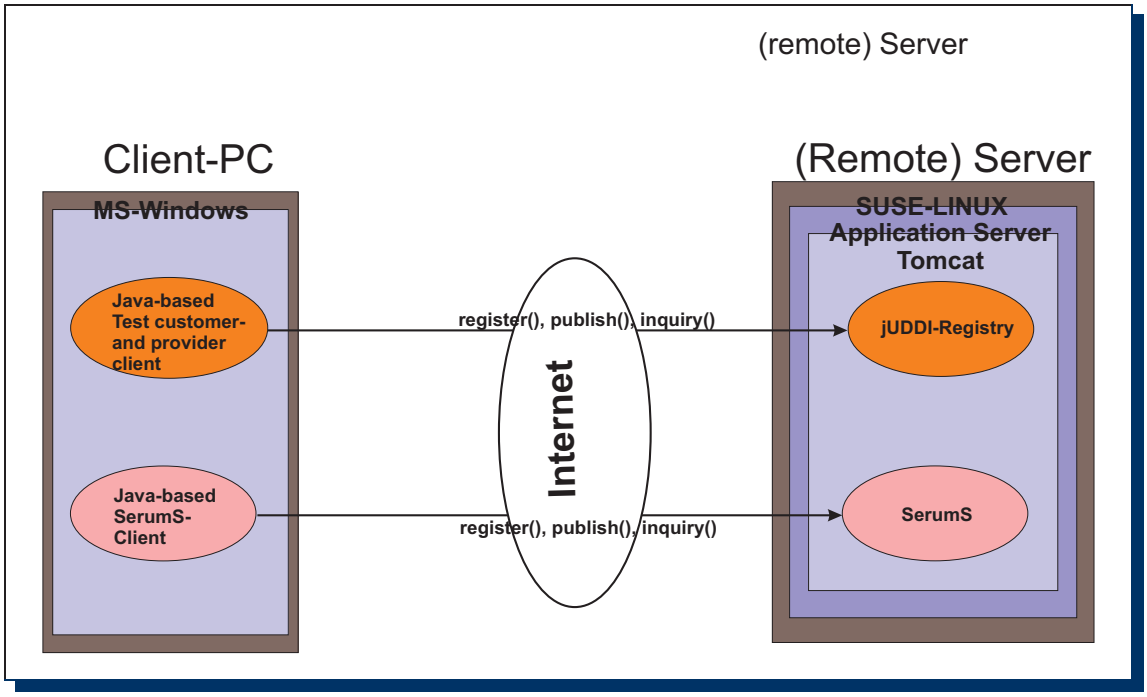

Figure 6.1: EOM Environment

For measuring the efficiency of the UDDI registry a test application is programmed, which sends the "publish(), inquiry()" requests to the registry. The test application for testing SerumS is a part of the implementation of the PoC (see chapter 5.5). Beside executing the standard elementary operations, the application can also be used to register a service customer or service provider to SerumS in the initial process.

- Software on the broker side:

  On the broker side SUSE-Linux (v. 9.0) serves as operating system with the following applications:

  – Tomcat[4] as Application Server
  – jUDDI[5] as UDDI registry, a free open source implementation of the UDDI specification
  – MySQL version 5 serves as DBMS for managing the UDDI content

- SerumS

  XMLBeans[6] serves instead of a DBMS for managing the XML based service customer and service provider profiles and also the WSDL documents. XMLBeans is a Java-based technology for handling XML based documents by binding it into Java based objects (see chapter 5.5.1). Hence one can easily convert the XML based documents into Java based objects or vice versa (Object Serialization/Deserialization) and treat them as objects on

[4]http://tomcat.apache.org
[5]http://ws.apache.org/juddi
[6]http://xmlbeans.apache.org

a file system. Using XMLBeans, a DBMS and some cumbersome SQL inquiries can be avoided.

6.2.0.2.2 Validation data: In order to get a representative validation result we need service customers and providers and have to modify their profiles in different ways.

i. Service customer profiles

A service customer can send an profile to SerumS, in general, after it has registered itself with a Subscribe Profile. A service customer profile is *optional but not mandatory*, because it only provides *additional* information about the customer, which can be used as parameters for a more accurate search. Listing 6.1 and 6.2 show examples of two different customer profiles.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <cus:Customer
3  xmlns:cus="http://www.tum-serums.org/customer">
4
5    <cus:Contact>
6      <cus:CustomerUUID>
7        2ceacef8-df16-40b2-86c3-90111dfd2e86
8      </cus:CustomerUUID>
9      <cus:Description>Customer Profile XML</cus:Description>
10     <cus:Phone>017829834579</cus:Phone>
11     <cus:Email>cao@in.tum.de</cus:Email>
12     <cus:Street>Strassename 3</cus:Street>
13     <cus:PostalCode>8...</cus:Postal
14     <cus:City>Garching</cus:City>
15     <cus:Country>Germany</cus:Country>
16   </cus:Contact>
17
18   <cus:Policy>
19     <cus:MustHaveContact>true</cus:MustHaveContact>
20     <cus:ValidedByServer>false</cus:ValidedByServer>
21     <cus:Ex-Providers>
22       <cus:Ex-Continents>Asia, Australia</cus:Ex-Continents>
23       <cus:Ex-Countries>USA, Argentinien</cus:Ex-Countries>
24     </cus:Ex-Providers>
25   </cus:Policy>
26
27   <cus:Service>
28     <cus:Binding-Types>ALL</cus:Binding-Types>
29     <cus:CategoryBags>10001000</cus:CategoryBags>
30   </cus:Service>
31
32 </cus:Customer>
```

Listing 6.1: Example customer profile 1

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <cus:Customer
3  xmlns:cus="http://www.tum-serums.org/customer">
4
5    <cus:Contact>
6      <cus:CustomerUUID>
```

```
7          2ceacef8−df16−40b2−86c3−90111dfd2e86
8        </cus:CustomerUUID>
9        <cus:contactPerson>Mayer Patric<cus:contactPerson>
10       <cus:Description>Customer Profile XML</cus:Description>
11       <cus:Phone>+4908912345678</cus:Phone>
12       <cus:Email>mayer.Partic@yahoo.de</cus:Email>
13       <cus:Street>Example street 3</cus: PostalCode>
14       <cus:PostalCode>8...</cus:Postal
15       <cus:City>Munich</cus:City>
16       <cus:Country>Germany</cus:Country>
17       <cus:homepage>http://www.mayer−patric.de<cus:homepage>
18       <cus:publicFingerPrint>
19         78ff9889ad0092q80
20       </cus:publicFingerPrint>
21       <cus:CustomerGroup>ff450ead12345678</cus:CustomerGroup>
22     </cus:Contact>
23
24     <cus:Policy>
25      <cus:MustHaveContact>true</cus:MustHaveContact>
26      <cus:ValidedByServer>false</cus:ValidedByServer>
27      <cus:Communication>secure</cus:Communication>
28     </cus:Policy>
29
30     <cus:Service>
31       <cus:ServiceExpiryTime>2006−12−30</cus:ServiceExpiryTime>
32       <cus:Binding−Types>rpc</cus:Binding−Types>
33       <cus:CategoryBags>100000010</cus:CategoryBags>
34     </cus:Service>
35
36   </cus:Customer>
```

Listing 6.2: Example customer profile 2

In contrast to the first customer profile the second one contains more information about the customer:

- The customer specifies the address (line 17) of his home page and a public fingerprint which could be used for an encrypted communication process. In addition to the service, a "CustomerGroup" identifier can be specified to indicate that the service customer belongs to this CustomerGroup which is needed to search for Web services with certain user access limitation, e.g. if a service provider allows only a certain user or user group to see or use its service.

- If the "<Communication>" element (line 27) is specified with the value "secure" then SerumS must use a secure protocol to communicate with the service client, e.g. https.

- The customer is only interested in services which are valid at least until the specified date (line 31).

For the validating process the following spectrum is available:

- Up to 300 service customers with different profiles

- Different locations

- Different interests (Web service types)

- Different base and technical requirements to the service providers, e.g input, output or conditional parameters.

ii. Service provider profile

Compared to a customer profile a provider profile is mandatory for SerumS in order to identify the service provider itself. Before a service provider can publish its service, it has to send a profile to SerumS. Unlike the content within a "<BusinessEntity>" element which represents a Web service entry of a service provider in a UDDI registry, a provider profile contains much more elements than the default information about the Web service and the service provider itself. As example from the provider profile, one can determine whether any input data during the execution of Web service application process is needed by the service provider, e.g. credit card number or a minimum amount of an order etc. Other important data are technical or conditional information which can be used for specifying the service more exactly. Such information can also be used by SerumS to search for Web services to support the matching process with customer profiles. Listings 6.3 and 6.4 show two examples of provider profiles.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <prov:Provider xmlns:prov="http://www.tum-serums.org/provider">
3
4    <prov:Contact>
5      <prov:Description>Siemens AG</prov:Description>
6      <prov:Phone>+498...</prov:Phone>
7      <prov:Email>support@siemens.de</prov:Email>
8      <prov:Street>Hoffmannstr. 51</prov:Street>
9      <cus:PostalCode>81359</cus:Postal
10     <prov:City>Munich</prov:City>
11     <prov:Country>Germany</prov:Country>
12   </prov:Contact>
13
14   <prov:Business>
15     <prov:Name>Siemens AG</prov:Name>
16     <prov:Description>
17       Provider Business Description XML
18     </prov:Description>
19     <prov:UNSPSCCategoryBag>10000011</prov:UNSPSCCategoryBag>
20   </prov:Business>
21
22   <prov:Service>
23     <prov:ServiceExpiryTime>2006-01-01</ServiceExpiryTime>
24   </prov:Service>
25
26 </prov:Provider>
```

Listing 6.3: Example provider profile 1

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <prov:Provider xmlns:prov="provider.profiles.templates">
3
4    <prov:Contact>
5      <prov:Description>Samsung - South Korea</prov:Description>
6      <prov:Phone>012345678</prov:Phone>
7      <prov:Email>support@samsung.com</prov:Email>
8      <prov:Street>examples treet</prov:Street>
9      <prov:City>example city</prov:City>
10     <prov:Country>South Korea</prov:Country>
11     <prov:Continent>Asia</prov:Continent>
12   </prov:Contact>
```

```
13
14    <prov:Business>
15      <prov:Name>Samsung, Ltd.</prov:Name>
16      <prov:Description>
17        Provider Business Description XML
18      </prov:Description>
19      <prov:UNSPSCCategoryBag>10000000</prov:UNSPSCCategoryBag>
20    </prov:Business>
21
22    <prov:Service>
23      <prov:ServiceExpiryTime>2006-09-01</ServiceExpiryTime>
24      <prov:ServiceLimited> (1)
25        <prov:ServiceLimitedID>
26          ff450ead12345678
27        </prov:ServiceLimitedID>
28      </prov:ServiceLimited>
29      <prov:InputData>
30        <prov:InputDataName>customerID</prov:InputDataName> (2)
31        <prov:InputDataType>stringType</prov:InputDataType>
32      </prov:InputData>
33    </prov:Service>
34
35  </prov:Provider>
```

Listing 6.4: Example provider profile 2

- The service provider specifies a "<ServiceLimitedID>" value (line 24) to inform SerumS about users who are allowed to explore the Web service content. In this case, the identifier matches the values specified in the profile from Listing 6.4 and allows this user to explore the service content.

- The service provider requires a "customerID" from the service customer as "<InputParameter>" (line 29) in order to identify the service customer and to start the Web service process.

Analogue to the customer profiles, the following configuration data for the service providers are available for the validation:

- up to 300 service providers with

- Different contact information

- Different locations

- Different service types

- Different service technical information

6.2.1 Validation procedure and result

6.2.1.1 Validation procedure

As mentioned earlier, the elementary operations "register(), publish(), inquiry()" are executed sequentially on the UDDI registry and then on SerumS. The test client applications for the UDDI registry and SerumS are equipped with a "timer" which measures elapsed time from sending the request till the return of the result (aka. response time).

6.2.1.2 Response time for the "register()" operation

To offer their functionality, both the UDDI registry and SerumS claim an authentication and authorization from service customers and service providers. The registration to a UDDI registry depends on each UBR node, whereby the registration to SerumS is based on a customer or provider profile. Because of the dependency of the registration process on each UBR node it is not possible to make a general conclusion about the performance of a UDDI registry. With SerumS the following results are obtained from the test:

- Customer's registration:

144 - 250 milliseconds depending on the options specified in the customer profile

- Provider's registration:

170 - 280 milliseconds depending on the options specified in the provider profile

6.2.1.3 Conclusion

The registration time for a provider is longer than that for a customer because a provider profile generally contains more information (options). In both cases the registration process costs less than half a second and is comparable to the time needed by Google for a simple search request[7].

6.2.1.4 Response time for the "publish()" operation

Using the "publish()" (aka. advertisement) interface specified in the UDDI specification, a service provider can place the information about its business and services in a UDDI registry. In our case, a service provider sends a profile and a WSDL document containing the technical specification of the Web service to SerumS, hence the response time needed for the "publish()" operation depends on two main parameters: the provider profile and the WSDL document. The time for handling the provider profile is already measured in the previous step (see "Time for a provider's registration"), so that we only need to measure the time for handling a WSDL document. During the validating process two WSDL documents with different technical specifications have been used:

- The technical specification of the Web service based search interface of Google.com "GoogleSearch.wsdl"[8]

  Time for publishing the GoogleSearch.wsdl document:

  700-820 ms depending on the network condition

- The WSDL-based technical description of the Publish/Subscribe Specification

  Time for publishing the WS-BaseN[9].wsdl document:

  850 - 870 ms depending on the network condition

[7] "simple search request" means a request atmost with 3 search key words
[8] http://www.google.com/apis
[9] "N" stands for "Notification"

The time for publishing the WS-BaseN.wsdl is longer than that for GoogleSearch.wsdl due to its size and complexity.

6.2.1.5 Conclusion:

We cannot directly compare the performance of the "publish()" operation between a UDDI registry and SerumS because a UDDI registry only receives a reference to the WSDL document and not the document itself. In case of SerumS, the content of a WSDL document is sent directly from the service provider and stored via XMLBeans as a local file on SerumS. Sending the content of a WSDL document over the Internet takes more time and is a disadvantage of SerumS, however, the following validation shows that the other operations benefit from this approach, e.g. with WSDL documents stored locally, SerumS can match more quickly between service customer profile, service provider profile and a WSDL document and notify changes of the Web service to the service customers.

6.2.1.6 Response time for the "inquiry()" operation

To get a reasonable result, the measurement of the "inquiry()" operation has been executed five times with different numbers of up to 300 of the Web service entries and up to 10 search criteria on both test UDDI registry and SerumS. The fluctuations in the following result are caused on one hand by the number of the Web service entries, which have to be managed by the broker, and on the other hand by the number of the search criteria specified in the "inquiry" string. Generally, the execution time in both systems SerumS and the test UDDI registry has increased with a noticeable value if the "inquiry" string contained more then 5 search criteria.

- up to 10 Web service entries on the broker site (UDDI registry or SerumS)

  UDDI:

  500 - 800 milliseconds

  SerumS:

  350-700 milliseconds

- with 10-50 Web service entries

  UDDI:

  700 - 1000 milliseconds

  SerumS:

  430-800 milliseconds

- with 50-100 Web service entries

  UDDI:

  800 - 1200 milliseconds

  SerumS:

  720 - 900 milliseconds

- with 100-300 Web service entities

  UDDI:

  1200 - 2400 milliseconds

  SerumS:

  900 - 2000 milliseconds

Figure 6.2 shows the average response times of both UDDI and SerumS and the possible time fluctuations depending on the number of the search criteria and of the Web service entities. The rectangles demonstrate the variance of the four sets of measurements. The red and green lines are simple interpolations which estimate the mean behaviour. The variance is greatly caused by the number of criteria.

6.2.1.7 Search correctness

Search correctness is the part of the whole search result (expressed in percentage), which fully matches the search criteria specified by the user. For example, if the user wants to search for "web site hosting service within Munich, Germany" and the search result contains four entries "car assurance within Munich", "web hosting service Berlin, Germany", "weather report service for Munich, Germany" and "web hosting service Munich, Germany", then the search correctness is $1/4 = 0.25 = 25\%$.

i. Correctness by search requests with "simple string pattern" (with at most 3 key words, e.g. search for a provider's name or provider's name with location)

   Result:

   UDDI: correctness 100%

   SerumS: correctness 100%

   (The result is obvious since both the jUDDI registry implementation and SerumS use the same String-Matching Rules of the Java-String API)

ii. Correctness by search request with more complex combination of:

   (a) until 4 criteria
   Result:
   UDDI: correctness 70%
   SerumS: correctness 70%

   (b) from 5 until 10 criteria (with extended search criteria like "expiry time", "input data condition" etc.)
   Result:
   UDDI: correctness 70%
   SerumS: correctness 95-100%

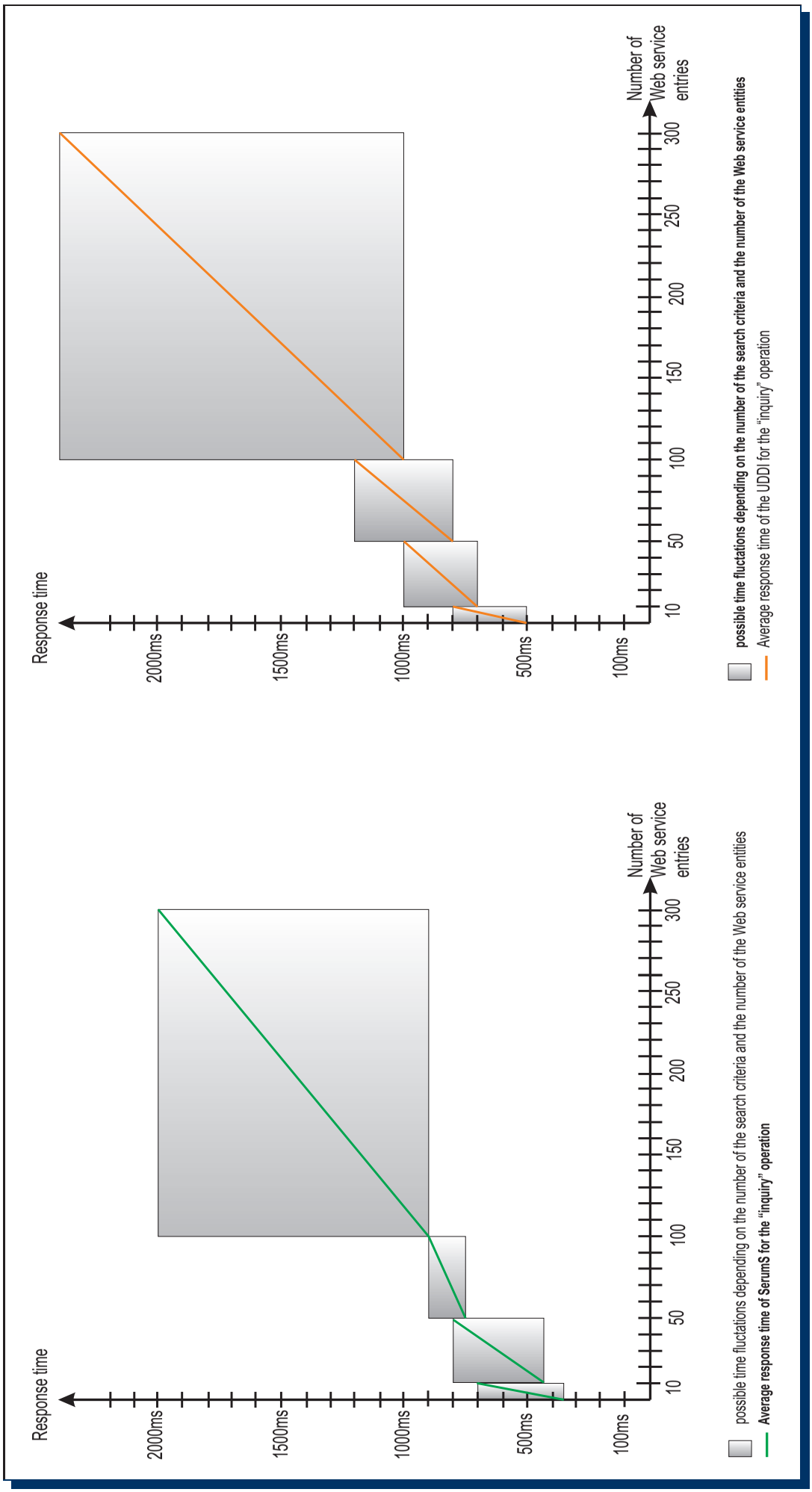

Figure 6.2: UDDI's and SerumS's response time for the "inquiry" operation

6.2.1.8 Search completeness

"Search completeness" in our context means "with which probability the search algorithm detects a matching Web service". Since there does not exist any statistic data regarding to the search completeness of the existing UDDI registries, the following assumptions are made:

i. Search completeness with a UDDI registry

   A UDDI registry can search only within itself for the Web services published by the service providers. New and/or not registered Web services in the UDDI registry cannot be recognized. Indeed, many service providers do not know the existence or the address of the UDDI registries and therefore do not publish their Web services to them. On the other hand, the registered Web services are distributed on many different UDDI registries, e.g. "Microsoft uddi Business Registry Node"[10],"IBM-UDDI"[11], "SAP UDDI v3 (beta) Test Business Registry[12]". Because of such distribution of existing UDDI registries, it is cumbersome for a service customer to search on all of them completely for a desired Web service. Therefore the search result completeness is reduced by an imaginable quotient compared to SerumS.

ii. Search completeness with SerumS

   SerumS searches proactively for Web services which are not yet known to service customers, because they are not published or published lately by the service providers. With the search functionality of Google the following advantages of SerumS compared to a UDDI registry arise:

   (a) The Web services are found by Google

   (b) By the nature of SerumS we assume that all Web services on all different Microsoft, IBM, SAP and other UDDI registries are known to SerumS and an immense completeness gain compared to a UDDI registry can be achieved.

6.2.2 Conclusion

In all configurations the search performance of SerumS is at least as good as that of the UDDI registry. We assume that a higher number of existing Web services, which have to be searched, delivers a better correctness with SerumS. The reason for the higher correctness is the fact that a provider profile contains more data (particularly referring to the Web service technical and conditional information) associated with Web services. The consequence is that the customer may specify the service with more details, which is used for the automatic search; in the UDDI approach this has to be done by additional inquiries to the provider which may even result in the insight that the service is not usable for the need of the customer. Hence, SerumS gives earlier and more accurate results.

The shorter response time of the elementary operations compared to the test UDDI registry is due to the fact that SerumS, on the one hand, uses XMLBeans for managing the XML documents locally, so that cumbersome SQL statements can be avoided; on the other hand, the set of the

[10] http://uddi.microsoft.com
[11] https://uddi.ibm.com/beta/registry.html
[12] http://udditest.sap.com/webdynpro/dispatcher/sap.com/tc uddi webui wdp/UDDIWebUI

test Web services is limited to 300 entries within the validation process. The performance gain compared to an UDDI registry will degrade at most to a certain value if SerumS has to deal with a higher number of service entries, and especially, if the structure of the service customer and service provider profiles get more complex. A DBMS will benefit from the capability of better indexing and more efficient reorganization of big amount of data. Therefore, in the long term, if the set of service customers and service providers increases, SerumS would better use a real DBMS instead of XMLBeans .

6.3 Quantitative Modelling (QAM)

6.3.1 Validation procedure and parameters

From the previous part EOM we have obtained the necessary basic results, which can be applied to the QAM. The goal of the QAM is to estimate the performance of SerumS and UDDI as a function of load. Compared to the EOM, where the service time of an operation (namely the time between the begin and the end of the operation logged with a timer) is measured, for the QAM a stationary multi-parameter asymptotic model is used to estimate the performance of SerumS and the resulting efficiency advantage compared to a traditional UDDI registry. The following parameters serve as input for the QAM:

i) Parameters regarding the service providers:

- Mean number of the services per service provider
- Rate of change of the Web services by the service providers
- Rate of change of the provider profiles

ii) Parameters regarding the service customers:

- Mean number of services used per service client
- Rate of change of customer profiles
- Rate of search for Web services (how often does a customer search for Web service(s))
- Rate of use of a Web service

The whole validating procedure is described in three parts. In the first part, the performance model of the traditional UDDI approach is presented, where the UDDI registry as service broker does not notify a service customer about changes in an existing service and a service provider also does not propagate any information about start or change of its service. A service client always has to "visit" the UDDI registry for a new Web service if it recognizes that the old one is not valid or not available anymore or it has not used it before. In the second part we calculate the performance of SerumS with respect to the extended functionality *"proactive searching for new Web services"* and *"notifying of changes at existing Web services to service customers"*. For both parts we compute the response time and estimate the response time under load for executing the service application between a traditional UDDI registry and SerumS using the results obtained from the previous EOM.

6.3.2 UDDI registry performance model

Figure 6.3 depicts the call relations between a Web service application and a traditional UDDI registry.

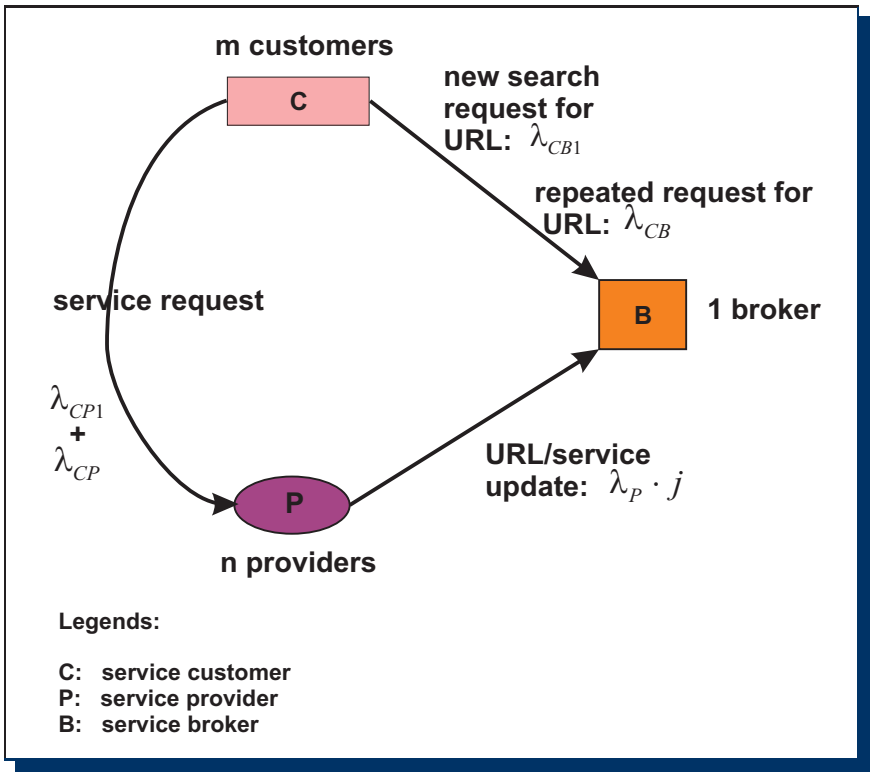

Figure 6.3: Call relations and call rates of a Web service application based on traditional UDDI registry

Within the service application process, the following service entities with their activities and associated times take part:

i) Client

- sends requests to the set of the service providers with a rate λ_{CP1} and service time b_P at the beginning of the service use. Before this, a request to the broker with a rate λ_{CB1} and service time b_{BS} (S: Searching) is necessary.
- sends requests to the set of the service providers at repeated use with the rate λ_{CP} and service time b_P
- sends requests to the service broker if service is stated as invalid or changed; rate λ_{CB}, service time b_{BS} and $2b_P$ for double requests to service provider.
- uses a mean number of k services

ii) Service broker

- no request since it is passive

iii) Service provider

- every service becomes invalid or non-specified with the rate λ_P and causes b_{BC} , the service time for updating an UDDI entry
- offers a mean number of j services

Explanation of the relation between the variables:[40]

We use an asymptotic approach (number of customers $m = 0$ and $m \to \infty$) with j , k and b as average values (maybe with large fluctuations). The $b's$ are service times, excluding the waiting time.

The probability $P[C/I$ since the last request] is the probability that the service has been changed (C) or become invalid (I) since the last request. A single service will be used with rate $\frac{\lambda_{CP}}{k}$ and will be changed or become invalid with the rate λ_P.

Figure 6.4 describes the situation, demonstrated on a discrete time-scale.

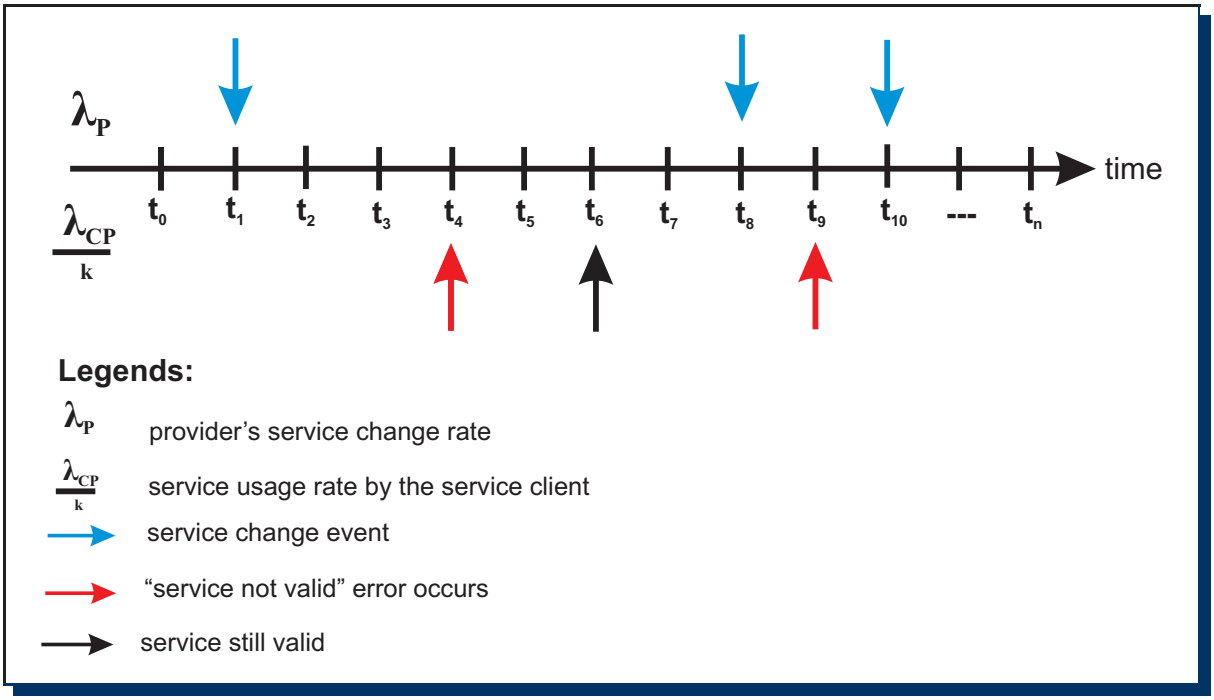

Figure 6.4: Service changed and invalid probability

In Figure 6.4, 2 events are to be recognized, when a "service not valid" error occurs and causes the service client to visit the service broker to get new and valid information about the service. In the first case, the service provider has changed the service at time point t_1 and the service client accesses it at time point t_4. In the second case, the service client accesses the affected service at time point t_9, immediately after the provider has changed its service at time point t_8.

$P[C/I$ since the last request$] = 1 - P[$no λ_P - events in the last Interval of the CP (Customer-Provider) process, rate $\frac{\lambda_{CP}}{k}] = 1 - \int_0^\infty (e^{-\lambda_P t} \cdot \frac{k}{\lambda_{CP}} \cdot e^{-\frac{\lambda_{CP}}{k}t})dt = 1 - \frac{\frac{\lambda_{CP}}{k}}{(\lambda_P + \frac{\lambda_{CP}}{k})} = \frac{\lambda_P}{\frac{\lambda_{CP}}{k} + \lambda_P} = \frac{k\lambda_P}{\lambda_{CP} + k \cdot \lambda_P}$

We obtain the following stationary utilization of the service broker:

$\rho_B = m(\lambda_{CB1} + \lambda_{CB}) \cdot b_{BS} + n \cdot j \cdot \lambda_P \cdot b_{BC} \leq 1$

For m we get the following equation (utilization does not exceed 1):

$m(\lambda_{CB1} + \lambda_{CB}) \cdot b_{BS} \leq 1 - n \cdot j \cdot \lambda_P \cdot b_{BC}$

Because of $\lambda_{CB} = \lambda_{CP} P[C/I \text{ since the last request}] = \frac{\lambda_{CP} \cdot k \cdot \lambda_P}{\lambda_{CP} + k \cdot \lambda_P} \Rightarrow$

$m \leq \frac{1 - n \cdot j \cdot \lambda_P \cdot b_{BC}}{(\lambda_{CB1} + \lambda_{CB}) \cdot b_{BS}} = \frac{1 - n \cdot j \cdot \lambda_P \cdot b_{BC}}{(\lambda_{CB1} + \frac{\lambda_{CP} \cdot k \cdot \lambda_P}{\lambda_{CP} + k\lambda \cdot \lambda_P}) b_{BS}} \lambda_{CB} \Rightarrow$

For "=" instead of $\leq$ we get m^* instead of m, the saturation value for the mean number of clients.

Effective service time for requests from the client to the provider:

$b_{CP} = b_P + \frac{\lambda_{CP1}}{\lambda_{CP1} + \lambda_{CP}} b_{BS} + \frac{\lambda_{CB}}{\lambda_{CP1} + \lambda_{CP}} b_{BS}$

The last summand disregards the request to the provider before C/I!

With $\lambda_{CB} = \frac{\lambda_{CP} \cdot k \cdot \lambda_P}{\lambda_{CP} + k \cdot \lambda_P}$

$b_{CP} = b_P + \frac{\lambda_{CP1}}{\lambda_{CP1} + \lambda_{CP}} b_{BS} + \frac{\lambda_{CP} \cdot k \cdot \lambda_P}{(\lambda_{CP1} + \lambda_{CP})(\lambda_{CP} + k \cdot \lambda_P)} b_{BS}$

We disregard that b_{BS} increases with n, j

Example calculation (with estimated service times and rates) :

$n = 100$

$j = 20$

$b_{BC} = 0,02s$ (broker time for handling changes of provider profile or Web services)

$b_P = 0,1s$ (provider time for handling a client request)

$k = 100$ (number of services used by a client)

$\lambda_p = \frac{10^{-7}}{s}$ (provider's service change rate)

$b_{BS} = 1s$

$\lambda_{CP} = \lambda_{CP1} = \lambda_{CB1} = 0,5/s$ (for the first time use of a service the service client always has to visit the broker for the service information. λ_{CP1} is the consequence of the mixed use of services with a very short lifetime at the client and long-used services. We assume a mean number $k = 100$ of services used by a provider, of which

- 50 services have a short mean usage time of $100s$; i.e. it follows by Little's formula that $\lambda_{CP1short} = 0,5/s$

- 50 services have a long mean usage time of $10^7 s$ (115 days); it follows $\lambda_{CP1long} = 50 \cdot 10^{-7}/s$

It is evident that the short used services determine $\lambda_{CP1} \approx \lambda_{CP1short} = 0,5/s$.)

$P[C/I \text{ since the last request }] = \frac{k \cdot \lambda_P}{\lambda_{CP} + k \cdot \lambda_P} = \frac{100 \cdot 10^{-7}}{0{,}5 + 100 \cdot 10^{-7}} = \frac{10^{-5}}{0{,}5 + 10^{-6}} \approx 2 \cdot 10^{-5}$

$b_{CP} = b_P + \frac{\lambda_{CP1}}{\lambda_{CP1} + \lambda_{CP}} b_{BS} + \frac{\lambda_{CP} \cdot k \cdot \lambda_P}{(\lambda_{CP1} + \lambda_{CP})(\lambda_{CP} + k \cdot \lambda_P)} b_{BS} = 0,1 + \frac{0{,}5}{0{,}5 + 0{,}5} \cdot 1 + \frac{0{,}5 \cdot 100 \cdot 10^{-7}}{(0{,}5 + 0{,}5)(0{,}5 + 100 \cdot 10^{-7})} \cdot 1$

$b_{CP} = 0,1 + 0,5 + 10^{-5} \approx 0,6s$

where waiting time for the broker is $0,5s$!

With $\lambda_{CB} = \frac{\lambda_{CP} \cdot k \cdot \lambda_P}{\lambda_{CP} + k \cdot \lambda_P} = \frac{0{,}5 \cdot 100 \cdot 10^{-7}}{0{,}5 + 100 \cdot 10^{-7}} \approx \frac{0{,}5 \cdot 10^{-5}}{0{,}5} = 10^{-5}/s \Rightarrow$

$m^* = \frac{1 - n \cdot j \cdot \lambda_P \cdot b_{BC}}{(\lambda_{CB_1} + \lambda_{CB}) \cdot b_{BS}} = \frac{1 - 100 \cdot 20 \cdot 10^{-7} \cdot 0{,}02}{0{,}5 + 10^{-5}} \approx \frac{1}{0{,}5} = 2$

For $m \geq 2$ the broker is the bottleneck (broker is completely overloaded!).

Check whether providers are bottleneck (load is shared on n providers):

$\rho_P = \frac{m}{n}(\lambda_{CP} + \lambda_{CP1}) \cdot b_P = \frac{1}{100} = 0,001 \leq 1$

it follows $m^* = 1000$, the providers are no bottleneck for less than 1000 clients. Providers are completely underloaded. With 2 clients the saturation of the broker begins, but $\rho_{provider} = 2 \cdot 10^{-3}$ for $m = 2$.

Figure 6.5 depicts the mean average response time for client requests to a provider including necessary requests to the broker.

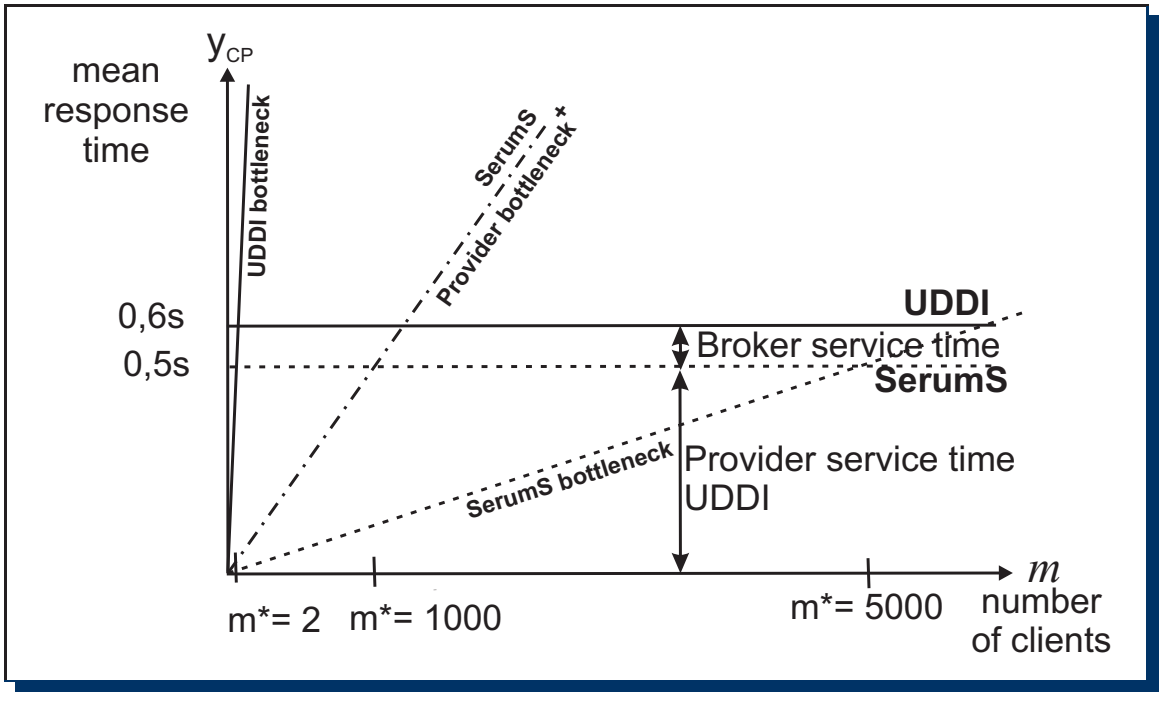

Figure 6.5: Asymptotic response time (y_{CP}) for the client's calls

6.3.3 SerumS performance model

6.3.3.1 Load for handling inner task

With SerumS, the following changes have to be considered for the calculation of the application process performance:

- For the "first-time" requests to the provider the broker (SerumS) is not needed, because the client knows the provider thanks to its profile (optimistic). Hence, this service time part $\frac{\lambda_{CP1}}{\lambda_{CP1}+\lambda_{CP}} b_{BS}$ is omitted for the client.
- At the C/I case, the client has also been already notified by SerumS and contacts directly the service endpoint which has been changed meanwhile by the service provider or goes to a new service provider who offers the same service, as it has been notified by SerumS. Again, the service time part $\frac{\lambda_{CP}}{\lambda_{CP1}+\lambda_{CP}} b_{BS}$ is omitted.
- In case of notification of changes of existing services (at the change rate λ_P) by a service provider, an appropriate entry has to be inserted in the service list managed by SerumS. In addition, SerumS has to check the matched profiles of the service customers in order to send them the new Web service addresses. The load for this process is $\lambda_P \cdot b_{Bcheck} \cdot j \cdot n$ (b_{Bcheck} is time for checking for one profile and maybe sending a message to the client), the part $\lambda_P \cdot b_{BC} \cdot j \cdot n$ (provider profile changed) remains.

Figure 6.5 shows that the service time for the client decreases to b_P and the b_{BS} parts disappear.

The load of the broker shrinks by $m(\lambda_{CB1} + \lambda_{CB}) \cdot b_{BS}$ but increases by $\lambda_P \cdot j \cdot n \cdot b_{Bcheck} \cdot m$

Assume that $b_{Bcheck} = 1s$,

thus

$\rho = n \cdot j \cdot \lambda_P \cdot b_{BC} + m \cdot \lambda_P \cdot j \cdot n \cdot b_{Bcheck} \leq 1$

or

$m^* = \frac{1-n \cdot j \cdot \lambda_P \cdot b_{BC}}{\lambda_P \cdot j \cdot n \cdot b_{Bcheck}} \approx \frac{1}{10^{-7} \cdot 20 \cdot 100 \cdot 1}$

$m^* = \frac{1}{2 \cdot 10^{-4}}$, thus m^* for broker: 5000. SerumS avoids the broker's bottleneck and makes the providers the bottleneck at a much higher performance level.

Serums dramatically improves the performance (about $5000/2 \approx 2500$ *times) situation by liberating the system from the broker bottleneck.*

6.3.3.2 Load for handling autonomous task

Assume that SerumS needs $5sec$ to search for new services in Internet, $1sec$ to check new services and $1sec$ to check changes at every existing service at a service provider. Assuming 20 services per provider (as in the previous calculations), it would cost $5sec + 100 \cdot 20 \cdot (1sec + 1sec) = 4005sec \approx 67min$ for SerumS to do the autonomous task.

With $60min \cdot 24(hours) = 1440min$ per day, SerumS needs $\frac{67}{1440}min \approx 0,05 \approx 5\%$ of its time for the autonomous task, which is acceptable. So this additional load is no argument against Serum.

6.4 Load Capacity Measurement (LCM) - Test of SerumS's functionality under load

6.4.1 General procedure

As mentioned at the beginning of this chapter, the purpose of LCM is to perform a stress test on SerumS only. On the one hand, the functionality of SerumS is tested and on the other hand, the load capacity (the execution time of SerumS under different load levels) is measured. The LCM is based on the EOM. The two elementary operations "register()" and "inquiry()" are used for the measurement. Each operation is executed in form of multiple parallel requests to SerumS. The load generator NeoLoad simulates a number of clients on the stimulator PCs with appropriate options configurable during the measurement, e.g. with request rate, number of users (how should the number of users be increased or decreased within a test time interval) and test duration. The whole measurement is performed with three different load levels "up to 60 parallel requests", "up to 140 parallel requests" and "up to 200 parallel requests". To ensure a reasonably stable result, 10 tests are executed per level and the average response time is calculated.

6.4.2 Hardware and software configuration

Unlike the EOM, the LCM uses a number of real client PCs. Since the available load generator application NeoLoad is available only as a trial version and limited to 10 parallel users, it is installed on 20 stimulator PCs, so that maximum 200 users can be simulated in parallel for the validation. Figure 6.6 depicts the LCM environment with the appropriate hardware and software configuration.

6.4.3 Tool description and measurement configuration

Figure 6.7 and 6.8 show the main screen of Neoload with the information about the destination server and the service endpoint where the requests have to be sent to.

The configuration for the destination server belongs to the "Repository" part of Neoload. For testing SerumS, the service endpoint http://131.159.41.23:8080/ws/customerSubscribe is used for a client to subscribe itself. "131.159.41.23:8080" is the network address of the SerumS with the port and "/ws/customerSubscribe" is the service endpoint.

The main screen also has two other tabs "Virtual User (VU)[13]" and "Population", which are depicted in Figures 6.9 and 6.10.

In the field "Virtual Users" one can create a number of instances of the VU. For testing SerumS,

[13] in SerumS's case they are service customers

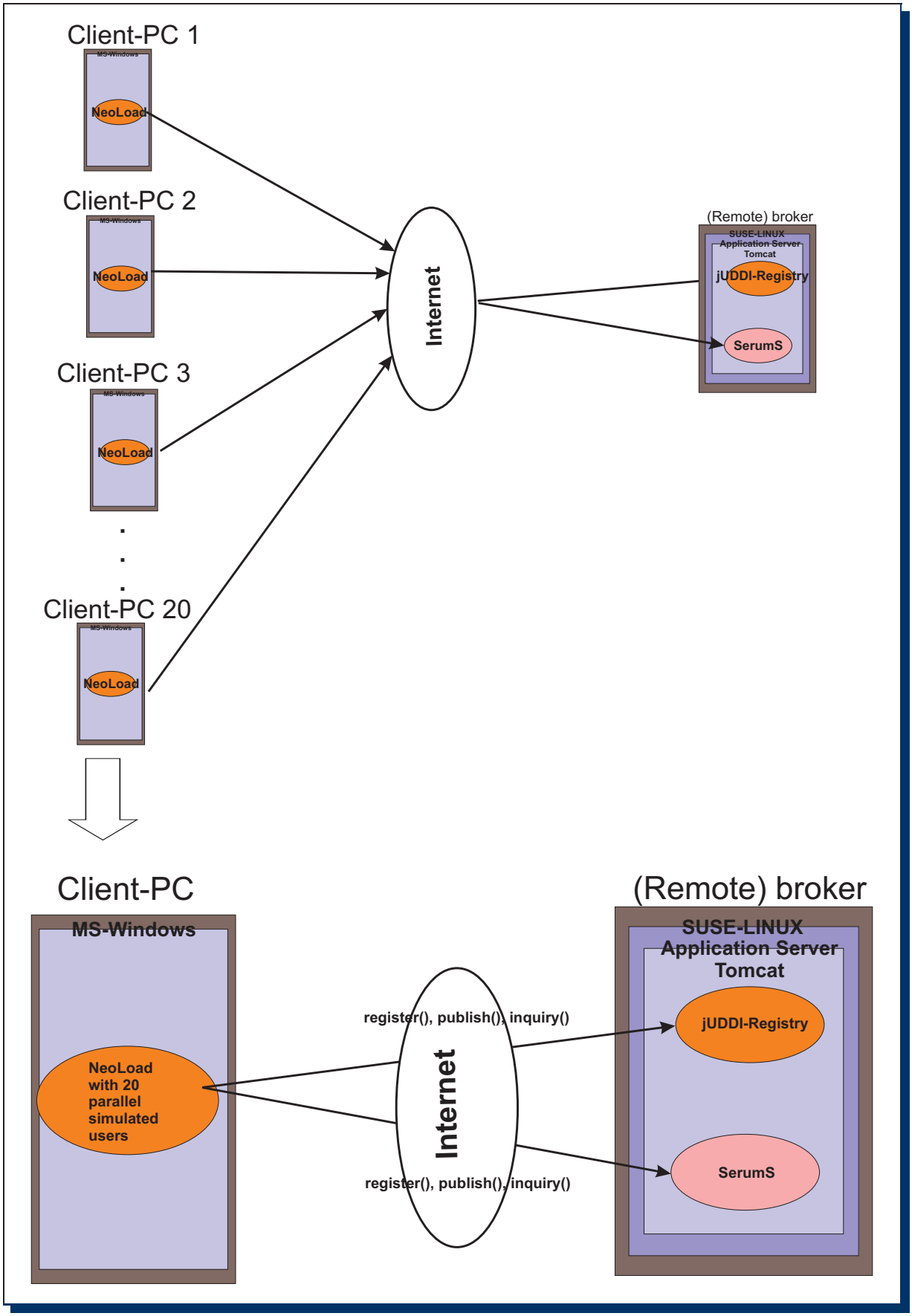

Figure 6.6: Hard- and software configuration for the LCM with 20 client (stimulator) PCs and the SerumS Broker

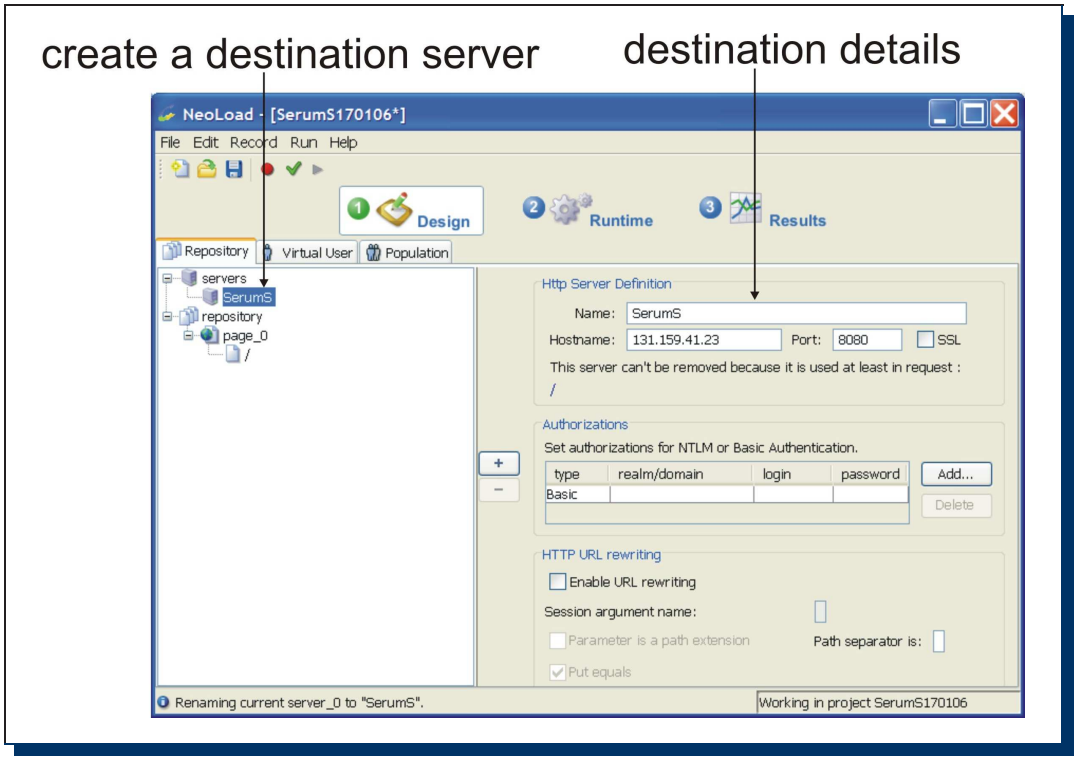

Figure 6.7: The main screen (Repository) of Neoload

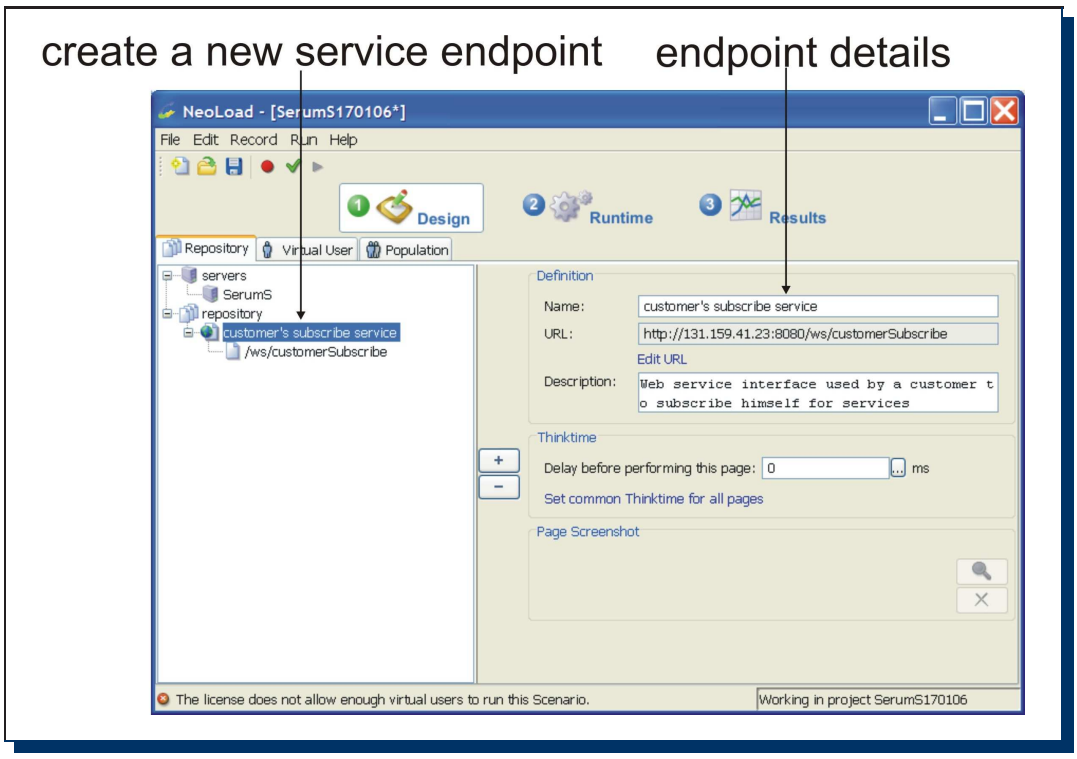

Figure 6.8: Detailed Information about the service endpoint

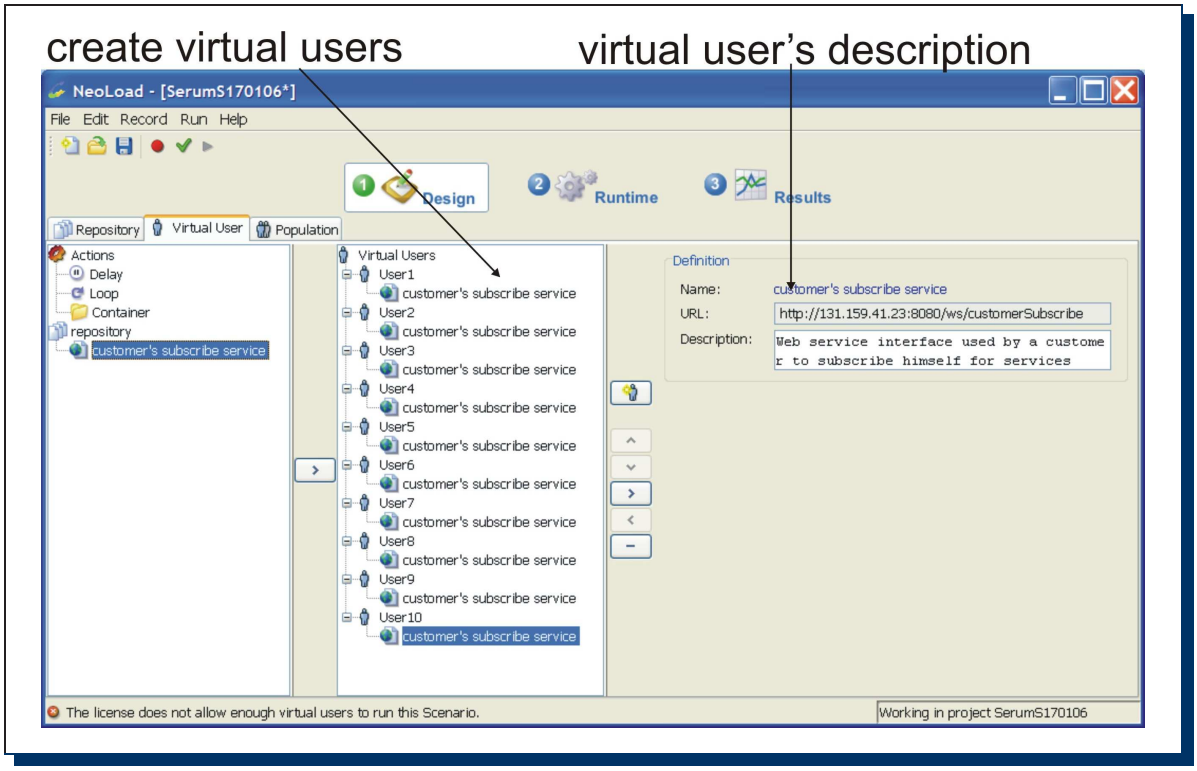

Figure 6.9: Creation and description of virtual users

10 VUs per PC are created which already reach the maximum number of VUs allowed by the trial version of NeoLoad.

In the population register one can configure the set of the virtual users which belong to a test case. In our case we have a 3-Users-Population, 6-User-Population and 10-User-Population with the following set up:

- Up to 60 parallel users:

Every PC simulates 3 parallel VUs, altogether: 3 x 20 (PCs) = 60 VUs

- Up to 140 parallel users:

Every PC simulates 8 parallel VUs, altogether: 7 x 20 = 140 VUs

- Up to 200 VUs:

Every PC simulates 10 parallel VUs, altogether: 10 x 20 = 200 VUs

Figure 6.11 shows the configuration capabilities within the "Runtime" tab. The meaning of each part of the "Runtime" tab is described below:

1) One can determine from where the requests have to be sent. In our case the requests are sent directly from the host where Neoload is installed. It is also possible for Neoload to cooperate with other hosts and send the test requests from them.

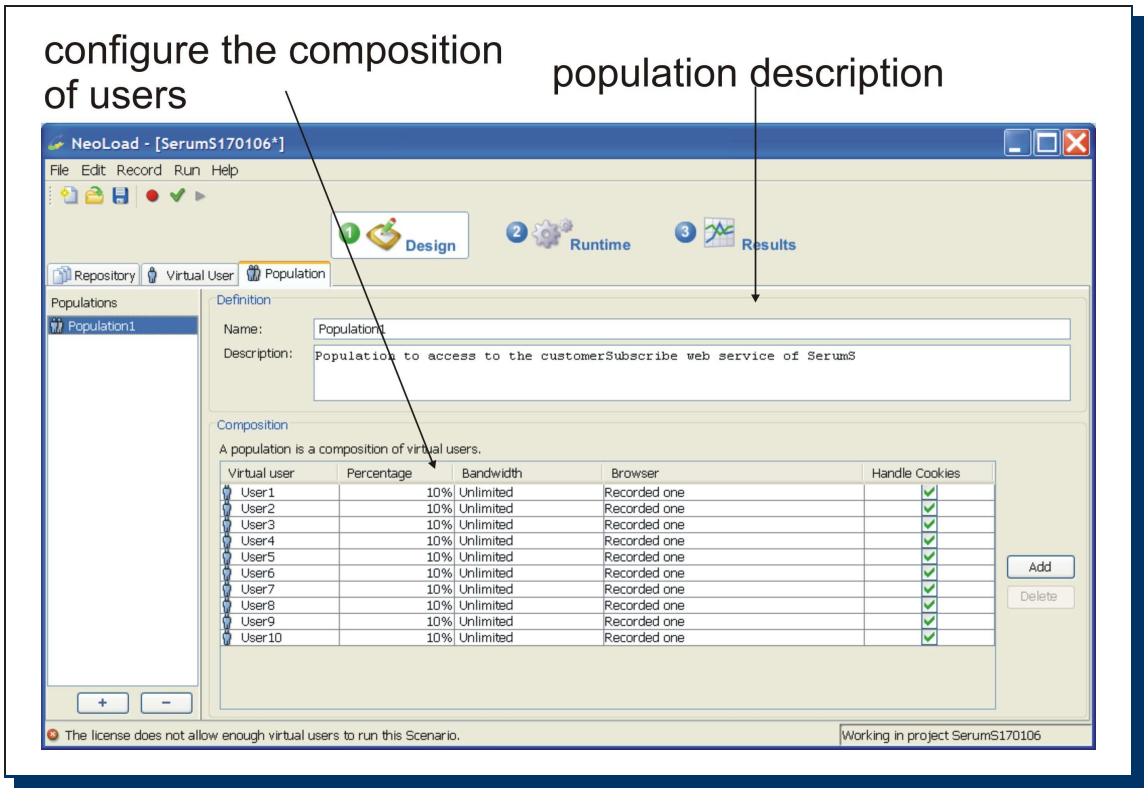

Figure 6.10: Population configuration for the Virtual Users

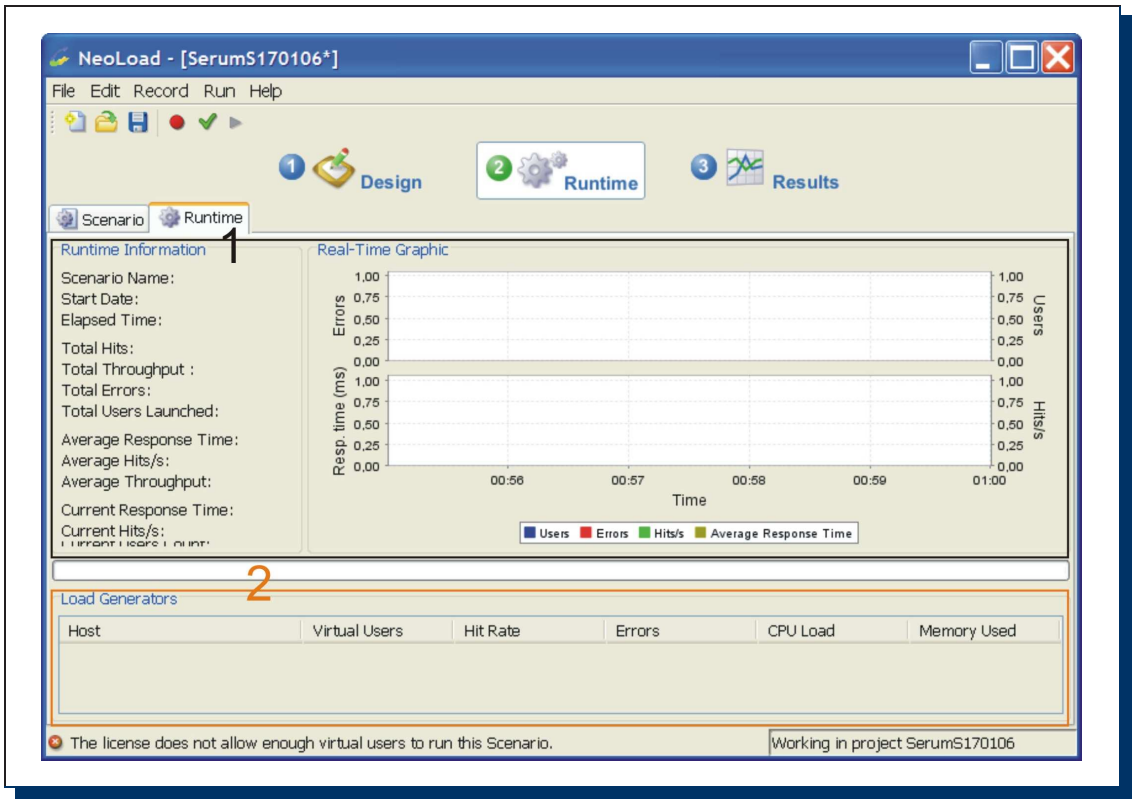

Figure 6.11: Configuration of the test scenario

2) Within the “Load Policy” tab the type of the “users volume variation” can be chosen. The “users volume variation” option indicates how the virtual users are run during the test. One can choose between “Constant variation”, if the number of the VUs is constant over time, “Ramp Up variation”, if the number of the VUs increases linearly with time, or “Peak variation”, if the number of the VUs increases to a peak value and decreases to a certain deep point. Because of the limitation of the number of virtual users which can be created in the trial version of the test software we choose the “users volume constant variation” whereby the “constant load for all the test span” is 1 VU.

3) The “Duration Policy” determines the duration of the simulation. One can stop the simulation manually or choose how long the simulation should run.

6.4.4 Validation result

As mentioned earlier, each test level is executed 10 times sequentially. Since both the test clients and SerumS are in the Munich Academic Network with negligible network delay (ping response time $< 1ms$) and SerumS is installed on a dedicated server PC which only has to listen to and handle the requests from the test clients, the fluctuations lie in the range of -2 up to +2 milliseconds. The following results are average response times recorded during the test.

In the first level, each of the 20 PCs simulates 3 parallel VUs. Neoload has started one after another but the average response time gets constant for every PC after a certain time. Figure 6.12 shows the result summary with 3 parallel VUs on every PC (altogether 60 VUs on 20 PCs).

The meaning of the information within the color ellipses is explained in the following:

1) The information about the test case (in our case a population with 3 parallel VUs on every PC)

2) The average response time is 352ms

3) The graph depicts the response time during the test (the gray curve marked within the ellipse)

One can conclude from the response time curve in the three graphs that SerumS works very constantly over time. Compared to the result from the EOM (see chapter 6.2.1.6) with 350ms for one single request, it follows that SerumS also can handle a multiple number of parallel inquiry requests without additional delay.

The second test level consists of 140 parallel VUs where every PC simulates 7 VUs. Figure 6.13 shows the result of the test. With 433ms for 140 parallel VUs the effective response time has not increased more than by 1 ms for every additional request compared to the result of the previous test level.

6.4.5 Evaluation of the test result

In this section the result of the LCM is summarized and evaluated in relation to the outcome of the different test levels.

We obtained the following result from the first level test (with 60 VUs) of the LCM (see Figure 6.13):

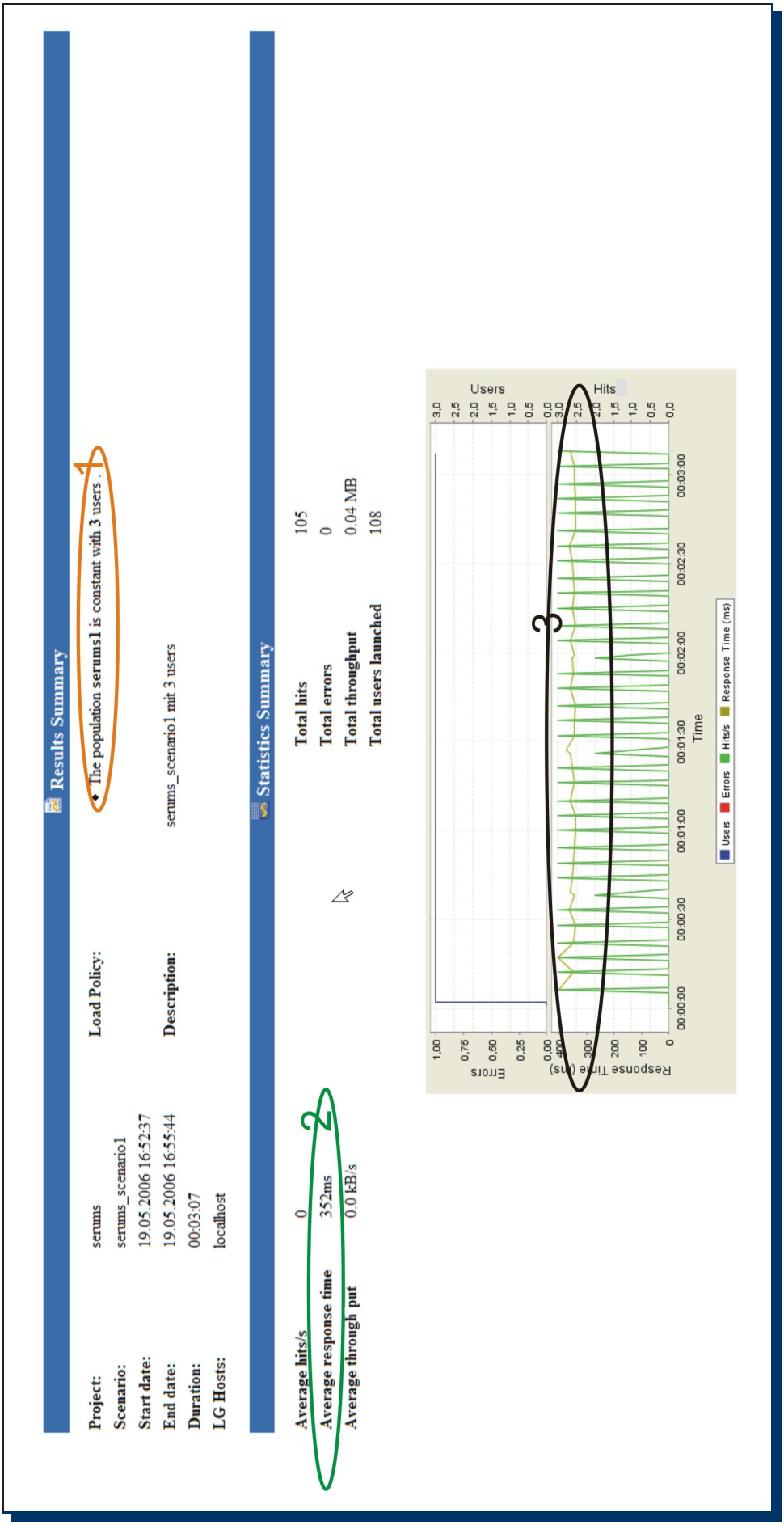

Figure 6.12: SerumS's performance with 60 VUs (inquiry requests)

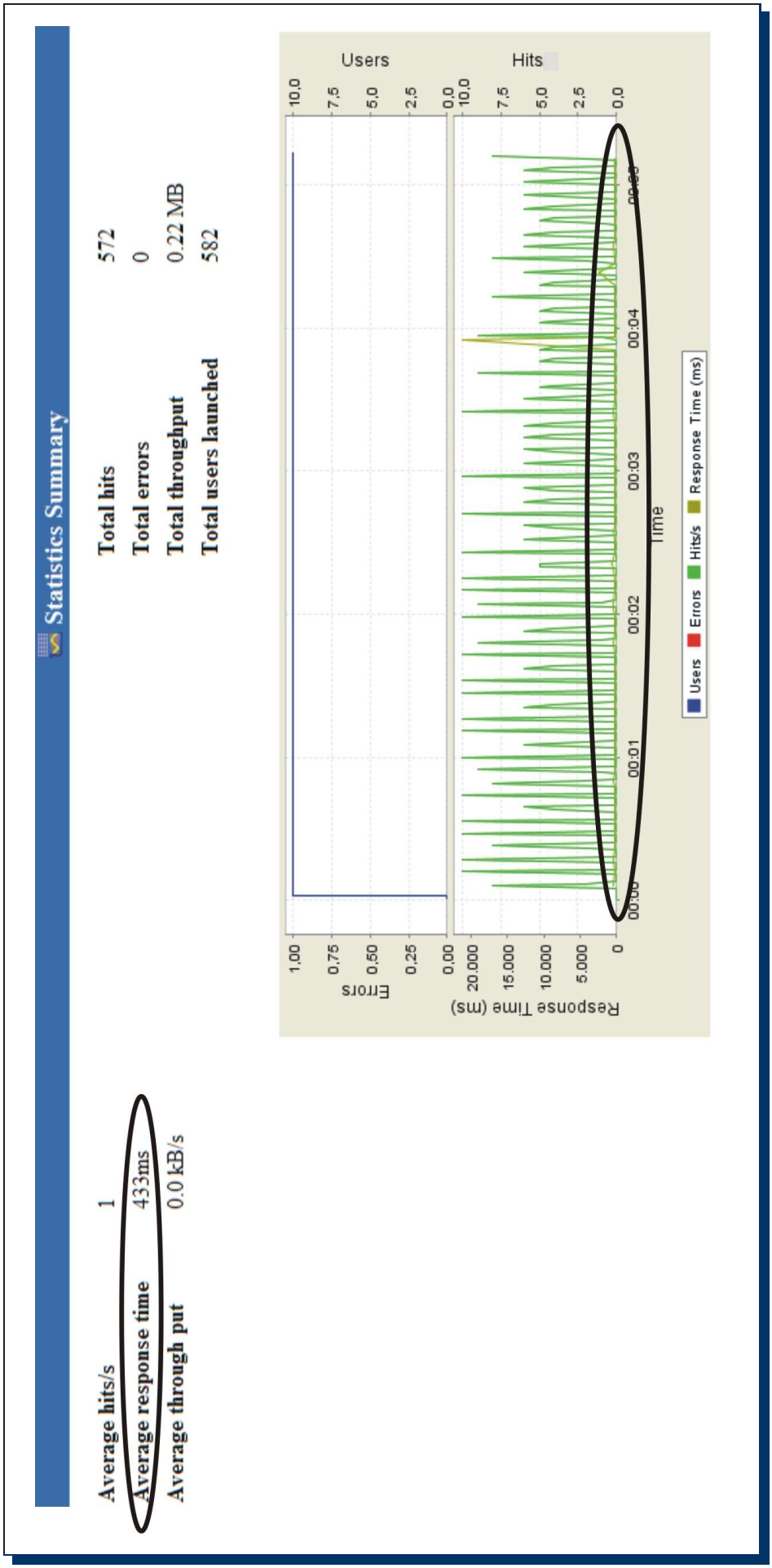

Figure 6.13: SerumS's performance with 140 VUs (inquiry requests)

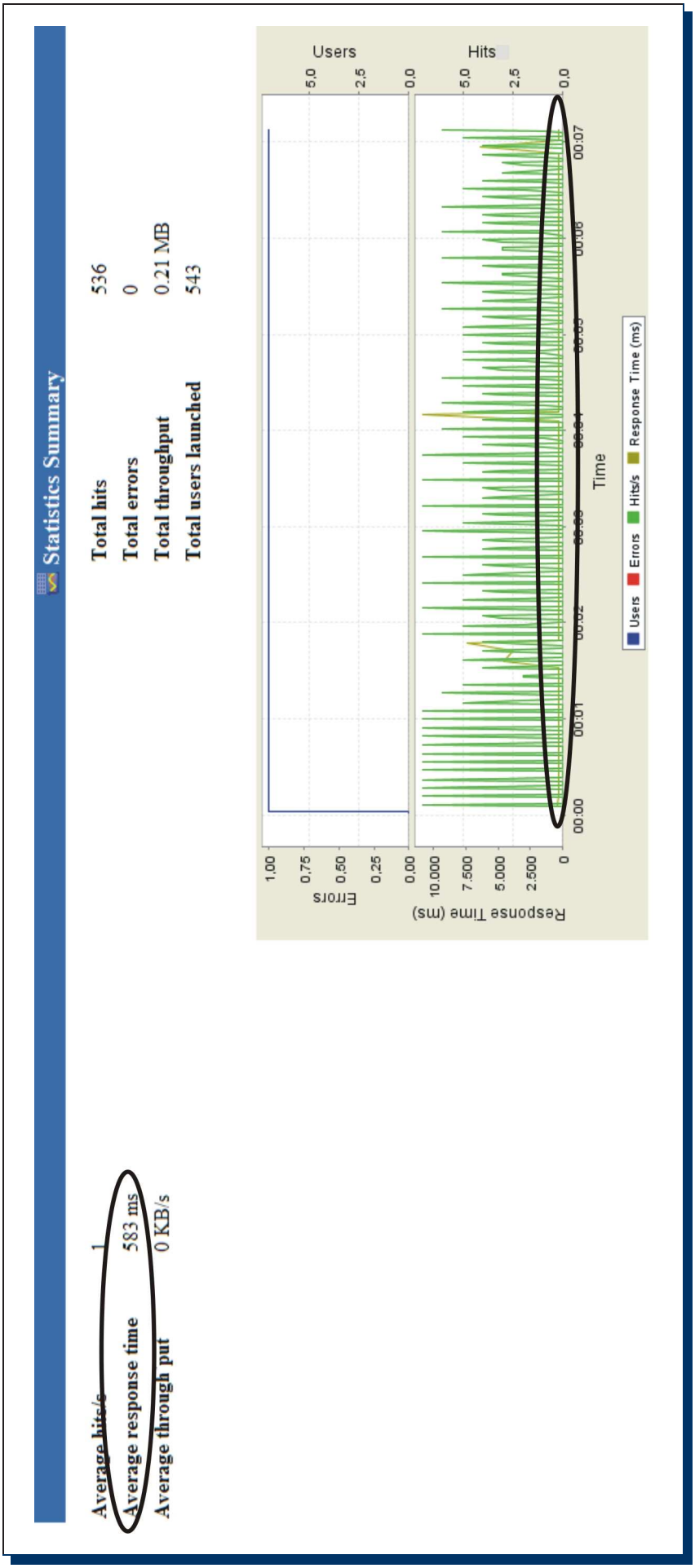

Figure 6.14: SerumS's performance with 200 VUs (inquiry requests)

Total users launched: 108

Test duration: 3 minutes

Test request rate: 1/sec

The important result is that SerumS only shows small average additional waiting time for handling a multiple number of the VUs (e.g. comparing the "Average response time/Total users launched" ratio between the first and second test level) as the following calculation shows:

The total number of users launched in the first test level: 108

The total number of users launched in the second test level: 582

Ratio: $\frac{582}{108} \geq 5$ (at least 5 times more lauchend users than in the first test level)

Average response time in the first test level: 352ms

Average response time in the second test level: 433

Response time difference: $433 - 352 = 81ms$,

Time increase in percentage: $\frac{81}{352} \approx 23\%$

These results say that SerumS only generates 23% *more time to handle 5 times the number of parallel inquiry requests. In the range up to 200 requests/min it shows only small average waiting times (*23%*).*

6.4.6 Résumé

SerumS did not show any significant weakness (crashes, slowdown or incorrectness) during all test levels of the LCM. Despite the limited possibility to validate SerumS because of the limited licence of the test tool NeoLoad, we can still get a reasonable result regarding its performance. The performance of SerumS is acceptable in a range up to 200 inquiries/min and the stability and robustness of the implementation is demonstrated by these test results.

Chapter 7

Conclusion and Future Works

This chapter surveys the results and concludes what is to be done in the future.

7.1 Conclusion

The original motivation for this thesis is the bad quality of the Web service information partially caused by lack of a "Subscribe/Notify" mechanism in the current UDDI standard, from which some serious issues arise, regarding consistency of the Web service information and performance of the execution of the Web service application process. The first chapter describes how in general the current Pull technique based distributed application systems works and which improvements can be achieved with the Pull-Push technique approach. After making the issues more specific, which come along with a traditional UDDI registry, some works are presented as current evolutions of this topic. However, these current works only contribute indirectly to the solution of the presented issues but they do not offer any acceptable approach within our context. The chapter 5 presents SerumS as a system for *efficient Search and Management* of the Web service contents by using the "Publish Subscribe Notification for Web services" specification to realize the "Subscribe/Notify" functionality which is missing in the current UDDI standard. With SerumS it is shown how the requirements for dealing with the problems mentioned in chapter 3 can be fulfilled. The "Proof of Concept" in the chapter 5 and the validation in the chapter 6 underline the feasibility and the advantages of SerumS compared to a traditional UDDI registry and also show how efficiently SerumS in a dynamic environment works, where the set of the service customers/providers as well as the Web services and their contents are permanently changed.

In summary the following contributions are achieved by our work:

i. The current UDDI standard is extended by a Publish/Subscribe mechanism

ii. Service customers can use the Publish/Subscribe mechanism to register themselves for desired Web services

iii. Service customers and service providers work proactively by delivering profiles. The service broker investigates the current offer of services and informs the customers about relevant services.

iv. Reduction of response time for service customers by automatic advance search for new Web services by the service broker (now SerumS)

v. Improved and extended specification (by metadata, e.g. “service expiry time”, “service excution condition” etc.) improves correctness and completeness of the search result. This releases service customers from cumbersome filtering process for relevant service information

vi. In-time reporting (notification) of service providers about service changes and new services to service customers reduces the number of the inquiry requests and leads to reduction of load on the broker (see chapter 6.3.3.1).

7.2 Future Works

In this sub-chapter further possibilities to extend and optimize the functionality of SerumS are determined so that it can be used in a larger context and for more purposes.

7.2.1 Improvement of the “proactive functionality”

One important functionality of SerumS is the autonomous task consisting of proactive search in Internet for new Web services and checking at the service providers for changes of existing services. Section 6.3.3.2 already shows that the time for this autonomous task needed by SerumS is negligible for 100 providers with each 20 services. However, if the number of the service providers increases, SerumS would spend more time for the automatic search and checking process as the following calculation shows.

The following assumptions are made for the calculation:

- 500 service providers

- Every service provider offers 20 services

- 5 seconds are needed for searching new Web services in Internet

- 1 second is needed for checking new services at a service provider

- 1 second is needed for checking changes of every existing service

Based on the calculation from chapter 6.3.3.2, SerumS would need the following time for the automatic operations:

$500 \cdot 20 \cdot (1sec + 1sec) + 5sec = 20005sec \approx 333min.$

With $60min \cdot 24(hours) = 1440min$ per day, SerumS needs $\frac{333}{1440}min \approx 0,23 \approx 23\%$ of its time for the autonomous task. Furthermore, if the number of the services offered by each service provider increases, the time for the autonomous task would also increase additionally.

The result above shows that SerumS still needs some improvements to be able to handle efficiently a big number of service providers and services. These improvements may be achieved

via technical or conceptional approaches. As technical approach one can implement an intelligent search algorithm to search more efficiently the Web services in Internet. Listing 7.1 shows an example how the *conceptional approach* is realised by specifying additional information in a service provider profile.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <prov:Provider xmlns:prov="provider.profiles.templates">
3
4    <prov:Contact>
5      <prov:Description>Samsung - South Korea</prov:Description>
6      <prov:Phone>012345678</prov:Phone>
7      <prov:Email>support@samsung.com</prov:Email>
8      <prov:Street>examples treet</prov:Street>
9      <prov:City>example city</prov:City>
10     <prov:Country>South Korea</prov:Country>
11     <prov:Continent>Asia</prov:Continent>
12   </prov:Contact>
13
14   <prov:Business>
15     <prov:Name>Samsung, Ltd.</prov:Name>
16     <prov:Description>
17       Provider Business Description XML
18     </prov:Description>
19     <prov:UNSPSCCategoryBag>10000000</prov:UNSPSCCategoryBag>
20   </prov:Business>
21
22   <prov:Service>
23     <prov:ServiceExpiryTime>2006-09-01</ServiceExpiryTime>
24     <prov:ServiceLimited>
25       <prov:ServiceLimitedID>
26         ff450ead12345678
27       </prov:ServiceLimitedID>
28     </prov:ServiceLimited>
29
30      <prov:ServiceValidTimeBeforeChange>
31        2006-11-31
32      </prov:ServiceValidTimeBeforeChange>
33
34     <prov:InputData>
35       <prov:InputDataName>customerID</prov:InputDataName>
36       <prov:InputDataType>stringType</prov:InputDataType>
37     </prov:InputData>
38   </prov:Service>
39
40 </prov:Provider>
```

Listing 7.1: Provider profile with "Not-changed" guarantee information

Within the element "<ServiceValidTimeBeforeChange>" in line 31 a service provider can specify how long a service is valid without any change until the given date. This gives SerumS a guarantee for the correctness and validity of the service, so that it does not need to check the service for changes during this time. Assume, that 150 of 500 service providers mentioned in the previous calculation specify this "Not-changed" guaranty information in all services for 2 months since they have been published, the time for the autonomous task would decrease as the following calculation shows:

$$350 \cdot 20 \cdot (1sec + 1sec) + 150 \cdot 20 \cdot 1sec + 5sec = 17005sec \approx 283min$$

With $60min \cdot 24(hours) = 1440min$ per day, SerumS spends $\frac{283}{1440}min \approx 0,2 \approx 20\%$ of its time for the autonomous task. Compared to the previous calculation (23% instead of 20% of the time), this approach would save SerumS about 3% of the time needed for the autonomous task. Dispite this approach, the load for the autonoumous task would be not bearable for a higher number of service providers which have to be handled by SerumS. An additional improvement is to parallelize some tasks which can be executed independently from each other. For example, searching for new Web services in Internet can be executed parallel to checking for new Web services and for changes of existing Web services at the service providers. One possible strategy is to apply a second machine or CPU, which enables the parallelizing process.

This example demonstrates how a little change of the service provider profile can affect the performance of SerumS in a positive way. Hence, future works should prefer conceptional improvements by adapting the XML data within the service provider profile for each use case. The *technical approach* need to modify technical details of the system and lead to recompiling of the source code, which is generally not recommended.

7.2.2 OWL-S Extension

One not cumbersome but meaningful enrichment which SerumS lacks is the capability to handle OWL-S as Web service description language. OWL-S is still in an (early) evolution stage but becomes more and more relevant because it offers the service provider a set of semantical constructs to describe its services in a better way. In the current architecture, a service provider sends a WSDL document with the whole description of the Web service within one "publish()" request to SerumS. However, an OWL-S-based Web service description consists of a set of documents (see 4.1) which altogether form a complete description. A set of separate "publish()" requests with appropriate SOAP messages (see 2.2) would cause traffic overhead and will result in bad performance for the execution of the Web service process, what contradicts the original idea of SerumS regarding efficient management and execution of the web service process. A possible solution is the integration of all parts of an OWL-S description in one SOAP message which has to be structured in a meaningful form so that SerumS can recognize and handle it in an efficient way. However, OWL-S offers a customer more information about a Web service (see 4.1.1)

7.2.3 QoS extension

The Quality of Service (QoS) aspect of Web services is very relevant for their evaluation. Currently SerumS combines the WSDL document and the provider profile to determine all properties of the Web services, however, there is no possibility for the provider to specify the QoS parameters to their services. Thus, if a customer wants to search for services with certain QoS conditions, he will not get satisfactory information from SerumS. Hence an extension of SerumS with the concept to give service providers a possibility to specify the quality of their services will make SerumS work even more precisely and offer the customers more satisfaction by giving them more information about the services and service providers and making the search process more efficient. In order to build this feature into SerumS there are independent "third parties" and an appropriate well-known QoS standard necessary. A service provider has to order an independent "third party" to monitor and evaluate the QoS of its Web services with certain criteria from the QoS standard before he publishes them to SerumS with additional entries in the provider profile as illustrated in Listing 7.2.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <prov:Provider xmlns:prov="provider.profiles.templates"
3  xmlns:qos="http://www.qos.org/ws/schemas">
4
5  ...
6
7      <prov:Service>
8        <prov:ServiceExpiryTime>2006-09-01</ServiceExpiryTime>
9      <prov:ServiceLimited>
10       <prov:ServiceLimitedID>
11         ff450ead12345678
12       </prov:ServiceLimitedID>
13     </prov:ServiceLimited>
14     <prov:InputData>
15       <prov:InputDataName>customerID</prov:InputDataName>
16       <prov:InputDataType>stringType</prov:InputDataType>
17     </prov:InputData>
18
19     </prov:QoS>
20       <qos:QoSOrganization>
21         <qos:QoSOrganizationName>
22           QoS Third Party Ltd
23         </qos:QoSOrganizationName>
24         <qos:QoSOrganizationDescription>
25           Some description of the organization
26         </qos:QoSOrganizationDescription>
27           <qos:QoSOrganizationContactDetails>
28         <qos:QoSOrganizationStreet>
29           ''the postal street name''
30         </qos:QoSOrganizationStreet>
31         </qos:QoSOrganizationContactDetails>
32         </qos:QoSOrganization>
33       <qos:QoSValues>
34          <qos:QoSNetworkBandwidth>
35            1GB/s
36          </qos:QoSNetworkBandwidth>
37          <qos:services>
38            <qos:service name="getStockInformation"
39               ID="ff001465CD">
40            <qos:serviceResponseTime>
41              160ms
42            <qos:serviceResponseTime>
43          </qos:services>
44       </qos:QoSValues>
45     </prov:Qos>
46
47     ...
48
49   </prov:Service>
50 </prov:Provider>
```

Listing 7.2: Provider profile with QoS information

The example illustrates how QoS information can be integrated in the provider profile.

- We assume that a definition of the QoS standard exists under the URL "http://www.qos.org/ws/schemas" (line 3) and we use its name space within the provider

profile.

- The “<QoSOrganization>” element (line 20) contains general information about the independent third party which evaluates the quality of the Web services.

- The independent third party gives information about the bandwidth of the network (line 34)

- The response time of the named service (interface) (line 40)

7.2.4 SLAs extension

Related to QoS are SLAs, which represent an important instrument for trading business contracts between business partners.

> “SLAs are contracts between service providers and customers that define the services provided, the metrics associated with these services, acceptable and unacceptable service levels, liabilities on the part of the service provider and the customer, and actions to be taken in specific circumstances...”[41]

Actually SLAs use QoS parameters by including appropriate rules into them to create business contracts which are negotiated between service provider and customer. Within the concept of SerumS, SLAs can be placed in a provider profile to inform a service customer about certain business agreements which have to be considered by using the affected services. The paper “A Logic Based SLA Management Framework” [42] describes how SLAs can be presented in the XML based rule based SLA language (RBSLA). Listing 7.3 shows an example of how RBSLA-based SLAs can be integrated into a provider profile to give more information about the Web services.

```
1  <?xml version=”1.0” encoding=”UTF−8”?>
2  <prov:Provider xmlns:prov=”provider.profiles.templates”
3  xmlns:qos=http://www.qos.org/ws/schemas
4  xmlns:rbsla= http://ibis.in.tum.de/staff/paschke/docs/rbsla/>
5
6    <prov:Service>
7      <prov:ServiceExpiryTime>2006−09−01</ServiceExpiryTime>
8      <prov:ServiceLimited>
9        <prov:ServiceLimitedID>
10         ff450ead12345678
11       </prov:ServiceLimitedID>
12     </prov:ServiceLimited>
13     <prov:InputData>
14       <prov:InputDataName>
15         customerID
16       </prov:InputDataName>
17       <prov:InputDataType>
18         String
19       </prov:InputDataType>
20     </prov:InputData>
21
22     <prov:QoS>
23      ...
24     </prov:Qos>
25
```

```
26      </prov:sla ServiceName="getStockInformation"
27         ServiceID="ff110024">
28
29        <rbsla:Implies>
30           <rbsla:head>
31              <rbsla:Atom>
32                 <rbsla:Rel>penalty</rbsla:Rel>
33                 <rbsla:Var>Service</rbsla:Var>
34                 <rbsla:Ind>100 Euro</rbsla:Ind>
35              </rbsla:Atom>
36           </rbsla:head>
37           <rbsla:body>
38              <rbsla:And>
39                 <rbsla:formula>
40                    <rbsla:Atom>
41                       <rbsla:Rel>less</rbsla:Rel>
42                       <rbsla:Cterm>
43                          <rbsla:Ctor>
44                             Availability
45                          </rbsla:Ctor>
46                          <rbsla:Var>
47                             Service
48                          </rbsla:Var>
49                       </rbsla:Cterm>
50                       <rbsla:Ind>
51                          95%
52                       </rbsla:Ind>
53                    </rbsla:Atom>
54                 </rbsla:formula>
55              </rbsla:And>
56           </rbsla:body>
57        </rbsla:Implies
58
59     </prov:sla>
60
61   </prov:Service>
62 </prov:Provider>
```

Listing 7.3: Example of RBSLA-based SLA within the provider profile

- The location of the RBSLA definition (line 4) whose name space is used in this provider profile.

- The name and (unique) identifier of the affected service which is bound to the SLA (line 26).

- Indication of penalty paid by the business party which can not fulfil a business liability (line 32).

- The description of the penalty condition (line 50).

The example shows the case what happens if the service “getStockInformation” is with 95% unavailable. In this case the service provider must pay 100 Euro as penalty to the service customer (line 34).

Appendix A

Java-based construct for searching Web services

```
1  .
2  .
3  .
4  // Setting how a UDDI has to handle the search result
5  FindQualifiers findQualifiers = new FindQualifiers();
6  Vector qualifier = new Vector();
7  qualifier.add(new FindQualifier("sortByNameDesc"));
8  findQualifiers.setFindQualifierVector(qualifier);
9
10
11 //Define the service type which has to be searched
12 KeyedReference key = new KeyedReference();
13 key.setKeyName("Internet related services");
14 key.setKeyValue("007406");
15
16 Vector keyedReferenceVector = new Vector();
17 keyedReferenceVector.add(key);
18
19 CategoryBag bag = new CategoryBag();
20 bag.setKeyedReferenceVector(keyedReferenceVector);
21
22 //find all Web services with the specified "service type"
23 ServiceList serviceList = (ServiceList) proxy.
24   find_service(businessKey,
25         names, bag ,null,findQualifiers,5);
26
27 // Process the returned ServiceList object
28 Vector serviceInfoVector = serviceList.getServiceInfos().
29   getServiceInfoVector();
30 String serviceName = "";
31 String serviceValue = "";
32 for( int i = 0; i < serviceInfoVector.size(); i++ )
33 {
34         ServiceInfo serviceInfo = (ServiceInfo)serviceInfoVector.
35           elementAt(i);
36
37  // Print name for each service
38 System.out.println("Name of Service : " + serviceInfo.
39   getDefaultNameString());
```

```
40            //Print the service key
41  System.out.println("Service key : " + serviceInfo.
42    getServiceKey());
43
44   // Print out the OverviewURL which contains
45       the physical address of the
46  WSDL document of the service provider
47  System.out.println("overviewURL of the BusinessService : " +
48  getOverviewURL(serviceInfo));
49
50  }
51
52  String getOverviewURL(ServiceInfo serviceInfo){
53  Node overviewNode = (Node) serviceInfo.getChildElementsByTagName().
54    item(0);
55  return overviewNode.getChildren().toArray()[1].toString();
56  }
```

Appendix B

Structure of a WSDL document

```
<?xml version="1.0"?>
[declarations of name spaces used in the document]
<definitions name=[name of the Web service definition]
[declarations of the data types used in the document]

<types>
        ...
</types>

[declaration of the interface signatures
(which are used in <portType> elements]:

<message name="...">
<part name="..." element="..."/>
</message>)*

[declarations of the call methods ("endpoint"),
which will be used in a "<binding name>" element]:

<portType name="a unique name">
      <operation name="name of the call method">
          <input message="name of the input call methode-signature"/>
          <output message=" name of the out call methode-signature "/>
      </operation>
</portType>)+

[declarations of "binding" elements which will be
used in "service" elements]:

     <binding name="..." type="...">
          <[binding-Protokoll]:binding style="..."
             transport="transport-Protokoll
 (e.g. http://schemas.xmlsoap.org/soap/http "/>
          <operation name="...">
             <[binding-Protokoll]:operation
                [binding-Protokoll]Action="..."/>
             <input>
               ...
             </input>
             <output>
                 ...
             </output>
```

```
43            </operation>
44        </binding>
45
46    </definitions>
```

Appendix C

WSDL document example

```
1
2  <?xml version="1.0"?>
3  <definitions name="StockQuote"
4   targetNamespace=http://example.com/stockquote.wsdl
5             xmlns:tns="http://example.com/stockquote.wsdl"
6             xmlns:xsd1="http://example.com/stockquote.xsd"
7             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
8             xmlns="http://schemas.xmlsoap.org/wsdl/">
9
10   <types>
11         <schema targetNamespace="http://example.com/stockquote.xsd"
12                 xmlns="http://www.w3.org/2001/XMLSchema ">
13              <element name="TradePriceRequest">
14                 <complexType>
15                     <all>
16                         <element name="tickerSymbol" type="string"/>
17                     </all>
18                 </complexType>
19              </element>
20              <element name="TradePrice">
21                 <complexType>
22                     <all>
23                         <element name="price" type="float"/>
24                     </all>
25                 </complexType>
26              </element>
27         </schema>
28  </types>
29
30   <message name="GetLastTradePriceInput">
31          <part name="body" element="xsd1:TradePriceRequest"/>
32          </message>
33
34          <message name="GetLastTradePriceOutput">
35          <part name="body" element="xsd1:TradePrice"/>
36          </message>
37
38   <portType name="StockQuotePortType">
39          <operation name="GetLastTradePrice">
40             <input message="tns:GetLastTradePriceInput"/>
41             <output message="tns:GetLastTradePriceOutput"/>
42          </operation>
```

```
43          </portType>
44
45          <binding name="StockQuoteSoapBinding"
46               type="tns:StockQuotePortType">
47          <soap:binding style="document"
48              transport="http://schemas.xmlsoap.org/soap/http"/>
49          <operation name="GetLastTradePrice">
50              <soap:operation
51                  soapAction="http://example.com/GetLastTradePrice"/>
52              <input>
53                   <soap:body use="literal"
54                       namespace="http://example.com/stockquote.xsd
55                       encodingStyle=
56                           "http://schemas.xmlsoap.org/soap/encoding/"/>
57              </input>
58              <output>
59                   <soap:body use="literal"
60                       namespace="http://example.com/stockquote.xsd"
61                       encodingStyle=
62                           "http://schemas.xmlsoap.org/soap/encoding/"/>
63              </output>
64          </operation>
65          </binding>
66
67      <service name="StockQuoteService">
68            <port name="StockQuotePortType"
69              binding="tns:StockQuoteSoapBinding">
70                          <soap:address location="http://soap.example.com"/>
71            </port>
72         </service>
73
74  </definitions>
```

Appendix D

GLUE Java-Class based WS objects of the "PeerChainApplication"

```
1  public class PeerChainServiceImpl
2  implements PeerChainService {
3  private PeerChainApplication app; // reference to application
4  public static String publish(PeerChainApplication app,
5  int port)
6  throws RegistryException {
7  PeerChainServiceImpl svc = new PeerChainServiceImpl(app);
8  Registry.publish(app.PEER_CHAIN_SVC_URN, svc);
9  // publish web service
10 return Registry.getPath(svc); // return web service URL
11 }
12 public void chainPeers(List peers) {
13 new Thread(new ChainPeers(peers)).start();
14 }
15 public void chainCompleted(List peers) {
16 new Thread(new ChainCompleted(peers)).start();
17 }
18 class ChainPeers implements Runnable {
19 ...
20 public void run() {
21 // find an unchained peer, else callback chainCompleted
22 peers.add(app.getWSDLUrl());
23 // add ourselves to the chain list
24 Set known = app.getKnownPeers();
25 List unchained = new ArrayList(known);
26 // create copy of known peers first
27 unchained.removeAll(peers);
28 // get set of known that are unchained
29 while (true) {
30 // attempt to reach one other peer
31 // until no other or success
32 if (unchained.isEmpty()) { // if no peers left to call
33 chainCompleted(peers);
34 // start chain completed callback
35 break; // finished
36 } else { // pick a random peer url
37 String url = (String)unchained.get(
38 app.randomInt(unchained.size()));
39 PeerChainService svc =
```

```
40  (PeerChainService)Registry.bind(url,
41  PeerChainService.class); // bind to web service
42  svc.chainPeers(peers);
43  // invoke chainPeers web service on peer
44  break; // finished
45              }
46          }
47      }
48  }
49
50  class ChainCompleted implements Runnable {
51  ...
52  public void run() {
53  // inform peer up the chain that the chain is complete
54  int mypos = peers.indexOf(app.getWSDLUrl());
55  // find this peer position
56  for (int i = mypos; i >= 0; iÑ) {
57  // attempt call up the chain
58  if (i > 0) { // this is not the last peer in the chain
59  String url = (String)peers.get(i − 1);
60  // get peer up the chain
61  PeerChainService svc =
62  (PeerChainService)Registry.bind(url,
63  PeerChainService.class); // bind to web service
64  svc.chainCompleted(peers);
65  // invoke chainCompleted on peer
66  break; // finished
67  } else if (i == 0) {
68  // we are the last peer to take the list
69  System.out.println(
70  ”in : peer chain completed at position ” + mypos);
71              }
72          }
73      }
74    }
75  }
```

Appendix E

Service customer profile XML schema

```
1  <?xml version="1.0" encoding="utf−8"?>
2  <xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema
3  xmlns:wstop="customer.profiles.templates"
4  targetNamespace="customer.profiles.templates"
5  elementFormDefault="qualified" attributeFormDefault="unqualified">
6
7    <xs:element name="Customer">
8      <xs:complexType>
9        <xs:sequence>
10         <xs:element ref="wstop:Contact" minOccurs="0"
11           maxOccurs="1"/>
12         <xs:element ref="wstop:Policy" minOccurs="0"
13           maxOccurs="1"/>
14         <xs:element ref="wstop:Service" minOccurs="0"
15           maxOccurs="1"/>
16       </xs:sequence>
17     </xs:complexType>
18   </xs:element>
19
20   <xs:element name="Contact">
21     <xs:complexType>
22       <xs:sequence>
23         <xs:element name="CustomerUUID"
24           type="xs:string"></xs:element>
25         <xs:element name="Description" type="xs:string"
26           minOccurs="0" maxOccurs="1"/>
27         <xs:element name="Phone" type="xs:string" minOccurs="0"
28           maxOccurs="1"/>
29         <xs:element name="Email" type="xs:string" minOccurs="0"
30           maxOccurs="1"/>
31         <xs:element name="Street" type="xs:string" minOccurs="0"
32           maxOccurs="1"/>
33         <xs:element name="City" type="xs:string" minOccurs="0"
34           maxOccurs="1"/>
35         <xs:element name="Country" type="xs:string" minOccurs="0"
36           maxOccurs="1"/>
37       </xs:sequence>
38     </xs:complexType>
39   </xs:element>
```

```
40
41    <xs:element name="Policy">
42      <xs:complexType>
43        <xs:sequence>
44          <xs:element name="MustHaveContact" type="xs:boolean"
45            minOccurs="0" maxOccurs="1"/>
46          <xs:element name="ValidedByServer" type="xs:boolean"
47            minOccurs="0" maxOccurs="1"/>
48          <xs:element ref="wstop:Ex-Providers" minOccurs="0"
49            maxOccurs="1"/>
50          <xs:element ref="wstop:Service" minOccurs="0"
51            maxOccurs="1"/>
52        </xs:sequence>
53      </xs:complexType>
54    </xs:element>
55
56    <xs:element name="Ex-Providers">
57      <xs:complexType>
58        <xs:sequence>
59          <xs:element name="Ex-Continents" type="xs:string"
60            minOccurs="0" maxOccurs="1"/>
61          <xs:element name="Ex-Countries" type="xs:string"
62            minOccurs="0" maxOccurs="1"/>
63        </xs:sequence>
64      </xs:complexType>
65    </xs:element>
66
67    <xs:element name="Service">
68      <xs:complexType>
69        <xs:sequence>
70          <xs:element name="Binding-Types" type="xs:string"
71            minOccurs="0" maxOccurs="1"/>
72          <xs:element name="CategoryBags" type="xs:string"
73            minOccurs="0" maxOccurs="1"/>
74        </xs:sequence>
75      </xs:complexType>
76    </xs:element>
77
78  </xs:schema>
```

Appendix F

Service provider profile XML schema

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3  xmlns:wstop="provider.profiles.templates"
4  targetNamespace="provider.profiles.templates"
5  elementFormDefault="qualified" attributeFormDefault="unqualified">
6
7    <xs:element name="Provider">
8      <xs:complexType>
9        <xs:sequence>
10         <xs:element ref="wstop:Contact" minOccurs="0"
11           maxOccurs="1"/>
12         <xs:element ref="wstop:Business" minOccurs="0"
13           maxOccurs="1"/>
14       </xs:sequence>
15     </xs:complexType>
16   </xs:element>
17
18   <xs:element name="Contact">
19     <xs:complexType>
20       <xs:sequence>
21         <xs:element name="Description" type="xs:string"
22            minOccurs="0" maxOccurs="1"/>
23         <xs:element name="Phone" type="xs:string" minOccurs="0"
24           maxOccurs="1"/>
25         <xs:element name="Email" type="xs:string" minOccurs="0"
26           maxOccurs="1"/>
27         <xs:element name="Street" type="xs:string" minOccurs="0"
28           maxOccurs="1"/>
29         <xs:element name="City" type="xs:string" minOccurs="0"
30           maxOccurs="1"/>
31         <xs:element name="Country" type="xs:string" minOccurs="0"
32         maxOccurs="1"/>
33       </xs:sequence>
34     </xs:complexType>
35   </xs:element>
36
37   <xs:element name="Business">
38     <xs:complexType>
39       <xs:sequence>
40         <xs:element name="Name" type="xs:string" minOccurs="0"
41           maxOccurs="1"/>
42         <xs:element name="Description" type="xs:string"
```

```
43              minOccurs="0" maxOccurs="1"/>
44            <xs:element name="UNSPSCCategoryBag" type="xs:string"
45              minOccurs="1" maxOccurs="1"/>
46          </xs:sequence>
47        </xs:complexType>
48    </xs:element>x
49
50  </xs:schema>
```

Bibliography

[1] M. Tian, A. Gramm, H. Ritter, J.Schiller, R. Winter: A Survey of current Approaches towards Specification and Management of Quality of Service for Web Services, PIK (Praxis der Informationsverwaltung und Kommunikation) 3/04, 27. Vol. 2004, page 133

[2] Steve Graham, IBM, Peter Niblett, IBM, Dave Chappell, Sonic Software, Amy Lewis, TIBCO Software, Nataraj Nagaratnam, IBM, Jay Parikh, Akamai Technologies, Sanjay Patil, SAP AG, Shivajee Samdarshi, TIBCO Software, Igor Sedukhin, Computer Associates International, David Snelling, Fujitsu Laboratories of Europe, Steve Tuecke, Globus / Argonne National Laboratory, William Vambenepe, Hewlett-Packard, Bill Weihl, Akamai Technologies: Publish-Subscribe Notification for Web services, Version 1.0, 03/05/2004
http://www.oasis-open.org/committees/download.php/6661/WSNpubsub-1-0.pdf

[3] Matt Reynolds: Web Services 101, What are Web Services?
http://www.webservicesarchitect.com/content/articles/reynolds01.asp

[4] Ed Ort: Sun Developer Network (SDN), Products and Technologies, Technical Topics, October 3, 2005
http://java.sun.com/developer/technicalArticles/WebServices/soa2/WhatsNewArticle.html

[5] Wolfgang Dostal, IBM Deutschland GmbH: Service-oriented Architecture powered by Semantic - Vision oder Illusion, December 12 2004
http://xml.fh-augsburg.de/xml-ak/2004-12-09/

[6] Kishore Channabasavaiah, Kerrie Holley, Edward Tuggle, Jr. : Migrating to a service-oriented architecture, Part 1, 16 Dec 2003
http://www-128.ibm.com/developerworks/library/ws-migratesoa/

[7] Wolfgang Dostal, Mario Jeckle, Info Melzer, Barbara Zengler: Semantic Web, JavaSPEKTRUM 5/2004, pp. 34

[8] Daniel Austin, W. W. Grainger, Inc., Abbie Barbir, Nortel networks, Inc., Christopher Ferris, IBM, Sharad Gard, The Intel Corporation: Web Services Architecture Requirements, W3C Working Group Note 11 Februar 2004
http://www.w3.org/TR/2002/WD-wsa-reqs-20021011#IDAGWEBD

[9] Heather Kreger, IBM Software Group: Web Services Conceptual Architecture, (WSCA 1.0), May 2001
http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf

[10] IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems: Business Process Execution Language for Web Services version 1.1, 30 Jul 2003 updated 01 Feb 2005
http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

[11] Erik Christensen, Microsoft, Francisco Curbera, IBM Research, Greg Meredith, Microsoft, Sanjiva Weerawarana, IBM Research: Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001
http://www.w3.org/TR/wsdl

[12] www.amazon.com: Amazon E-Commerce service
http://www.amazon.com/gp/browse.html/ref=sc_fe_l_2/104-9177053-1734311?%5Fencoding=UTF8&node=12738641 &no=14256891&me=A36L942TSJ2AJA

[13] SAP UDDI v3 Test Public Registry, Universal Description, Discovery and Integration (UDDI)
http://udditest.sap.com

[14] W3C: XML Schema, 12 September 2005
http://www.w3.org/2001/XMLSchema

[15] Peter Brittenham, Francisco Cubera, Dave Ehnebuske, Steve Graham: Understanding WSDL in a UDDI registry, Part 1, How to publish and find WSDL service descriptions, 01 Sep 2001
http://www-128.ibm.com/developerworks/webservices/library/ws-wsdl/

[16] John Colgrave (IBM), Karsten Januszewski (Microsoft), Francisco Curberea (IBM), David Ehnebuske (IBM), Dan Rogers (Microsoft): Using WSDL in a UDDI Registry, Version 1.08
http://www.oasis-open.org/committees/uddi-spec/doc/bp/uddi-spec-tc-bp-using-wsdl-v108-20021110.htm

[17] Mike Clark: UDDI - The Weather Report, The Outlook is mixed, November 28 2001
http://www.webservicesarchitect.com/content/articles/clark04.asp

[18] Shuping Ran: A Model for Web Services Discovery With QoS, ACM SIGecom Exchanges, Vol. 4, No. 1, 2003, page 2

[19] Romin Irani: Versioning of Web Services, Solving the Problem of Maintenance, August 8 2001
http://www.webservicesarchitect.com/content/articles/irani04.asp

[20] David Martin: OWL-S: Semantic Markup for Web Services, 24 Juli 2004
http://www.daml.org/services/owl-s/1.1B/owl-s/owl-s.html

[21] OWL-S: OWL-S 1.1 Release
http://www.daml.org/services/owl-s/1.1/

[22] Specification: Web Services Inspection Language (WS-Inspection) 1.0, November 2001
http://www-106.ibm.com/developerworks/webservices/library/ws-wsilspec.html

[23] Timothy Appnel: An Introduction to WSIL, 10/16/2002
http://www.onjava.com/pub/a/onjava/2002/10/16/wsil.html

[24] Tarak Modi: WSIL: Do we need another Web Services Specification?, Explaining the difference between UDDI, January 16 2002
http://www.webservicesarchitect.com/content/articles/modi01print.asp

[25] Timothy Appnel: Antroduction to WSIL 10/16/2002
http://www.onjava.com/pub/a/onjava/2002/10/16/wsil.html

[26] Microsoft Windows Server 2003, Enterprise UDDI Services: An Introduction to Evaluating, Planning, Deploying, and Operating UDDI Services, Microsoft Corporation, Published: February 21, 2003
http://www.microsoft.com/windowsserver2003/techinfo/overview/uddiguide.mspx

[27] Sun Microsystems Inc, The Internet Society
http://www.jxta.org/

[28] Wikipedia: Peer-to-peer, 8 September 2006
http://en.wikipedia.org/wiki/P2p

[29] Mario Schlosser, Michael Sintek, Stefan Decter, Wolfgang Nejdl, Standford University: A Scalable and Ontology-Based P2P Infrastructure for Semantic Web Services
http://www.kbs.uni-hannover.de/Arbeiten/Publikationen/2002/P2P2002.pdf

[30] Web Service Toolkit GLUE:
http://www.webmethods.com

[31] Ludwig Mittermeier, Roy Oberhauser: Peer to Pee in Theorie und Praxis, Teil 2: Ad-hoc-Web-Services durch P2P-Technologien, JavaSpektrum 4/2002
http://www.sigs.de/publications/js/2002/04/Mittermeier_JS_04_02.pdf

[32] Dominik Dahlem, David McKitterick, Lotte Nickel, Jim Dowling, Bartosz Biskupski, René Meier: Binding- and Port-Agnostic Service Composition using a P2P SOA, December 1, 2005
https://www.cs.tcd.ie/publications/tech-reports/reports.06/TCD-CS-2006-13.pdf

[33] Jun Shen, Yun Yang, Quang Huy Vu: SwinDeW-B: A P2P Based Composite Service Execution System with BPEL, pages 73-84, December 1, 2005
http://domino.research.ibm.com/library/cyberdig.nsf/papers/AABDCCBCB8F787DD852570D000570430/$File/rc23822.pdf

[34] Sun Microsystems Inc, The Internet Society: JXTA v.20 Protocols Specification,
http://spec.jxta.org/nonav/v1.0/docbook/JXTAProtocols.html

[35] Sun Microsystems Inc, The Internet Society: JXTA v.20 Protocols Specification, Section 2.2
http://spec.jxta.org/nonav/v1.0/docbook/JXTAProtocols.html

[36] Gunjan Samtani and Dimple Sadhwani: Web Services and Peer-to-Peer Computing, Companion Technologies
http://www.webservicesarchitect.com/content/articles/samtani05print.asp

[37] Jochen Dinger, Hannes Hartenstein: Die vermeintliche Robustheit von Peer-to-Peer Netzen
http://dsn.tm.uni-karlsruhe.de/english/print/397.php

[38] Jochen Dinger, Hannes Hartenstein: Defending the Sybil Attack in P2P Networks: Taxonomy, Challenges, and a Proposal for Self-Registration von Peer-to-Peer Netzen
http://dsn.tm.uni-karlsruhe.de/english/print/397.php

[39] The United Nations Standard Products and Service Code:
http://www.unspsc.org/

[40] Eike Jessen, Rüdiger Valk: Rechensysteme, Grundlagen der Modellierung, Springer-Verlag 1987, pages 362-391
http://www.unspsc.org/

[41] http://www.iec.org/, Service-Level Management, Definition and Overview
http://www.iec.org/online/tutorials/service_level/

[42] Adrian Paschke, Jens Dietrich, Karsten Kuhla: A Logic Based SLA Management Framework, 2005
http://ibis.in.tum.de/staff/paschke/docs/ISWC05_Paschke_final.pdf

[43] Universal Unique Identifier:
http://www.opengroup.org/onlinepubs/9629399/apdxa.htm