Forschungszentrum Jülich
in der Helmholtz-Gemeinschaft

Zentralinstitut für Angewandte Mathematik

Prospects and Realization of Flexible Service Offers in a Grid Environment

Morris Riedel

Jül-4154

Prospects and Realization of Flexible Service Offers in a Grid Environment

Morris Riedel

Berichte des Forschungszentrums Jülich ; 4154
ISSN 0944-2952
Zentralinstitut für Angewandte Mathematik Jül-4154

Zu beziehen durch: Forschungszentrum Jülich GmbH · Zentralbibliothek
D-52425 Jülich · Bundesrepublik Deutschland
☎ 02461 61-5220 · Telefax: 02461 61-6103 · e-mail: zb-publikation@fz-juelich.de

Abstract

This thesis aims at the process of flexible service offering and negotiation aspects to reach an agreement between service providers and service customers in Grid environments. At first, a general introduction to this process is given to identify the necessity for formal descriptions and work-flow in these processes. For a realization of such processes, concepts of service-based Grid architectures are analyzed that lead to the Grid Service technology and to special languages for service characteristics, including Service Level Agreement (SLA) parameters and Quality of Service (QoS) issues. The complexity of such service characteristics and their precise realization as services are the main goal of a whole working group of the Global Grid Forum (GGF) which leads the global standardization effort for Grid computing. So the specifications and the work of this Grid Resource Allocation Agreement Protocol (GRAAP) working group are the core of this thesis, enlarged with domain-specific requirements to create several prototypes addressing the switching of network connections.

Because this thesis also focuses on a realization of flexible service offers in a Grid environment, many technology frameworks are considered to find the best choice for now. Therefore, this thesis gives an overview of many currently available Web Services Resource Framework (WSRF) hosting environments to make a substantiated choice for creating WSRF compliant prototypes.

This thesis evolves a WSRF compliant design of an agreement factory and agreement service. There are also interoperable prototypes implemented and discussed but beside this a different approach, concerning the differentiated view of factory and resource is introduced. The complete realization of a working Grid Service application is out of the scope of this thesis. Conclusions sum up the cognitions compiled out of the prototype implementations and future work, concerning security aspects and persistent agreements, is described.

Acknowledgments

This diploma thesis was written at the Central Institute for Applied Mathematics (Zentralinstitut für Angewandte Mathematik, ZAM) of Research Center Jülich. I would like to express my gratitude to the many people who have helped me directly or indirectly with this thesis and my study of General Informatics at the University of Applied Sciences Cologne.

First and foremost, I would like to thank my advisor, Dr. Volker Sander, who is a scientist at the ZAM and responsible for managing different projects in the context of Grid Computing. This thesis would not have been possible without his invaluable advice and continuous support. I also want to express my gratitude to Dietmar Erwin, head of ZAM department Operating Systems, for giving me the possibility to write this thesis at the ZAM and his suggestions. I would like to acknowledge Philipp Wieder, Dr. Roger Menday and Dr. Wojciech Klimala for their supports, ideas and many technical discussions.

I am also deeply indebted to Prof. Dr. Frank Victor, University of Applied Sciences Cologne, for his effort in guiding and supporting my study of General Informatics. His suggestions, ideas and comments were more than helpful to me and did never stop. Thanks to Prof. Dr. Lutz Köhler, who give me important knowledge of working with heterogeneous distributed systems and to Prof. Dr. Erich Ehses for his endless time, inspiration and his lectures of getting a technology foresight.

Special thanks to Ulrich Winterhoff, holder of Starlogik Company Kierspe, for giving me the chance to work with brand-new technologies and to share trainings and roadshows concerning future techniques. It was and is an honor to collaborate with Thomas Clever, who teached me the strict and precise view on details during the implementation of software, and Clemens Otte for all the discussions about Internet technologies.

I would also like to express my appreciation to Lars Fischer, Oliver Bücker and Stephan Graf for all the technical discussions we had and their incredible support. I wish to heartily thank all the authors for writing the detailed specifications and very interesting papers I work with. Furthermore, thanks to all the other friends, colleagues, and collaborators who assisted and encouraged me during my studies.

Above all, I would like to express my deepest gratitude to the people I love and to whom I would like to dedicate this thesis: My parents Horst and Christa, who have aided me during my studies and my lovely girlfriend Cordula.

Morris Riedel
August 2004, Kierspe, Germany

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The term Grid describes an emerging distributed computing infrastructure that facilitates the routine interaction between service requester and provider. A particular problem that arises in such an environment is that a service request often relies on particular capabilities provided by the service. This implied level of service is of major importance during the selection process of a service provider since its actual capabilities might depend on the resource situation at the requested time of service. In modeling the implied level of service explicitly as a Service Level Agreement (SLA) we formalize the relation between service requester and consumer. However, the question arises how these SLAs are established. This leads to the problem of formally defining agreement terms as well as a standard way for negotiating them. In absence of a widely accepted standard, agreements are often—if modeled explicitly at all—created by some proprietary mechanism assuming a specialized scenario without any predefined structure. The lack of a common agreement structure and standardized methods to create an agreement is contrary to usual fundamental design principles of software engineering. Furthermore, if agreements were modeled explicitly, the question arises how the related service description terms and Quality of Service (QoS) parameters were interpreted.

These problems motivate the creation of a uniform way for agreement negotiations and to define a general structure of such an agreement. Clearly, service guarantees must be monitored and any failure to meet these guarantees must be propagated to service consumers. This facilitates the development of applications that can revert to a language and a well-defined protocol for advertising the capabilities of service providers. This is accomplished by creating agreements based on formal specified creational offers and a standard specification. The problem of setting concrete QoS terms is nontrivial, because the monitoring of agreement compliance at runtime is also important. Furthermore, the definition of QoS description terms by some formal language allows the automatic interpretation and validation of agreements. This thesis will present a comprehensive analysis of these problems and will present potential solutions based on recent efforts.

In particular, this thesis is motivated by recent efforts of the Grid Resource Allocation Agreement Protocol (GRAAP) working group of the Global Grid Forum (GGF). The earlier work of this group and the current working draft of the Web Service agreement specification ([**WS-Agreement**]) have been obtained for this thesis. While this draft provides a detailed specification this thesis provides a practically oriented realization of the components defined in [**WS-Agreement**]. Furthermore, this thesis applies the concept of domain-specific extensions for an existing network service to specialize the structure of general agreements.

A particular focus of this thesis was given by the recent introduction of the Web Service Resource Framework (WSRF) ([**WSRF**]) . The aim of this thesis is to focus on how - and if - it is possible to implement the agreement creation process defined by the GRAAP working group of the GGF within the WSRF architecture. Furthermore, an enlargement for the normal *'WSRF factory pattern'* is introduced, which is called the *'autonomic factory approach'* by the author. This approach supports the creation of one central factory for multiple different services using one central policy decision point and is hence advancing the role of a factory. The question on how much responsibility a factory should and must have is one of the contributions of this thesis.

The thesis is structured as follows: The first chapter gives the basic background information and introduces the core terminology. This background is then extended to contain more theoretical backgrounds and a survey of potential solutions to provide flexible service offers in a Grid environment. The last chapter is a more practical chapter that describes prototypes and discusses the different *'autonomic factory approach'*.

In detail, Chapter 2 clarifies the term service and examines existing service architectures. The chapter introduces the Web Service technology as an advantageous service architecture and motivates Web Services as a choice for Grid infrastructures. Furthermore, it focuses on Web Service enhancements that provide the means to address the demand for flexible service offers in Grid environments. After introducing the Open Grid Services Architecture (OGSA) the chapter will discuss existing approaches for Service Level Agreement languages.

Chapter 3 gives the foundation of the thesis by guiding to the recently introduced Web Services Agreement draft. It will explain agreements and a simple agreement negotiation message exchange that is used for creating an agreement. Furthermore, Chapter 3 introduces high level examples to explain the need for standardized agreement negotiations.

Since the aim of the thesis is the realization of flexible service offers in a grid environment, Chapter 4 presents an evaluation of potential technology frameworks that could be used to achieve these goals. It will explore the constraints of Grid environments and will map these to the different approaches starting with the well-known Open Grid Services Infrastructure (OGSI) approach, its evolution and refactoring process, leading to the WSRF architecture.

'WSRF compliant Services' can be realized with an WSRF hosting environment and correspondent libraries. Therefore, Chapter 5 gives a comprehensive analysis of currently available WSRF hosting environments.

Chapter 6 will introduce the *'autonomic factory approach'* for WSRF in agreement negotiations.

Finally, Chapter 7 presents a conclusion and future work related to the possibilities of combining different Web Service technologies.

Chapter 2

Concepts of Service Based Grid Architectures

This chapter provides an introduction to Service Oriented Architectures in general and introduces the Web Service technology as an advantageous service architecture of today.

2.1 Service Oriented Architectures

The term Service Oriented Architecture (SOA) is one of today's most used terms in the context of Web technology related to services. The demand for flexible e-business services grows permanently and is since 2002 also being influenced by an infrastructure that advances science: Grids. This section provides an overview of the ideas beyond Service Oriented Architectures.

2.1.1 Flexible e-business Services

A world in which thousands of e-business services connect and collaborate with each other over the Internet is becoming more and more real. These collaborations, also called Business-to-Business (B2B) services, use a wide variety of interactions that range from simple end-to-end Electronic Data Interchange (EDI) interactions to complicated Web auctions, like eBay or eSteel. EDI is a transfer standard between organizations which is becoming increasingly important as an easy mechanism for organizations to buy, sell, and trade pieces of information or goods over the Internet. However, this format is more and more substituted by Extensible Markup Language (XML) documents. But according to [**Wikipedia**] EDI is still the engine behind 95% of all electronic commerce transactions in the world and many corporations have already made some of their IT services available to their customers using this technology. This results in a variety of e-business services. While stock tickers or product catalogs simply provide information, other services like mission-critical B2B commerce transactions, such as multi-million-dollar purchases of material by a correspondent manufacturer. Consistent well-understood conventions will eventually evolve out of trial-and-error techniques whether or not e-business participants make formal efforts to agree on a common architecture for service collaboration ([**Tao**]). But those trial-and-error techniques are inappropriate, often expensive and slow, leading to the demand for a generally accepted unifying architecture that makes it clear how various services can work together.

Many companies have proposed service architectures, for instance the Microsoft BizTalk initiative, which help organizations to efficiently and effectively integrate systems, employees, and trading partners through manageable business processes. Most important is the aim to enable organizations to automate and orchestrate interactions in a highly flexible and automated manner ([**BizTalk**]). Another proposal for a service architecture is Sun's Jini system. The main concept of Jini is a set of components that provides an infrastructure for federating services in distributed heterogenous environments. Furthermore, Jini specifies a complete programming model that supports and encourages the production of reliable distributed services. Jini enables organizations or users to share services and resources over a network ([**JINI-Spec**]). However, according to [**Tao**], these proposals lack sufficient generality to cause a worldwide infrastructure of services based on these proposed architectures. The problem is that each vendor envisions a worldwide infrastructure of interacting services, but the realization of services depends on proprietary technology This leads to the demand for an architecture, which is independent of specific programming languages or operating systems, and uses open, for example standardized, transport technologies, for instance Hypertext Transfer Protocol (HTTP) and standard data encoding techniques, like XML.

2.1.2 Ideas beyond a Service Oriented Architecture

A Service Oriented Architecture (SOA) focuses on the description and organization of services to support their dynamic, automated discovery and use ([**Tao**]). SOA is not based on manually hardwired interactions that were commonly used in EDI applications. But before describing such an architecture it is necessary to understand what a service is and how a service can be defined.

Definition
*A service is a function that is well-defined, self-contained, and does not depend on the context or state of other services ([**SOA-Definition**]).*

A service can be seen as an endpoint of a network connection which has normally some type of underlying computer system that can do something special, for example computing some complex mathematical problems. Note that services and the enlarged model of Service Oriented Architectures are not new. According to [**SOA-Definition**], a modern Service Oriented Architecture is essentially a collection of services that communicate with each other. This communication process can involve simple passing of data or it can involve two or more services coordinating some activity.

The fundamental idea of an SOA is that there are three different roles, namely service provider, service broker and service requestor, which communicate with each other and work loosely-coupled together. While service providers are publishing the availability of their services, service brokers are registering and categorizing such published services to provide the search for services. A service requestor can therefore use a service broker to find a needed service and then employ that service. The collaborations among these roles are supported by standardized network protocols and these collaborations are depicted in Figure 2.1. In SOA service descriptions are formulated in standard XML. The idea beyond these platform-neutral service descriptions is that they can be easily used for all three roles, and the content provides the required information to categorize, choose and invoke e-business services ([**Tao**]). That means a service description implies the semantics and the message Application Programming Interface (API) of the service. The API is described using XML, including named method calls,

named arguments and typed arguments. Often the semantics of the description begins with a human-readable service description of the service offers. Finally, both human-readable service description and the XML API definition guide the placement of the service in the categorization, for instance in service brokers. And this, in turn, allows services an automated search for other services and their automated use. Note that service descriptions can also specify non-functional requirements, for example security, authentication and privacy issues related to the exchange of information necessary for the service to be consummated.

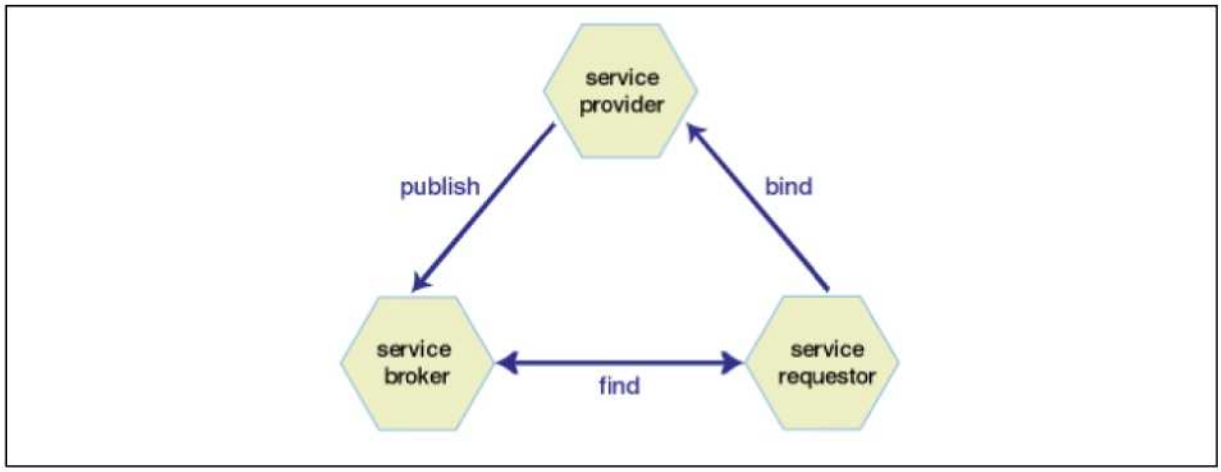

Figure 2.1: Relationship between the three roles in an SOA ([**Tao**]).

Normally, a service requestor or customer sends a service request message to a service provider and the service provider returns a response message to the service customer. The request and response messages are defined in a way that is understandable to both service requestor and service provider ([**SOA-Definition**]). Figure 2.2 shows this simple message exchange mechanism. Note that the service provider itself can also be a service consumer or requestor of other services.

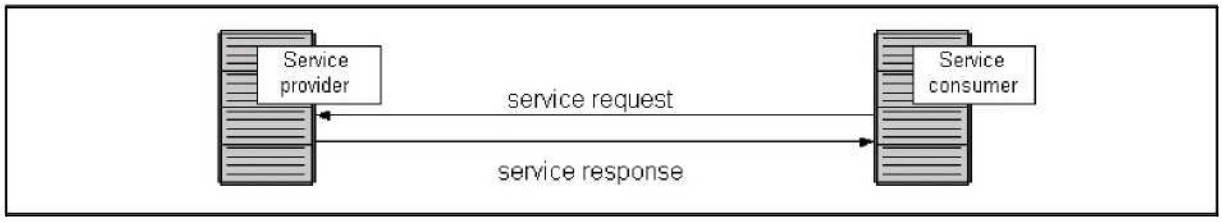

Figure 2.2: Simple request and response message exchange ([**SOA-Definition**]).

2.2 Web Services as Service Oriented Architectures

The concepts of Service Oriented Architectures for building better software applications grows with the well known term Web Service. So the software engineering concepts behind Web Service technologies play an important role for the creation of the standardized Service Oriented Architecture design. As already introduced in the last chapter e-business will be based on a Service Oriented Architecture in the future ([**Tao**]). As a consequence, e-business services can be seen as loosely coupled computing tasks communicating over the Internet. Every day new

strategies or ideas enter the Web as e-business services, such as computerized auctions or e-marketplaces. These emerging services must be based on shared highly dynamic organizing principles that constitute a SOA. The architectural concept behind the Web Services technology is the SOA, including how services are described and organized to support flexible or dynamic, automated discovery and use. Solutions for such loosely coupled computing systems are not new and the Web Service technology and the Service Oriented Architectures themselves were not created at once. In history, many solutions for distributed heterogenous environments were created and the term Middleware comes more and more in use for it. The basic ideas of Web Services and particular concepts of Service Oriented Architectures learned very much out of the Middleware approach, including benefits of Middleware and also their disadvantages. To understand the Web Service technology a historical background of such Middleware models and their approaches, ideas and concepts is necessary. This helps to understand better how the Web Service technology was derived and established over the years and why it is the technology of choice in the future.

2.2.1 Historical background of Service Oriented Architectures

Solutions for distributed operating systems and network operating systems established, over the years, some well known Middleware models for distributed communication. According to [**Distributed**] one of the earliest Middleware model is called Remote Procedure Call (RPC). In this model a process calls a procedure that is implemented on a remote machine. All parameters are transparently transferred to the remote machine that starts a remote process to generate the results of the procedure. Finally the results are sent back to the caller. RPC mimics a local procedure call, but actually the procedure is called remotely. Some loss of performance could only be realized if the network connection is not adequate or the execution on the remote machine needs too much time. Consuming information can be transported from the caller to the remote computer inside the parameters and back in the procedure results. There is no explicit message passing, but the problems of different address spaces causes complications. Information has to be passed to the remote machine and back, which can be complicated if the computers use a different representation of data. Packing data as parameters into a message is called marshalling. When all the parameters and results are standard types, such as integers, characters or booleans this model works fine. But if the client and server machine are not identical this model is unsuitable for large distributed environments, wherein multiple machine types are present. These problems are usually called *'interoperability problems'* of distributed environments. Some of these problems are listed in the following enumeration:

- problem of interoperable data formats and mutually understanding
- problem of different heterogenous hardware systems
- problem of usage of external resources outside the own memory area
- problem of the migration on new hardware or software platforms

Middleware was established to address these issues. A Middleware is a layer between the Operating System (OS) and the application. The key ideas of the Middleware model solves the above mentioned problems by hiding the heterogenous nature of different machines or operating systems as well as the distributed nature, using technologies to guarantee global access to resources. Furthermore, a Middleware uses uniform rules for data formats, protocols and interfaces for mutual understanding. Finally it supports the developers with code generators and specialized services, for instance a naming service. Note that a naming service can be used

to associate data or objects with more usual names, by which these objects can be found more easily.

The described RPC model can be seen as such a Middleware model, but it is not general applicable, because many of today's applications follow a distributed object-based approach. Instead of simple procedure calls, in distributed object-based systems everything is treated as an object and the servers offer services and resources in form of objects that they can invoke ([**Distributed**]). The benefits of objects here is that it is relatively easy to hide implementation details of distributed objects behind an object's interface, which can be seen as the access point to the remote object and its methods. There are two widely deployed distributed systems today the open standard Common Object Request Broker Architecture (CORBA) and Distributed Component Object Model (DCOM) from Microsoft. CORBA itself is not a real distributed system, it is more, as its name suggests, a specification or architecture of a distributed system. This specification was created by a nonprofit organization named Object Management Group (OMG). The primary goal was to define a distributed system that could overcome many of the *'interoperability problems'* of distributed environments with integrating networked applications. According to [**Middleware**], the OMG defines two standards the Object Management Architecture (OMA), a general platform description for creating distributed, object-oriented applications, and CORBA, which is a specialization of OMA for a concrete platform. The fundamentals of OMA are an abstract object model and the description of the relationships between objects in distributed environments. The core of this architecture is called Object Request Broker (ORB). The ORB acts as a bus for communication between Object services, Common Facilities, Domain and Application Interfaces as seen in Figure 2.3. More information, for example how these components work together, can be found in [**Middleware**]. However, the fundamental idea of CORBA is that the ORB is responsible for delivering the method calls to the correct objects, a process called *'dispatching'*.

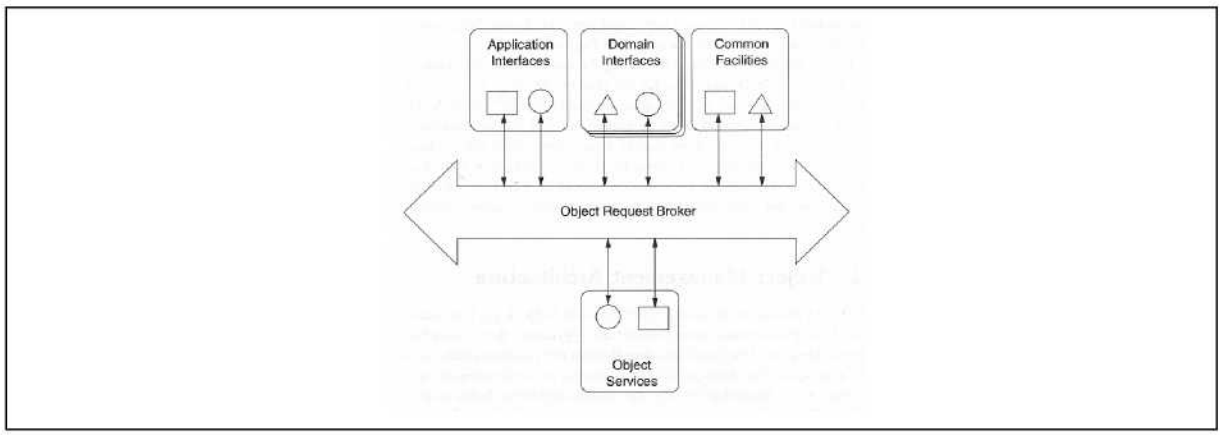

Figure 2.3: ORB as bus for communication in CORBA ([**Middleware**]).

A remote object is named a *'servant'* in CORBA. Communication aspects and its implementation are realized by client-sided *'stubs'* and server-sided *'skeletons'*. These two are also called *'proxies'* and could be generated automatically from an interface, which is described using the Interface Description Language (IDL). These proxies take care of getting object references, method names, parameters and results in a serialized form that is sent over the network. This process is called *'marshalling'* and its counterpart process, the restore of object references from the serialized form is called *'unmarshalling'*.

The IDL is used for the specification of interfaces for objects in a *'programming language neutral'* way. The differences between programming languages in a distributed object-based system is also an *'interoperability problem'*, so the idea of a special, declarative language is used to solve this problem. IDL contains no algorithmic details, like branches or loops, it is only for describing the interface methods and the corresponding parameters. There are many platform dependent *'IDL Compilers'* available, which generate platform specific code based on the descriptions inside the IDL interface. To handle the interoperability of different CORBA products, all created in different programming languages, [**Corba**] defines an Internet Inter-ORB Protocol (IIOP). IIOP is an interoperability framework that includes technical details like addressing of communication-endpoints and which uses the Transmission Control Protocol (TCP) for transportation. To summarize, the goals of CORBA are:

- object-oriented design,
- transparency of the distributed nature, and
- hardware-, system- and language independency.

Note that CORBA is also an open standard, so the architecture is supposed to be vendor independent. Experiences over the years have shown that CORBA is a useful technology, even though there are often troubles with firewalls ([**GT3-Tutorial**]). But important to know is that CORBA offers programmers a lot of supporting services, for instance persistency, notifications, lifecycle management or transactions.

The second object-based distributed system is the proprietary DCOM technology created by Microsoft. DCOM is based on the Component Object Model (COM) that is the underlying technology of various Microsoft Windows operating systems since Windows 95. DCOM is widely used, because millions of people using Windows daily in network environments. The COM technology stands for development of components that can be dynamically activated and interact with each other, realized by dynamically linked libraries or executable programs. The advanced DCOM technology adds the possibility to communicate with remote components placed on another machine. A DCOM object is an implementation of well-defined interfaces. In contrast to CORBA, DCOM supports only binary interfaces that are defined in Microsoft Interface Description Language (MIDL), a language corresponding to CORBAs IDL. Such binary interfaces are essentially a table with pointers to the implementations of the related methods and are therefore *'programming language independent'*. There is no standardization necessary for the mapping of IDL specifications to a new supported programming language as in CORBA. Figure 2.4 shows the differences more precisely.

It is important to know that in reverse to CORBA, all DCOM objects are transient and automatically destroyed when there are no more references on it. Another difference to CORBA is that DCOM is not a complete distributed system, because it assumes the existence of external services, for example a naming service, which is realized through the Windows Active Directory ([**Distributed**]). Remember that a naming service is a software that manages a naming system that provides a natural and for humans understandable way of identifying and associating defined names with any kind of data. For instance, the Disk Operating System (DOS) file system uses a naming system to associate any kind of data with human readable file names and hierarchically folder structures. Thus a naming system gives humans an easier way to interact with complex addresses inside the computer by associating any kind of data with understandable names. However, this need for a naming service limit portability, another item of

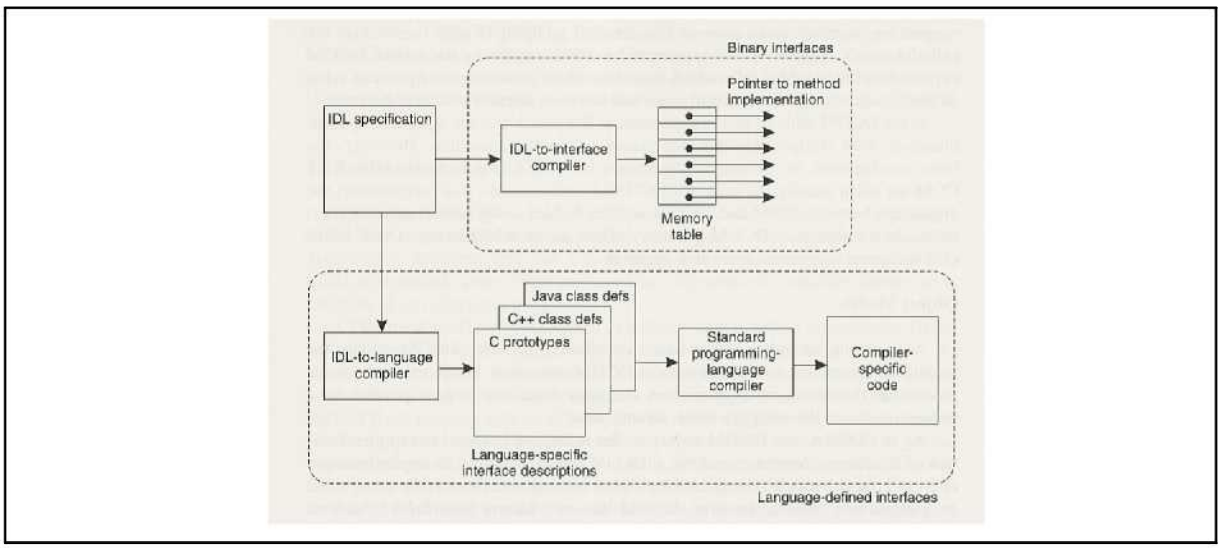

Figure 2.4: The difference between language-defined and binary interfaces ([**Distributed**]).

the *'interoperability problems'* in distributed environments when using DCOM objects in a Uniplexed Information and Computing System (UNIX) environment, because they cannot make use of the same naming services as offered in a Windows environment. But note that virtually all DCOM applications are running in Windows environments.

2.2.2 Key Concepts of Web Services

In a simplified view Web Services are another Middleware technology like CORBA, or DCOM. Web Services are based on standard Web technologies to create client/server applications for heterogenous distributed environments. Many organizations rely on Web Services for their enterprise applications today. Note that standards and related technologies are defined by the World Wide Web Consortium (W3C). As a very simple example of a Web Service, imagine an organization that has branch offices all around the world. The master database is only available at the central office at one particular location and this database needs to be accessed by all other branch offices. A Web Services approach for accessing the database does not imply publishing data on a Web site using HyperText Markup Language (HTML), Extensible HyperText Markup Language (XHTML) or some related technology. Instead well-defined, not human readable documents are generated and made available on the Web. Note that information which is available through a Web Service will always be accessed by software and never directly by a human. The human is using this software, but he don't know that the application uses Web Services to retrieve remote pieces of information. Even though Web Services rely heavily on existing Web technologies, like XML, HTML or HTTP, they have no direct relation to Web browsers. While Web-sites are human readable, Web Services are software oriented. In the simple example all the branch offices would contact the Web Service and send a service request to the database at the central office. The central office would return a service response, including all the requested data. Figure 2.5 shows this very simple example.

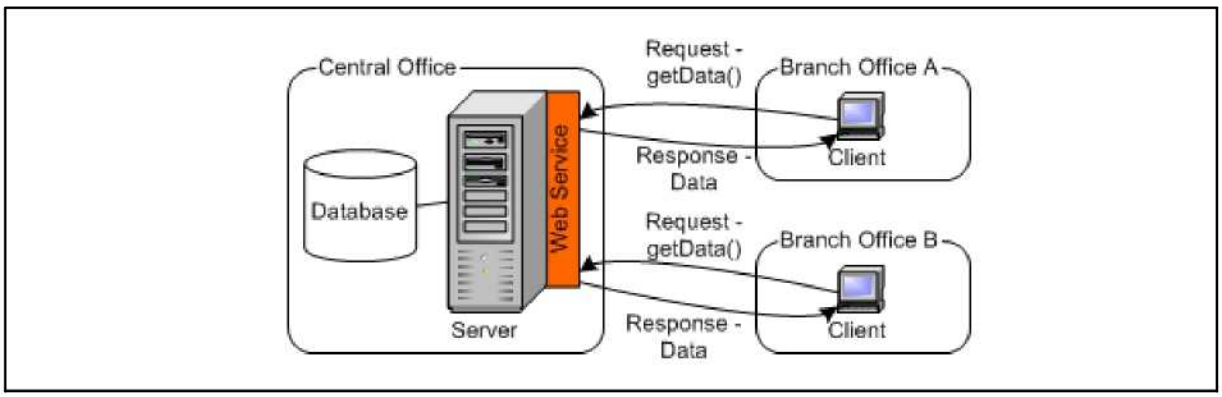

Figure 2.5: Database access through a Web Service.

While the main concepts of Web Services are intuitive, the knowledge how Web Service-based applications are structured is not more difficult. Here background knowledge of technologies like CORBA or DCOM is important, because the structure looks pretty familiar. First of all, there are many protocols and techniques used by Web Services. To concentrate on the core idea of Web Services, underlying protocols and techniques are explained later in detail. For now, it is important to list the core building blocks of Web Services:

- Hypertext Transfer Protocol (HTTP) is usually used for transport
  (but also the Simple Mail Transfer Protocol (SMTP) or other transport protocols can be used)
- Simple Object Access Protocol (SOAP) is used for service invocation
- Web Service Description Language (WSDL) is used for service description
- Universal Description, Discovery and Integration (UDDI) is used for service discovery

Usually a Web Service developer do not need to write whole Web Services Description Language (WSDL) interface descriptions nor all details for the use of Simple Object Access Protocol (SOAP). That means when a client application needs to invoke a Web Service we can delegate this task on a part of software called a client stub ([**GT3-Tutorial**]). Like in CORBA or other RPC technologies there are many of tools available that automatically generate client stubs, usually based on the WSDL description of the Web Service. Remember that similar tools were also used in CORBA to create stubs based on IDL description files of a remote interface.

There are four steps for a typical invocation of a Web Service at the client side. At first, a Web Service that meets the requirements is located through UDDI. After that the Web Service's WSDL description has to be obtained. The description is used to generate the client stubs once and to include the stubs in the applications. Here they are used each time the application needs to invoke the Web Service. The programming of the service side Web Services itself is relatively easy. The server program is not necessary a complex program that dynamically interprets SOAP requests and generates complicated SOAP responses. The server side developer can concentrate on the functionality of the Web Service itself and rely on tools to generate a server stub. Like within CORBA, this stub is also named skeleton and is in charge of interpreting requests and forwarding them to the service implementation created by the developer. So the concept beyond a stub is that when the service implementation computes a result it will hand it over to the server stub that generates the appropriate SOAP response. It is important to know that the service implementation and the server stubs are managed by a special software called

hosting environment or Web Services container. This hosting environment will make sure that incoming HTTP requests intended for a special Web Service are directed to the related server stub. Figure 2.6 illustrates this procedure.

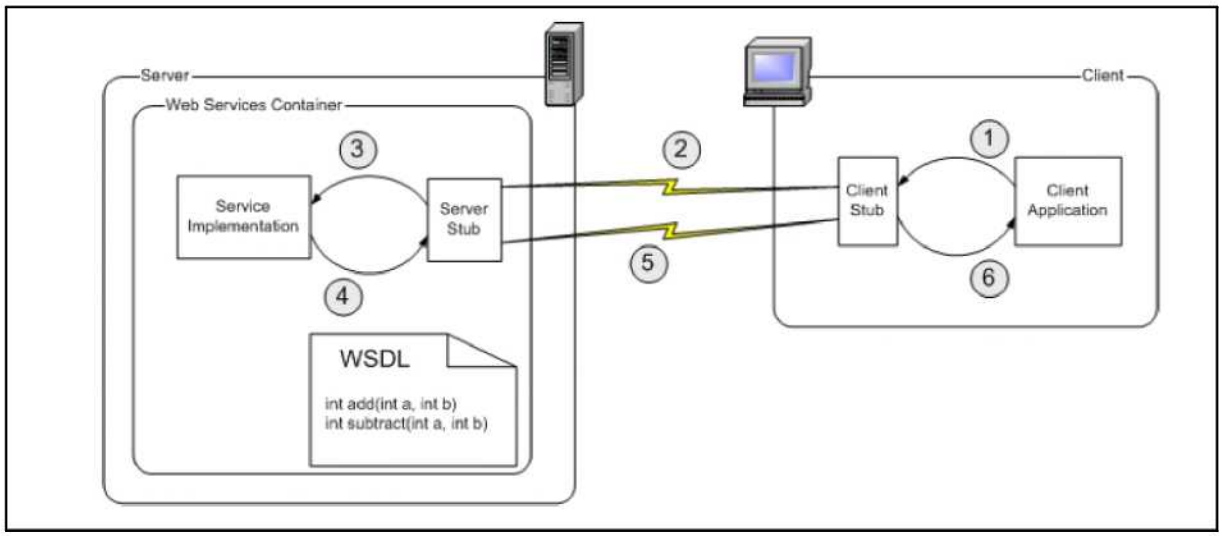

Figure 2.6: A typical Web Service invocation using stubs ([**GT3-Tutorial**]).

The work-flow in Figure 2.6 supposes that the Web Service is already located using UDDI and that all the stubs have already been generated:

1. The client application always uses the client stub to invoke the Web Service. This stub turns the invocation into a proper SOAP request. This process is called the *'marshalling process'*.

2. The created SOAP request is sent over the Internet using vanilla HTTP. The hosting environment or Web Service container receives the SOAP request, interprets its receive and hands it to the correct server stub that converts the request into a service specific format. This process is called the *'unmarshalling process'*.

3. When the service implementation receives the request from the server stub, it will immediately start the associated computing process, for instance the `int add(int a, int b)` method described in the WSDL description.

4. The server stub receives the result of the method and converts the results into a SOAP response.

5. This SOAP response is sent over the Internet back to the client stub that extracts the application specific format.

6. The client stubs forwards the response to the client application that can now work with the results of the Web Service method.

There are various other Web Service standards defined, for instance the Web Services Flow Language (WSFL) that can be used for Web Services orchestration ([**WSFL**]). Orchestration means the building of sophisticated Web Services by composing simpler Web Services. There is also a Web Services Inspection Language (WSIL) defined in [**WS-Inspection**] that can be used to create documents containing a collection of service descriptions and links to other sources

of service descriptions. Here, service descriptions can be a Uniform Resource Locator (URL) ([**RFC1738**]) to a WSDL document. WS-Inspection enables the service provider to create a WSIL document and to publish this document on the network to advertise in a central document. So service requestors can easily retrieve this WSIL document and can discover what services the service provider advertises ([**Physiology**]). Even if there are many of standards keep in mind that WSDL and UDDI are the spine of Web Services ([**Java-XML**]).

The illustrated process of a remote invocation can be also realized by old CORBA or DCOM technologies. So one central question is what makes Web Services so special? They have many advantages over CORBA or DCOM, but Web Services have also some disadvantages. This section provides a short summary of advantages and disadvantages of Web Services. However, note that it is a relatively new technology and there are still many issues open to the Web Services architecture.

Advantages of Web Services

First and foremost, Web Services are solving many of the *'interoperability problems'* that were discussed in the section about CORBA and other technologies for distributed systems. The Web Services paradigm differs from other approaches such as CORBA or DCOM in its focus on simple, Internet-based standards to address present heterogenous distributed environments ([**Physiology**]). The use of XML standard enables Web Services as platform-independent and language-independent solution. Hence a client program can be in C++ within some Windows environment while the requested Web Service is implemented in Java running under Linux. The use of the standard language WSDL makes Web Services self-describing, another advance of this technology. Furthermore, Web Services are typically using HTTP for transmitting their service requests and responses. Like a Web browser, Web Services can be used without any problems related to firewalls unlike the CORBA technology that has often trouble with firewalls.

There is also one important characteristic of Web Services that can be seen as an advantage for heterogenous distributed systems: Web Services are oriented towards loosely coupled systems where the client is not required to have knowledge of the Web Service until the client invokes it. Technologies like CORBA or DCOM, on the other hand, are oriented towards more tightly coupled systems existing in Intranet applications, but perform poorly on an Internet scale ([**GT3-Tutorial**]). To create Internet applications Web Services are better suited for this. Also the problem of different address spaces is solved by passing the parameters to the remote machine. And the solution for this problem is solved by using XML as encoding standard. This established standard makes RPC that is the basic idea beyond Web Services, again suitable for large distributed environments with multiple machine types. Therefore modern Web Services are also called *'XML-based RPC'*. Another advantage of Web Services is the use of WSDL. WSDL provides a standard mechanism for defining interface definitions separately from their embodiment within a particular binding ([**Physiology**]). That means that bindings for transport protocol and data encoding formats can be changes without to changed without being required to update the implementation.

To summarize, Web Services have the following advantages:

- platform-independent
- language-independent
- self-describing
- portability
- coexistence with firewalls
- supporting loosely coupled distributed systems
- separation of interface definitions from their embodiment

Disadvantages of Web Services

Even if there are many advantages of the Web Service technology there are always some disadvantages connected with them. Web Services are extremely associated with XML for transmission and this is not as efficient as the use of proprietary binary code. While portability is gained, efficiency is lost. This is called the *'overhead'* of Web Services transmission, but this overhead is usually acceptable for the most applications. Whether critical real-time applications can use Web Services as Middleware is still open. Another disadvantage is the lack of versatility, because Web Services only allow basic forms of service invocation. In plain Web Services there are no supporting services, such as persistency, notifications or lifecycle management, like in CORBA. To summarize the disadvantages of Web Services in a very short way:

- data conversion overhead
- lack of versatility in the context of supporting services

2.2.3 Overview of used protocols and techniques

This section provides an overview of the used protocols and techniques that are used by the Web Service technology.

Hypertext Transfer Protocol (HTTP)

HTTP ([**RFC2616**]) is used for transportation of messages from the client to the server and vice versa. Therefore Web Services are using the same protocol used to access conventional Web pages on the Internet. In theory there could be also other protocols, but HTTP is the most used one ([**GT3-Tutorial**]).

Simple Object Access Protocol (SOAP)

The SOAP protocol is specified as the backbone to a new generation of cross-platform cross-language distributed computing applications, termed Web Services ([**SOAP**]). The SOAP protocol is practically XML based RPC and is used for service invocation. Service invocation in distributed systems requires the passing of messages between client and server. In fact, this is not special to Web Services, because any kind of distributed technologies such as CORBA are using messages to communicate over the Internet or Intranet. But more special to Web Services is the use of SOAP that specifies the format of the request to the server and the format

of the response to the client. In fact, SOAP is by far the most popular choice for Web Services ([**GT3-Tutorial**]), because it is a simple enveloping mechanism for XML payloads that defines a RPC convention and a messaging convention. SOAP payloads can be carried on HTTP, File Transfer Protocol (FTP) or Java Messaging Service (JMS), so SOAP is independent of the underlying transport protocol ([**Physiology**]). Figure 2.7 shows the basic SOAP structure used by the enveloping mechanism.

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap=
    "http://www.w3.org/
      2001/12/soap-envelope"
  soap:encodingStyle=
    "http://www.w3.org/
      2001/12/soap-encoding">

  <soap:Header>
    ...
  </soap:Header>

  <soap:Body>
    ...
    <soap:Fault>
      ...
    </soap:Fault>
  </soap:Body>

</soap:Envelope>
```

Figure 2.7: Basic structure of a SOAP message.

The `<soap:Envelope>` element defines the XML document as a SOAP message. The optional `<soap:Header>` element contains application specific information about the SOAP message, e.g. authentication or payment details. The actual SOAP message can be found inside the required `<soap:Body>` element. To conclude, SOAP allows applications to communicate language-neutral via standard HTTP.

Web Services Description Language (WSDL)

The idea of the IDL in the context of the CORBA technology was migrated into a new standard language called WSDL. This language is used for service description to establish self-describing Web Services. As a consequence, any Web Service can describe itself inside a WSDL document, including input parameters and output parameters of the service's methods and how to invoke them. WSDL files are XML documents for describing Web Services as a set of endpoints operating on messages containing either document-oriented, also called messaging, or RPC payloads ([**Physiology**]). The whole WSDL file defines abstractly an service interface in terms of message structures and sequences of operations. Important is that these operations are bound to a concrete network protocol and data-encoding formats to build an complete endpoint. Finally, all endpoints are bundled to define abstract endpoints, called services. Furthermore, WSDL allows a description of endpoints for different message formats and network protocols together with standardized binding conventions. These binding conventions describe the use of WSDL in conjunction with SOAP, HTTP `GET/POST` or Multipurpose Internet Mail Extensions (MIME) ([**Physiology**]). According to [**WSDL 1.1**], services are defined using the following six major elements:

- `service`
  represents the service that is aggregated by a set of related ports.
- `port`
  specifies an address for a binding to establish an communication endpoint.
- `binding`
  provides a protocol-specific binding and data format specifications for operations defined by a particular portType.
- `portType`
  represents a set of abstract operations while each operation refers to an input message and output messages.
- `message`
  represents an abstract definition of the data being transmitted, including input and output messages.
- `types`
  provides data type definitions that can be used to describe a message exchange.

Universal Description, Discovery and Integration (UDDI)

Web Services are an advantageous technology that can be used to compute remote jobs. An usual problem is to find the service that provides an appropriate function at a particular time. But the most relevant question is, where can this Web Service be found, when the requestor does not know where to start searching. However, this start of searching can be done using UDDI. UDDI supports locating Web Services that meet certain requirements. So UDDI techniques can be used for service discovery.

2.3 Grid Fundamentals

This section provides a brief overview about Grid computing and focuses on its use by applications. Grid computing represents unlimited opportunities in terms of business and technical aspects ([**Grid-Fundamentals**]).

2.3.1 Ideas beyond Distributed Computing

The World Wide Web (WWW) can be viewed as a huge distributed system consisting of millions of different computers accessing very much linked documents using a common standardized protocol. The Internet explosion created over the years many efforts that address the demand for standards of communications between heterogeneous distributed systems. A distributed system is meant as a collection of independent computers that appears to its users as a single coherent system ([**Distributed**]). The most important fact is that differences between the various architectures and the ways in which they communicate, also known as *'problems of heterogenous environments'*, are hidden from the end user. Distributed systems should be easy to expand or scale and should be continously available, even if some elements of the system are temporarily out of order. Therefore, they are organized using Middleware to support heterogeneous computers and networks and establish a standard access environment for higher level distributed applications. Distributed systems are usually realized by high-speed computer networks using the standardizes protocols TCP or the Internet Protocol (IP). Such distributed systems allow distributed computing.

2.3.2 Grid Computing

According to ([**Grid-Fundamentals**]), Grid computing is distributed computing taken to the next evolutionary level. The basic idea of a Grid is to create the illusion of a simple large and powerful self-managing virtual computer. The virtual computer consists of a large collection of connected heterogeneous systems sharing various combinations of resources. A Grid provides the electronic underpinning for a global society in:

- business
- government
- research
- science
- entertainment

Therefore the Grid can be seen as a computing and data managing infrastructure that is used in different usage scenarios in different global societies. The Grid provides a virtual platform for computation and data management in the same way that the Internet integrates resources to form a virtual platform for information ([**GRID-Computing**]). A Grid integrates:

- networking
- communication
- computation
- provision of information

To conclude, Grids are intrinsically distributed, heterogenous and dynamic. Grids realizing effectively infinite storage and access to instruments such as visualization devices without regard to geographic location. A Grid infrastructure will connect multiple regional and national computational Grids, creating a universal source of computing power. Note that the term 'Grid' is chosen by analogy with the electric power grid that provides pervasive access to power and, like the computer and a small number of other advances, has had a dramatic impact on human capabilities and society. By providing pervasive, dependable and inexpensive access to advanced computational capabilities and databases, computational Grids will have a similar transforming effect, allowing new applications to emerge ([**GRID-Blueprint**]).

2.3.3 Grid Applications

According to [**GRID-Blueprint**], there are five major application classes for computational Grids. This section provides a brief overview on the different application classes. While Grid developer still have to face the absence of a mature grid infrastructure there are many already successful applications in use today, which are also briefly mentioned as examples. Note that all applications classes have in common a tremendous appetite for computational resources.

Distributed Supercomputing

Some problems cannot been addressed using a single system. So distributed supercomputing uses the Grid infrastructure to aggregate substantial computational resources to solve such problems. Depending on the Grid infrastructure available, these resources might comprise the majority of the supercomputers in the country or simply all of the workstations within a company ([**GRID-Blueprint**]). To give a real application example, Distributed Interactive Simulation (DIS) is a technique, which is used for training and planning in the military. Imagine, realistic scenarios with thousands of entities, each with potentially complex behavior patterns. As described in [**GRID-Blueprint**] even the largest supercomputer can handle at most 20.000 entities, like troops or materials, but when they are coupled like a Grid these multiple supercomputers can achieve record-breaking performance. However, not only military can have a use of distributed supercomputing. Such coupled supercomputers can also used to resolve fine-scale resolution details of accurate simulations of complex physical processes. This ends up in new scientific results, for instance in cosmology, chemistry, physics or in climate modeling. Therefore the Grid infrastructure coschedules scarce and expensive resources. Also the scalability of protocols and algorithms to thousands of nodes are relevant in these applications to achieve high levels of performance across heterogenous systems within Grids.

High-Throughput Computing

Many computers all over the world are frequently unused. High-throughput computing aims for using the Grid infrastructure to schedule large numbers of loosely coupled independent tasks to unused computers. The independent nature of the tasks leads to different types of problems and problem-solving methods within Grids ([**GRID-Blueprint**]). For example, the Condor system from the University of Wisconsin is used to manage pools of hundreds of workstations at universities and laboratories around the world. Condor identifies idle workstations and schedules background jobs on them. Furthermore, if the owner of a workstation resumes activity, Condor checkpoints the remote job running on the station and transfers it to another workstation ([**Condor**]). The results were new virtual developed resources that were used for studies such as molecular simulation of liquid crystals or design modeling of diesel engines.

Access to thousands of computers distributed worldwide can also be used to tackle hard cryptographic problems.

On-Demand Computing

The third application class for computational Grids is called on-demand computing. These applications use Grid capabilities to meet short-term requirements for resources that are not conveniently located locally and not usually cost-effectively. Such resources are computation, software, data repositories or specialized sensors. In contrast to distributed supercomputing applications, these applications are often driven by cost-performance concerns rather than absolute performance ([**GRID-Blueprint**]). An example is the computer-enhanced Magnetic Resonance Imaging (MRI) scanner and Scanning Tunneling Microscope (STM), an improved medical device technology, such as the one developed at the National Center for Supercomputing to achieve real-time image processing of detailed images of the body's organs and structures without the use of X-rays or other radiation. However, these resources support a significant enhancement in the ability to understand what we are seeing. Another example for on-demand computing is a system that is developed at the Aerospace Corporation for processing of data from meteorological satellites. This system uses the on-demand approach to deliver results of cloud detection algorithms to remote meteorologists in approximately real time. This is achieved by the dynamically acquiring of supercomputer resources. So on-demand computing addresses the dynamic nature of resource requirements and the potentially large populations of users and resources within Grids, including resource location, scheduling, fault tolerance, security, and payment mechanisms ([**GRID-Blueprint**]).

Data-Intensive Computing

Data-intensive computing applications are focused on synthesizing new information from data that is maintained in geographically distributed repositories, digital libraries and databases ([**GRID-Blueprint**]). These synthesizing processes are often very computationally and communication intensive.

The mostly known example is the high-energy physic and their experiments that will generate terabytes of data per day in the future leading to one petabyte per year. The computationally intensive data analysis are working on large fractions of data to detect any interesting new events. Furthermore, the data systems, in which parts of the data is stored, are widely distributed as well as the scientists that are interested in accessing the data ([**GRID-Blueprint**]). To conclude, data-intensive computing addresses complex, high-volume data flows through Grids and the use of this data.

Collaborative Computing

The last application class for computational Grids is collaborative computing. Collaborative Computing is concerned with enabling and enhancing human-to-human interactions. Such applications are often structured in terms of a virtual shared space. As a consequence many of these collaborative applications enabling the shared use of computational resources such as data archives and simulations ([**GRID-Blueprint**]). An example for such human-to-human interactions is the Boilermaker system, which is developed at the Argonne National Laboratory. The system allows multiple users to collaborate on the design of emission control systems in industrial incinerators within a simulation of such an incinerator.

Another really nice example is the Narrative Immersive Constructionist/Collaborative Environments (NICE) system developed at the University of Illinois at Chicago. With the help of this system, children can participate in the creation and maintenance of realistic virtual worlds ([**NICE-Home**]). So beside entertainment, industry or education the scope of collaborative applications helps to integrate the human perceptual capabilities into a rich variety of interactions in Grids.

2.4 An Open Grid Services Architecture

This section describes an Open Grid Services Architecture (OGSA) as introduced in [**Physiology**] that defines a concept called Grid Services. The OGSA is based on Web Service technologies. The aim of this section is to get the knowledge how Grid Services are an extension of Web Services and how they can be used in a Grid environment. The goals of OGSA are to define a commonly used and a well-designed architecture for applications in a Grid environment.

2.4.1 Grid Technologies aligned with Web Services technologies

The situation for application developers and the target environments for their implemented applications has changed in the last years. In former times, applications were created for mostly homogenous, reliable, secure and centrally managed environments. Today, computing is concerned with collaboration, data sharing and other new modes of interaction. These emerging scenarios involve distributed resources, which changes the focus on interconnection of systems both within and across enterprises. Companies have realized that there is a potential for reducing IT costs by outsourcing nonessential parts of their IT infrastructures to various forms of service providers. This leads to decentralized and distributed software and hardware ([**Physiology**]). New concepts are achieved that allow applications to access and share resources and business logic, wrapped as services, across distributed, enterprise-overlapped, wide area networks. This sharing is not only simple file exchange. Instead it includes direct access to computers, software, data and resources leading to a collaborative problem-solving environment for large-scale science and engineering. These scenarios led to the development of Grid Technologies that are designed to support the sharing and the coordinated use of diverse resources in dynamic, distributed Virtual Organizations (VO). VOs are meant as a collective name of the Grid problem - *'the flexible , secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resources'* as defined in [**Anatomy**]. The OGSA introduces a framework that describes how Grid functions can be realized and what types of technologies they should based on. Certainly, OGSA does not give a technical and detailed specification that would be necessary to implement solutions for Virtual Organizations (VO). The more technical specification was done within the Open Grid Services Infrastructure (OGSI) as specified in [**OGSI-Spec**]. OGSA itself focuses on the nature of the services and related protocols that were introduced in [**Anatomy**], required for interoperability among VOs. The OGSA explains how Grid Technologies can be aligned with the following Web Service Technologies:

- Service description and discovery
- Automatic generation of client and server code from service descriptions
- Binding of service descriptions to interoperable network protocols

Web Services tools and its broad commercial support is used as the basic architecture that is aligned with Grid Technologies to create an widely accepted OGSA. Finally, OGSA is a well-defined set of basic interfaces to create open systems with loosely coupled clients and servers in a service-style, including communication extensibility and vendor neutrality.

2.4.2 The Ideas beyond Grid Services

Web Services are the favored technology for Internet-based applications with loosely coupled clients and servers and that's why OGSA defines Grid Services that are basically Web Services with improved properties and advanced service features. These improvements are necessary, because plain Web Services have the following limitations:

- stateless
- non-transient

The statelessness means that plain Web Services can not store any kind of data for using information created by some former invocation of that service. Very simplified, the invocation of an operation is done in a temporary environment. After the results are sent back all information is lost. A chain of remote operations were a subsequent invocation call needs the results of the last invocation calls is only possible by sending the result values again as parameters of the invocation. This is not necessary net traffic, because the client could not do anything with the result values of parts of the chain of operations. The Grid Service approach includes the server-sided storage of parameters to avoid such net traffic. That approach allows mobility of the service, because each invocation of the service needs only the really demanded parameters. Interim or temporary results could be stored in some way at the service provider. Hence, this leads to a lightweight client design that can also be used for complicated chain operations with many parameters. However, some Web Services containers actually work around this stateless problem ([**GT3-Tutorial**]) and in industry very much workarounds can be found for it, but plain Web Services are usually stateless services.

The second limitation, non-transient Web Services, is easy to understand in an example. Supposing there is a Web Service which computes a long time on special values and stores many temporary results inside variables. When one client invoke the Web Service another client could invoke any time this Web Service also and if the first call is not finished yet, the second call could mess up the first invocation by storing different values to the variables. So non-transient Web Services outlive all their clients or service invokers. This means that values from one invocation of the Web Service is still present in the next invocation of the same Web Service ([**GT3-Tutorial**]). This could be the solution for the stateless problem, but remember that any client could invoke the Web Service at any time, so it is not sure that the complete chain of operations could be invoked successively from the same client. Maybe after an incompleted set of invocations some other client invoke the Web Service and thus interfere with the chain operations, resulting into broken computation. So the demand for creating a stateful Web Service exists. In such a stateful Web Service every invoke call might start for example an extra Thread, also called Thread Pooling, or might create some kind of new instance of the Web Service.

The proposed solution is a factory approach for Web Services as defined in [**Physiology**]. That means, the client invokes some kind of create call addressed to a factory service that returns a unique instance of the wanted Web Service used to interact with. Note that the factory service is also a Web Service! To solve the non-transient problem, the OGSA defines a limited lifetime

for the instances of the Web Service. The lifetime is limited, but varied from application to application. One approach to managing lifetime is that an instance only lives as long as the invoking client has any use for it. There is also another approach that defines the self-destruction of an instance when no interaction with the instance was performed for a certain time. This factory approach as shown in Figure 2.8 is the most significant improvement offered by Grid Services. The factory creates for every client its own personal stateful instance that allow an exclusive modifying of its state. But there could still be the possibility that many clients work with one instance if this is required. With this whole functionality the lack of versatility of plain Web Services is fixed with Grid Services. Additionally there are a lot of supporting services for Grid Services, such as persistency, notifications or lifecycle management.

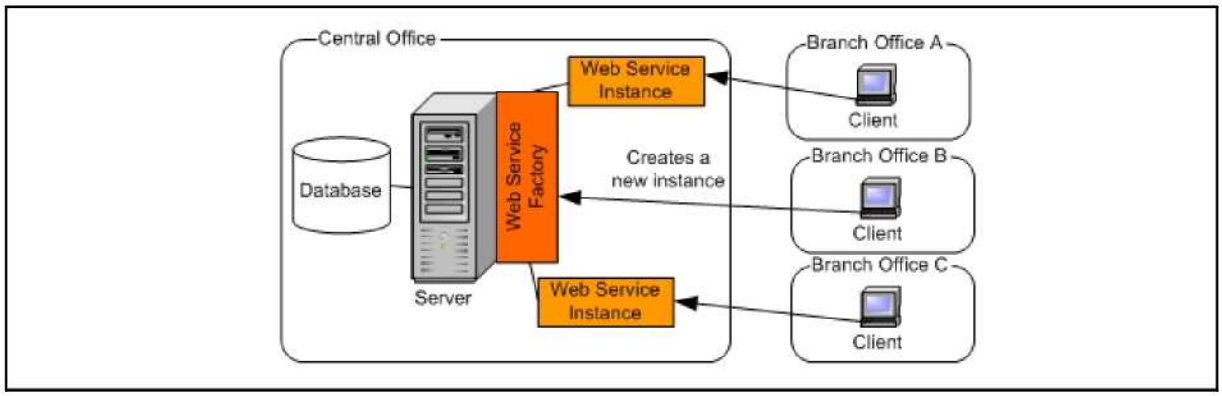

Figure 2.8: The factory approach for Web Services.

However, OGSA also defines further improvements that Grid Services have. A complete lifecycle management is introduced in which Grid Services provide callback functions during special moments in a its lifetime. This can be important during the creation and destruction of an instance in the context of persistent instances. So the callback function for the creation of an instance can be used to retrieve specific values out of a database for example, wherein all properties representing the state of the instance could be easily stored and retrieved. Imagine that a callback function during the destruction of an instance could be used to store the relevant properties to the database. Figure 2.9 depicts this way of using stateful persistent instances. As a side-remark, this is an example of the kinship of all technologies for distributed environments. In CORBA for instance, developers use the Servant-Manager hooks incarnate and etherealize to store and retrieve the state of an individual CORBA object.

A further improvement concerns the discovering of available Grid Services and a way of determining the characteristics of those services. As a consequence, services can configure themselves and their requests to other services appropriately ([**Physiology**]). So Grid Services can have a set of associated data that describes the Grid Service instances in details, structured as a set of named and typed XML elements. These service data is no part of the interface description in WSDL that describes methods, protocols and technical details. Service data is used to index Grid Services according to their characteristics and capabilities. When a Grid Service is deployed, other Grid Services or remote Clients can discover and invoke the service using its published interface. Therefore, Grid Services are self-contained and self-describing applications providing functionality that can be published, located and invoked over the Internet.

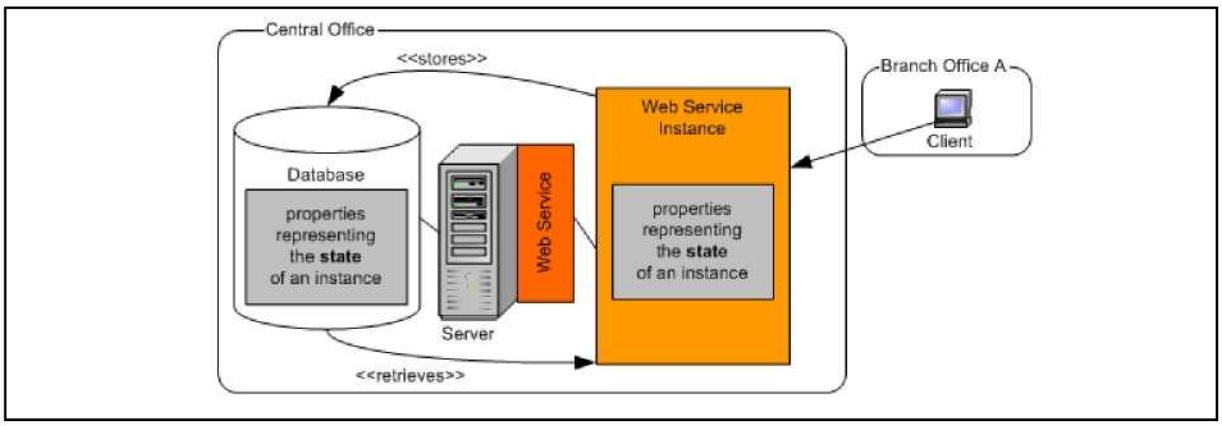

Figure 2.9: Stateful persistent instances by using a database.

To conclude, the term Grid Service denotes a particular service offered to clients following the set of rules defined in Grid environments. It is essential to know that the OGSA only defines a Grid Service. The detailed specification is left to OGSI. An OGSI compliant Grid Service is therefore also an OGSA compliant Grid Service. Basically, Grid Services are just an extension of Web Services and their widely used standard interoperable technologies like XML, WSDL and SOAP. Note that the motivation for the improvement of Web Services is to adapt Web Services to the requirements typically found in a Grid environment ([**GT3-Core**]). Grid Services can perform a wide range of functions including simple resource requests or complicated business and scientific computing. For the realization of flexible service offers in a Grid environment Grid Services are the best choice for now.

2.5 Languages for Service Characteristics

The first section motivates the need for a language that can be used to specify QoS description terms and service characteristics. Service characteristics can be unavoidable pieces of information to create a service, for instance start-time or end-time of the service or many other pieces of information needed for accessing computational resources or storage capabilities. While many languages exist for submitting computational jobs to resources, potentially through a batch system, most of them are not explicitly XML based. [**JSDL-Wg-Home**] enumerates the following interoperable languages:

- EU-DataGrid's Job Description Language (JDL)
- Condor's ClassAds
- Globus's Resource Specification Language (RSL)

For some of these languages are incompatible XML equivalents provided that are inhibiting interoperability between those language and their systems. The intention of this section is to identify a suitable XML based language, following the argument towards standards. There should be a definition of an abstract standard language and a normative XML schema ([**XML-Schema**]) of this language, to ensure compatibility. The Web Service Level Agreement (WSLA) language is analyzed to identify if this language could be used for that. Apart from that other languages are also respected such as the Job Submission Description Language (JSDL) language or the Web Ontology Language of the W3C.

2.5.1 The Need for Quality of Service Description Terms

Flexible service offers in a Grid Environment, realized by Grid Services offer support for negotiation capabilities. To implement such services a special language is needed. This language should specify the following parts in a formal way:

- Functional service description terms, for instance information needed to address the parties involved in the communication, and
- Non-functional service description terms, e.g. QoS issues.
- Additionally domain-specific information needed to create a service.

Since, Grid Services are basically improved Web Services, already existing service description languages for Web Services are relevant for Grid Services. The purpose of service description terms is to identify and to describe parameters of a SLA and to which service they relate to. Note that it is not the aim to describe the service in a technical way like WSDL. SLA parameters in Grid environments are mostly equivalent to the SLA parameters in classic distributed systems. In addition, the information on how to measure SLA parameters is relevant for service description. Monitoring services can be automatically configured to monitor the SLA, based upon the detailed specifications inside these SLA specifications. However, this monitors can also be configured for respecting domain-specific situations, because different QoS parameters require different domain-specific implementation strategies and monitoring ([**WS-QOS**]). This automatic approach reduces the need to costly slow and error-prune manual intervention to a minimum and, in fact, this becomes increasingly important for emerging complex SOAs, such as Web Services ([**WSLA-Report**]). Furthermore, a Web Service agreement language is used to define the obligations of the parties involved in an agreement. Those obligations are sometimes also called SLA parameters formulated in a special language to specify formally clear SLA terms. As a consequence the SLA can be monitored by service providers, customers or even by third-parties. Those SLA terms or parameters can be viewed as service characteristics that are respected by a service provider to perform a service according to agreed-upon guarantees for IT-level service parameters such as availability, response time and throughput for Web Services ([**WSLA-Spec**]). Note that there are also some description terms that are not necessarily connected with QoS issues. For example the start- and end-time of the service itself. So this SLA parameters and non QoS description terms can be seen as service characteristics.

The benefit of non proprietary languages for service characteristics and agreement terms is that they are focusing on the parameters that are of common interest of all the involved parties. Therefore all participants of the agreement have various degrees of freedom to create an implementation policy for a service and its surveillance realizations. Beside surveillance of services other service level management issus can be covered by a clear and formal representation of an SLA, like the specification of measures to be taken in case of deviation and failure ([**WSLA-Spec**]). The detailed definition of service parameters, including the way of how metrics should be measured in systems and how they aggregated into composite metrics and SLA parameters, are a necessary information for service providers to setup and perform services correctly. There are maybe also third parties, for example a special management service providers, that take over all the measurements of metrics and surveillance of guarantees specified in the agreement. There must be a language that not only service providers and service consumers can understand. Also third parties must be minded when there are many interactions among parties supervising the agreement terms. According to [**WSLA-Spec**] those interactions among parties are also known as *'multi-party constellations'*.

A language for the representation of terms and their interrelationships is often also called an ontology, a notion which becomes more and more common. In the context of an ontology and in a more abstract view from all described above, the future of the Web, wherein information is given explicit meaning, is named the *'Semantic Web'* ([**OWL-Overview**]). It's like a vision for making it easier for machines to automatically process and integrate pieces of information that are available on the Web. This goal is reached by using standards like XML to define customizing tagging schemes or the Resource Description Framework (RDF) to use its flexible approach to representing data. According to [**RDF**] the framework is designed to represent information in a minimally constraining, flexible way. RDF has a simple approach that makes it is easy for applications to process and to manipulate data models. RDF provides simple semantics for a data model for resources and relations between them, described by using XML. The RDF schema defines vocabulary for describing properties and classes of resources. An ontology language adds more vocabulary for describing richer typing of properties and characteristics of properties. Above all relations between classes, for instance cardinality. So technologies like XML or RDF are extended to get an ontology language that can formally describe the meaning of terminology used in documents and to allow machines to perform useful reasoning tasks on these documents.

2.5.2 WSLA Language

The WSLA project of IBM addresses such previously described service level management issues and is aimed at three different situations for SLAs in a Web Service environment: The SLA specification, creation, and monitoring. WSLA documents are agreements between service providers and customers and as such define the obligations of the parties involved ([**WSLA-Spec**]). IBMs WSLA technology can be used by service providers, intermediaries, customers, and also third parties. The WSLA language can be used to specify SLA terms in a formal way that they can be verified by a compliance monitor. The WSLA creation and monitoring framework is only one part of various projects of IBM that are in the context of proactive management in a Web Service environment, including the provisioning of resources or workload management. The term workload management denotes the discipline of effectively managing, controlling, and monitoring job-flow processes across heterogeneous distributed computing environments. The XML schemas defined to represent WSLA terms allow template based authoring, simplifying the creation of SLA after automated negotiations. The global structure of the official WSLA schema file is shown in Figure 2.10. The distributed monitoring framework can be deployed in a single site or across multiple sites ([**WSLA-Home**]). This framework can be used to translate a negotiated SLA into configuration information for the individual service provider components. For instance, the SLA definition can be used by a component that is responsible for setting the maximum bandwidth of a network connection. Note that these components must understand the WSLA language terms. Suppose, there are penalties defined inside the SLA when some terms are temporary not retained. The framework helps third party services to perform measurement actions and supervision activities to identify such not retained terms. The compliance monitor allows taking measurements from different sources, which are distributed over multiple nodes in the network. For example a measurable SLA term could be bound on response time for a supported throughput level. The description of how to measure performance to enforce violations of performance guarantees can be described in unambiguous terms with the WSLA language, which will be delineated in the next paragraph. Finally, SLA specifications of performance guarantees are associated with Web Services operations and e-business processes.

```
<xsd:schema>
  ...
  <xsd:complexType name="WSLAType">
    <xsd:sequence>
      <xsd:element ref="wsla:Parties"/>
      <xsd:element name="ServiceDefinition"
                   type="wsla:ServiceDefinitionType"
                   maxOccurs="unbounded"/>
      <xsd:element name="Obligations" type="wsla:ObligationsType"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string"/>
  </xsd:complexType>
  ...
</xsd:schema>
```

Figure 2.10: Global WSLA structure inside the official WSLA schema file.

The WSLA language is specified in the WSLA language description ([**WSLA-Spec**]) and allows an unambiguous and formally clear specification of SLA terms. One of the important aspects of WSLA is its capability to deal with other specifics of particular domains. That means the language itself is extensible. Hence, that specific types or operation descriptions can be easily included, for example WSDL parts for describing a Web Service operation. The extension mechanism makes use of the ability to create derived types using XML schema definition language ([**XML-Schema**]). Therefore, the core of the WSLA language is kept very compact though WSLA authors can define agreements that relate to Web Services very easy by using a set of WSLA standard extensions. This includes guarantees like response time, throughput and other well-known metrics. There is also the possibility to define specific extensions for other domain-specific agreement terms.

The structure of an SLA is divided into three sections named parties, service description, and obligations as shown in Figure 2.10. The parties section identifies all the contractual entities, including signatory and supporting parties. Note that the signatory parties are the parties that negotiate about an agreement while supporting parties are only to provide several support functionality for the negotiated service itself. The second section is the service description section. all service characteristics and observable SLA parameters are specified in detail. Such SLA parameters are properties of a service object and each parameter has a name, type, and unit. In the WSLA language every SLA parameter refers to one metric that can be an composition of other metrics. These composition can be done by a specific metric that defines a function that uses other metrics as operands. Alternatively, it has a measurement directive that describes how the value of the metric should be measured. Composite metrics are maximum response time of a service, average availability of a service, or minimum throughput of a service. On the other hand, resource metrics are system uptime, service outage period or number of service invocations ([**WSLA-Report**]).

```
<SLAParameter name="OverUtilization" type="float" unit="Percentage">
  <Metric>PercentOverUtilized</Metric>
  <Communication>
    <Source>YMeasurement</Source>
    <Pull>ZAuditing</Pull>
    <Push>ZAuditing</Push>
  </Communication>
</SLAParameter>
```

Figure 2.11: SLA Parameter definition using WSLA ([**WSLA-Report**]).

Figure 2.11 shows that the language defines inside a SLA parameter, which party is supposed to provide the value by the `<Source>` element and which parties can receive it. The reception can be either event-driven, indicated by the `<Push>` element, or through polling, defined by the `<Pull>` element. Furthermore, this example demonstrates how SLA parameters and correspondent metrics can be defined using the WSLA language. The independent definition of metrics facilitates the reuse in the context of other SLA parameters. According to [**WSLA-Report**] `YMeasurement` means a measurement service and `ZAuditing` cooperates with this service as a condition evaluation service that needs to understand the SLA information to evaluate if a service level objective has been reached or violated. Figure 2.11 also shows that the SLA parameter is assigned to the metric `PercentOverUtilized`, which is defined independently. This metric is shown in Figure 2.12. As described in [**WSLA-Report**], the metric is used to determine the amount of time when a system is overloaded.. The value `0.8` in this example indicates that system utilization less than 80% is a safe operation region and above this value the system is considered as overloaded. The function `PercentageGreaterThanThreshold` is used for yielding the percentage of values over a defined threshold in a time series. Such customized functions can be added to the WSLA language as needed. The `<Schedule>` element defines the time intervals during which the function is executed to compute the metric. In this example the schedule `BusinessDay` specifies when and how often the data is supposed to be collected during working days.

```
<Metric name="PercentOverUtilized" type="float" unit="Percentage">
  <Source>YMeasurement</Source>
  <Function xsi:type="PercentageGreaterThanThreshold" resultType="float">
    <Schedule>BusinessDay</Schedule>
    <Value>
      <LongScalar>0.8</LongScalar>
    </Value>
  </Function>
</Metric>
```

Figure 2.12: Metric definition using WSLA ([**WSLA-Report**]).

The obligations section defines the terms, i.e. guarantees and constraints that may be imposed on SLA parameters. There are two types defined:

- Service level objectives
- Action guarantees

While service level objectives represent promises with respect to the state of SLA parameters, actions guarantees are promises of a signatory party to perform an action, including notifica-

tions of service level objective violations or invocations of management operations. The explicit representation of service level objectives and action guarantees provides a very flexible mechanism for the definition of obligations ([**WSLA-Report**]). Both types define the obliged party and the time when the obligations need to be evaluated. Both have a similar syntactical structure but their semantic is different. The description of the structures are defined in [**WSLA-Spec**] and are not part of this thesis. However, Figure 2.13 shows an self-describing example of such a service level objective. Note that details can be found in [**WSLA-Report**].

```
<ServiceLevelObjective name="Conditional SLO For AvgThroughput">
  <Obliged>ACMEProvider</Obliged>
  <Validity>
    <Start>2001-11-30T14:00:00.000-05:00</Start>
    <End>2001-12-31T14:00:00.000-05:00</End>
  </Validity>
  <Expression>
    <Implies>
      <Expression>
        <Predicate xsi:type="Less">
          <SLAParameter>OverUtilization</SLAParameter>
          <Value>0.3</Value>
        </Predicate>
      </Expression>
      <Expression>
        <Predicate xsi:type="Greater">
          <SLAParameter>AvgThroughput</SLAParameter>
          <Value>1000</Value>
        </Predicate>
      </Expression>
    </Implies>
  </Expression>
  <EvaluationEvent>NewValue</EvaluationEvent>
</ServiceLevelObjective>
```

Figure 2.13: Service level objective definition using WSLA ([**WSLA-Report**]).

Overall, the WSLA language is very powerful and allows reasonable authoring of agreements for Web Services. The WSLA is focused on SLA parameters and is rather metric oriented. While there is the possibility to use this language for describing service terms, its somehow overloaded, because there are many interactions possibilities based on functions. Furthermore, WSLA only gives the global structure of an WSLA building a gantry for an agreement. A thinner gantry for agreements in particular would be advantageous for comparing different agreements more effective.

2.5.3 Job Submission Description Language

The JSDL is another language that can be used for the description of terms related to services. The JSDL has a little different orientation compared to the WSLA language. While the WSLA language focuses on an unambiguous formally clear specification of SLA terms, JSDL is more oriented towards the description of requirements for computational jobs, particularly in Grid environments and other Systems. The structure of the language provides the grammar and vocabulary that facilitates those requirements as a set of JSDL job terms ([**JSDL-Spec**]). The

specification of JSDL is developed by the participants of JSDL working group of the Scheduling and Resource Management (SRM) area of the GGF. Since JSDL is a work in progress and not standardized yet there is no implementation available. According to **[JSDL-Wg-Home]**, the working group will provide a specification for an abstract standard language that is independent of actual language bindings. The specification should include attribute semantics, the definition of the relationship between attributes and the range of attribute values. To support currently available systems there is also the intention for the creation of translation tables needed to map to and from the scheduling languages of a set of popular batch systems. That includes the description of job requirements and resource description attributes. JSDL also covers the specification of transfer related issues, like the locations of required input and output files or techniques used for performing the transfers. The language will also provide descriptions of basic dependencies between jobs concerning the status of jobs, for instance states can be named initiated, in calculation or ready. Furthermore, the relationship between input data and output data of jobs.

The simplified goal of JSDL is to abstract the description of jobs in a standard way that different resource-management systems can interpret it. Note that these different resource-management systems are mostly batch management systems that have their own description languages. This again demonstrates an interoperability problem, which means that a user, who wants to use multiple systems, must have several job descriptions, one for each of the existing systems that they wish to use. JSDL can be seen as a standard intermediary language that eliminates the requirement to maintain the knowledge of the syntax of the underlying description logic of systems. Of course, this concept requires that the implementors of the different resource-management systems provide the necessary means by which JSDL can be translated into their proprietary languages. This concept can be seen as a translator or adapter approach that incorporates many currently existing systems. Thereby, one single job, described in JSDL, can be used on any system in a complex heterogenous environment, such as Grid environments.

A deeper look into the JSDL language attributes shows that the requirements for computational jobs can be abstracted to cover the description of service characteristics and SLA parameter in a formal way for services. The JSDL differentiates the requirements of computational jobs into three categories. These categories are:

- job-identification requirements
- resource requirements
- data requirements

The structure of a JSDL document is defined in an XML schema, which main structure is shown in Figure 2.14. JSDL compliant systems must be able to parse and handle all of the core attributes, ie. job identity attributes, resource attributes and data staging attributes (**[JSDL-Spec]**).

The results of the analysis of JSDL language is that it is not an appropriate language for describing service characteristics. The JSDL is simplemindedly too specially designed to compute jobs. That implies that JSDL is not useful for describing service characteristics in Grid environments with QoS and SLA issues. While WSLA is too multi-functional and too overloaded, JSDL is too specialized for building a kind of gantry for an agreement or to describe a special Grid resource available through a Grid Service.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.gridforum.org/JSDL"
            xmlns="http://www.gridforum.org/JSDL"
            elementFormDefault="qualified">
  ...
  <xsd:element name="job">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="JobIdentification"
             minOccurs="0" maxOccurs="1" />
        <xsd:element ref="Resource"
             minOccurs="0" maxOccurs="unbounded" />
        <xsd:element name="ProcessTopology" type="xsd:string"
             minOccurs="0" maxOccurs="1" />
        <xsd:element ref="Environment"
             minOccurs="0" maxOccurs="unbounded" />
        <xsd:element ref="SoftwareRequirements"
             minOccurs="0" maxOccurs="1" />
        <xsd:element ref="Application"
             minOccurs="0" maxOccurs="1" />
        <xsd:element ref="Host"
             minOccurs="0" maxOccurs="unbounded" />
        <xsd:element ref="HostGroup"
             minOccurs="0" maxOccurs="unbounded" />
        <xsd:element ref="DataAttributes"
             minOccurs="0" maxOccurs="1" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  ...
</xsd:schema>
```

Figure 2.14: Main structure of the JSDL schema.

2.5.4 Web Ontology Language

Looking into the Internet and its standards there is another language of interest: the Web Ontology Language (OWL) of the W3C. Note that the abbreviation of this language OWL is not a typo, its the official shortcut for it. The OWL is intended to be used for any kind of information that can be described inside documents, for instance SLA terms. The language can be used to represent the meaning of terms in a formal way including expressive vocabularies and relationships between such terms. Important to know is that OWL has more facilities for expressing meaning and semantics than plain XML or RDF ([**OWL-Overview**]). So OWL is more powerful to represent machine interpretable content on the Web and is a potential candidate for describing service characteristics and SLA terms.

For addressing different and specific communities of implementers and users OWL provides three increasingly expressive sublanguages:

- *'OWL Lite'*
- *'OWL Description Logics'*
- *'OWL Full'*

The first sublanguage is called *'OWL Lite'*. It supports classification hierarchy and simple constraints, for instance only cardinality constraints of 0 or 1. The tool support for this first sublanguage is simple. To be *'OWL Lite'* compliant many restrictions on the use of the OWL language are defined to keep the description simple and to provide a minimum useful subset of language features. The limitations on *'OWL Lite'* place it in a lower complexity class than the second sublanguage *'OWL Description Logics'* ([**OWL-Overview**]). While *'OWL Lite'* has limitations to use only some OWL language constructs, the *'OWL Description Logics'* sublanguage includes all OWL language constructs, but they can be used only under certain restrictions defined in [**OWL-Ref**]. Description logics means a field of research that has studied the logics that form the formal foundation of OWL. Inside [**DL-Handbook**] these logics are defined as a family of knowledge representation languages that have been studied extensively in Artificial Intelligence (AI). These studies covered all aspects of research in this field, including theory, implementation and applications. The logics are embodied and are used to develop various real-life applications and many knowledge-based systems are premised on this description logic, as also the OWL. Therefore the constraints of *'OWL Description Logics'* are based on the work in this logics area and are not just an arbitrary set ([**OWL-Overview**]). The third sublanguage is *'OWL Full'* and allows the maximum expressiveness and the full syntactic freedom of RDF with no computational guarantees. So *'OWL Full'* allows an ontology to augment the meaning of the pre-defined vocabulary and can be viewed as an extension of RDF. *'OWL Lite'* and *'OWL Description Logics'*, on the other hand, can be viewed as extensions of a restricted view of RDF. So *'OWL Full'* is not actually a real sublanguage of OWL, because it contains all the language constructs and allows a not limited use of RDF constructs. Because OWL provides these three increasingly expressive sublanguages the XML schemas for OWL are unitized. This means there is a set of XML schema modules for the XML presentation syntax. The design goal of this modularization is to set up a core module common to all OWL sublanguages and to define additional modules to be customized for the three different sublanguages. More details for collecting required modules together by an schema driver can be found in [**OWL-XML-Syntax**]. Figure 2.15 shows an example of using OWL for a XML representation.

OWL offers interesting perspectives for its use for service characteristics, because with the use of *'OWL Full'* nearly every kind of description of documents are possible. In fact this is precisely the power of OWL, which is a semantic markup language and markup languages are well known to be very variable and powerful. However, the problem is that there is no defined structure for descriptions of SLA terms or QoS issues. OWL appears to be rather abstract. Therefore it is a significant effort to generate a first structure for agreements with the help of OWL. As well as in WSLA or JSDL there is also a thin gantry or structure missing, which can be used in a very simple way.

```
<!DOCTYPE Ontology [
    <!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY vin  "http://www.example.org/wine#" >
    <!ENTITY food "http://www.example.org/food#" > ]>
<owlx:Ontology owlx:name="http://www.example.org/wine"
  xmlns:owlx="http://www.w3.org/2003/05/owl-xml">
  ...
  <owlx:Class owlx:name="VintageYear" owlx:complete="false" />
  ...
  <owlx:DatatypeProperty owlx:name="yearValue">
    <owlx:domain owlx:class="#VintageYear" />
    <owlx:range owlx:datatype="&xsd;positiveInteger" />
  </owlx:DatatypeProperty>
  ...
</owlx:Ontology>
```

Figure 2.15: Examples of the XML Presentation Syntax of OWL ([**OWL-XML-Syntax**]).

2.6 Conclusion

The use of the OGSA and the Grid Services approach are the best choice for now to implement flexible services in a Grid environment. Therefore this thesis will follow the approach of *'OGSA compliant services'*. The demand for describing SLA parameters or QoS issues in a formal way requires an advantageous language. The WSLA language is overloaded and the ontology language of the W3C is a bit to abstract. On the other hand, the JSDL is too much focused on jobs and agreements have not always a job character. To conclude, the definition of a thin well-defined gantry for an agreement is still missed in all the analyzed languages.

Chapter 3

Agreement Negotiations

The standardization of agreement negotiations are the aim of the GGF working group GRAAP of the SRM area of the GGF. That includes the standardization of the terminology, concepts and an overall agreement structure. This chapter provides an overview of the group's proposals.

3.1 Structure of an Agreement

Definition
*An agreement is the act of agreeing or coming to a mutual arrangement ([**Websters**]).*

In a distributed service-oriented computing environment, service consumers like to obtain guarantees related to the services they use. Such guarantees are often related to QoS that depends on the actual resource usage at the requested time of service. Therefore, guarantees, offered by the service provider, cannot simply be advertised as an invariant property of a service. As a consequence, the service consumer must request state-dependent guarantees from the service provider that leads to an agreement on the service and the associated guarantees. Furthermore, service quality must be monitored and failures to meet these guarantees must propagated to the service consumers. The **[WS-Agreement]** specification defines a language and a protocol for advertising the capabilities of service providers and gives a framework to monitor agreement compliance at service runtime. In addition, the specification includes the creation of agreements based on offers by the service provider. To reflect this, the **[WS-Agreement]** specification provides a more specific definition of an agreement:

Definition
*An agreement between a service consumer and a service provider specifies one ore more service level objectives both as expressions of requirements of the service consumer and assurances by the service provider on the availability of resources and/or on service qualities ([**WS-Agreement**]).*

An agreement can, for instance, provide assurances on the bounds on service response time and service availability. In addition, the agreement can provide assurances on the availability of minimum resources such as memory, Central Processing Unit (CPU) or storage. However, to obtain such assurances on service quality, the service consumer must establish a service agreement with the service provider. Important is that the service level objectives relate to the definition of the service. Therefore, the service definition must also be a part of the terms of the agreement. This leads to a schema for defining the overall structure for an agreement that is introduced in **[WS-Agreement]**. Note that the specification includes a special language to

define service level objectives and business values associated with these objectives. Figure 3.1 depicts the proposed structure of an agreement.

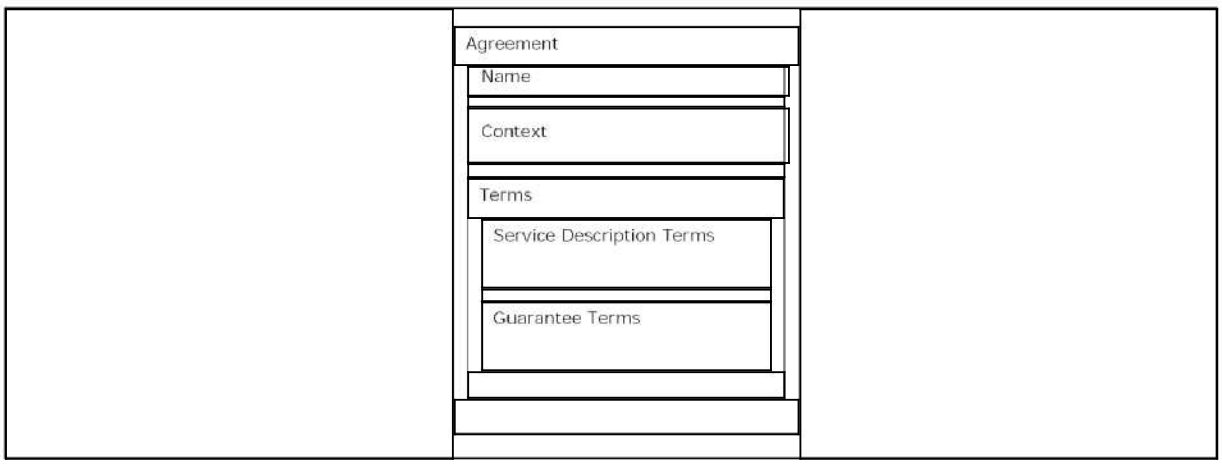

Figure 3.1: Structure of an agreement ([**WS-Agreement**]).

The agreement structure of [**WS-Agreement**] is conceptually composed of several distinct parts. The first part is an optional name for the agreement. The two other parts are the context and terms of an agreement.

Agreement Context Information

The agreement context contains information about the participants in the agreement, including the `<AgreementInitiator>` element that defines the initiator of the agreement creation and the `<AgreementProvider>` element that contains the identification of the agreement provider. Note that all these elements can use the Endpoint Reference of [**WS-Adressing**] for identification of participants. Furthermore, the element `<AgreementInitiatorIsServiceConsumer>` can be used to specify if the agreement initiator is also the service consumer. An agreement context also contains information about the lifetime of the agreement. The `<ExpirationTime>` is used to specify the time at which the agreement is no longer valid. Note that this lifetime is often a result of negotiations and must not be identical to the TerminationTime of a Web Service encapsulation of an agreement that uses [**WS-ResourceLifetime**] of the WSRF. According to [**WS-Agreement**], the `<ExpirationTime>` applies to the agreement, that is, the time at which the agreement terms are no longer in effect. On the the other hand the TerminationTime applies only to the service which encapsulates the agreement and makes it available for inspection. As a consequence, the value of the TerminationTime should be greater or equal to the `<ExpirationTime>`.

The `<TemplateName>` element specifies the name of the agreement template from which this agreement is created. The reference to the template is useful both for future modification of the agreement as well as for the provisioning of the service environment by the service provider. Finally, the context contains links to other related agreements, defined by the contents of the `<RelatedAgreements>` element. This element contains a list of any number of related agreements and thus allows to create agreement compositions. Note that the related agreements are represented in the agreement service as related agreement services. Figure 3.2 shows an overview of the proposed context of an agreement structure.

```
<wsag:Agreement>
  ...
  <wsag:Context>
    <wsag:AgreementInitiator>xs:anyType</wsag:AgreementInitiator>
    <wsag:AgreementProvider>xs:anyType</wsag:AgreementProvider>
    <wsag:AgreementInitiatorIsServiceConsumer>
      xsd:boolean
    </wsag:AgreementInitiatorIsServiceConsumer>
    <wsag:ExpirationTime>xs:DateTime</wsag:ExpirationTime>
    <wsag:TemplateName>xs:string </wsag:TemplateName>
    <wsag:RelatedAgreements>
      <wsag:RelatedAgreement
        <wsag:AgreementEPR>
          wsa:EndpointReferenceType
        </wsag:AgreementEPR>
      </wsag:RelatedAgreement>
    </wsag:RelatedAgreements>
  </wsag:Context>
  ...
</wsag:Agreement>
```

Figure 3.2: XML schema of the agreement context.

Agreement Terms

The agreement terms consist of one or more Service Description Terms (SDT) and zero or more guarantee terms. The SDT are the fundamental terms of an agreement, because they can be used to describe the essential service terms of the correspondent service, also called functional terms. As a consequence, SDT define the functionality that will be delivered under an agreement. Note that the content of SDT is usually domain-specific. However, the [**WS-Agreement**] specification defines the `<ServiceDescriptionTerm>` element and its content to specify a SDT structure. The `Name` attribute of this element represents the unique name given to a term. In addition, the attribute `ServiceName` identifies a service across multiple SDT. Note that this identifier is scoped within the agreement and not to identify the service outside of the agreement. Figure 3.3 shows an example of an SDT, including a domain-specific service description term, called `numberOfCPUs`. The example shows the description of a computational job to execute.

```
<wsag:Agreement>
  ...
  <wsag:Terms>
    ...
    <wsag:ServiceDescriptionTerm
            wsag:Name="numberOfCPUsLow"
            wsag:ServiceName="ComputeJob1">
      <job:numberOfCPUs>8</job:numberOfCPUs>
    </wsag:ServiceDescriptionTerm>
    ...
  </wsag: Terms>
  ...
</wsag:Agreement>
```

Figure 3.3: Example of a `ServiceDescriptionTerm`.

The primary motivation for creating a service agreement is to provide assurance to the service consumer on the service quality. Guarantee terms define this assurance on service quality, associated with the service described by the SDT ([**WS-Agreement**]). The `<GuaranteeTerm>` element contents represent an individual guarantee related to the SDTs that describe the service and can therefore be called non-functional terms. The element `<ServiceScope>` contains a list of service names referring to the respective `<ServiceName>` attribute of one or more SDT. Hence, the guarantee applies to every service in the list. The `<ServiceLevelObjective>` element expresses the condition that must be met to satisfy the guarantee. To give a small example, Figure 3.4 defines a `<ServiceLevelObjective>` that is related to a SDT called `<numberOfCPUsLow>`. The element `<BusinessValueList>` contains a list of business value elements associated with a service level objective. One business value element is the `<Importance>` that can be used to express relative importance of meeting an objective. In addition, the `<Penalty>` element can be used to express the penalty to be assessed for not meeting an objective. The `<Reward>` element expresses reward to be assessed for meeting an objective. Finally, the `<Preference>` element specifies a list of fine-granularity business values for different alternatives. Note that each alternative refers to a SDT and its associated utility. However, Figure 3.4 shows an example of a guarantee term used in a computational job agreement. A more detailed description of SDT and guarantee terms as well as different usage scenarios can be found in [**WS-Agreement**].

```
<wsag:Agreement>
  ...
  <wsag:Terms>
    ...
    <wsag:GuaranteeTerm wsag:Name="ConfigurationPreference">
      <wsag:ServiceScope>
        <wsag:ServiceName>ComputeJob1</wsag:ServiceName>
      </wsag:ServiceScope>
      <wsag:ServiceLevelObjective>
        <SDT>numberOfCPUsLow</SDT>
      </wsag:ServiceLevelObjective>
      <wsag:BusinessValueList>
        <wsag:Importance/>
        <wsag:Penalty/>
        <wsag:Reward/>
        <wsag:Preference>
          <wsag:ServiceTermReference>
            numberOfCPUsLow
          </wsag:ServiceTermReference>
          <wsag:Utility>0.5</wsag:Utility>
        </wsag:Preference>
      </wsag:BusinessValueList>
    </wsag:GuaranteeTerm>
    ...
  </wsag: Terms>
  ...
</wsag:Agreement>
```

Figure 3.4: Example of a `GuaranteeTerm`.

3.2 Agreement creation process

The agreement creation process typically starts with a pre-defined agreement template specifying customizable aspects of the document. In addition, the agreement template contains rules that must be followed in creating an agreement. The **[WS-Agreement]** specification named this creation constraints. An agreement factory provides such agreement templates at the service provider. Usually a service consumer makes an agreement creation offer to an agreement factory that has the same structure as an agreement. In particular, the agreement factory advertises the types of agreement offers it is willing to accept by means of agreement templates. According to **[WS-Agreement]**, an agreement template is composed of three distinct parts as shown in Figure 3.5. As a side-remark, the creation of an agreement can be initiated by the service consumer or by the service provider. This symmetry is also provided in **[WS-Agreement]**.

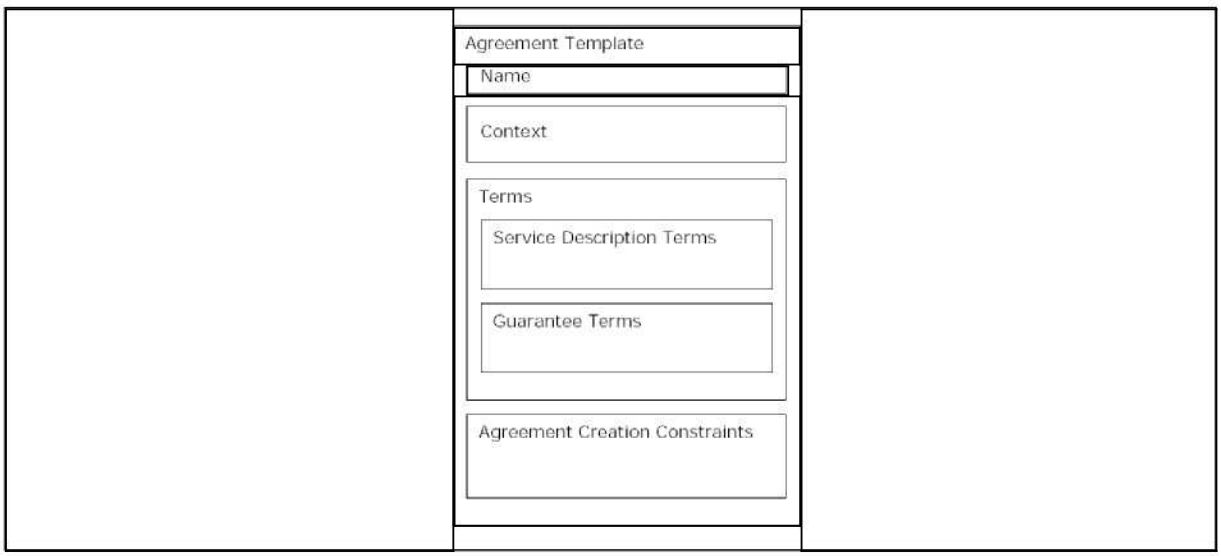

Figure 3.5: Structure of an agreement template (**[WS-Agreement]**).

The structure of an agreement template is the same as that of an agreement, but an agreement template can also contain a creation constraint part. This part lists constraints on possible values of terms for creating an agreement. Furthermore, the constraints make it possible to specify the valid ranges or distinct values that the terms may take. Note that the specification of a creation constraint part in a template does not necessarily state a promise that all agreement creation offers that fulfilling the constraints will be accepted. In particular, the agreement provider published an agreement template containing a creation constraint part, outlining agreements that the service provider is generally willing to accept. But if the agreement provider really accepts a given offer depends on the current resource situation or a special policy running on the service provider. Note that an agreement template should also consists of an explicit period of validity for the template, for instance one hour after receipt of it. Figure 3.6 depicts the schema of the `<CreationConstraints>` element of a template. The `<Item>` element specifies that a particular field of the agreement must be present with a value in the agreement offer, and which values are possible. The child element `<Location>` can be used as a structural reference, for instance an XML Path Language (XPATH) expression that points to the location in the terms of the agreement that can be changed and filled in. Finally, the `<Constraint>` element defines any constraint on the values of one ore more terms. More detailed information about the elements related to an agreement template can be found in **[WS-Agreement]**.

```
<wsag:Template>
  ...
  <wsag:CreationConstraints>
    <wsag:Item Name="xsd:NCName">
      <wsag:Location>xsd:anyType</wsag:Location>
    </wsag:Item>
    <wsag:Constraint>...</wsag:Constraint>
  </wsag:CreationConstraints>
  ...
</wsag:Template>
```

Figure 3.6: Structure of the `CreationConstraints`.

3.3 Aspects of Negotiations

The [**WS-Agreement**] specification includes a well-defined message exchange that can be used for simple agreement negotiations. Hence, an agreement template can be used to create an agreement offer by the service requestor. This is accomplished by filling out the terms supported by the template that leads to an agreement offer with the same structure as the template, excluding some creation constraint details. The agreement offer can be accepted or rejected by the service provider depending on the terms within the offer. If the offer is not accepted a new offer with different term definitions can be created to negotiate again. Information about the rejection can be transfered inside the create response to the service consumer. To conclude, this agreement creation process, as defined in [**WS-Agreement**], can be seen as a simple mechanism for agreement negotiations. The message exchange of this mechanism is shown in Figure 3.7.

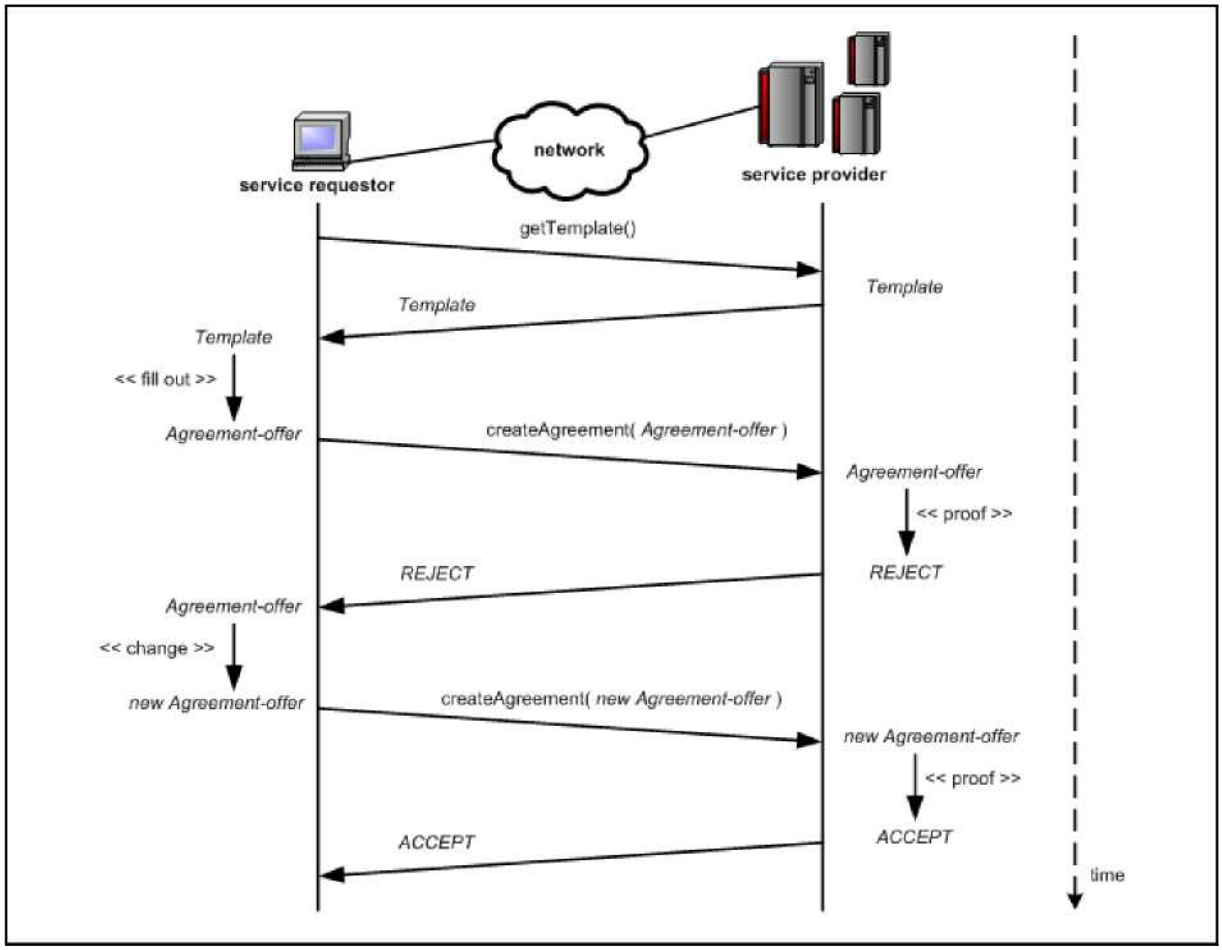

Figure 3.7: Simple agreement negotiations message exchange.

The aim of the GGF GRAAP group is also the provisioning of an specification that includes aspects such as a better support for negotiations and re-negotiations of agreements. The message exchanges and operations of this aspects will be defined in the [**WS-AgreementNegotiation**] specification that is still an unusable working draft.

3.4 Example Scenarios

The proposals of the [**WS-Agreement**] specification can be used in various application scenarios related to the establishment of an agreement between a service provider and a service consumer. This section provides three examples of such application scenarios and concentrate on a network reservation example that will be used later as a case study. Wihtin the examples the service provider acts as the agreement provider while the service consumer acts as the agreement initiator.

3.4.1 Network Reservation

An organization wants to establish a virtual connection between one distributed application. This application can be a resource intensive real-time 3D application or videoconferencing software. The applications can be connected to each other using a network infrastructure. Now consider a situation in which several application sessions are in operation simultaneously, while other organizations are concurrently attempting to perform high-speed bulk-data transfers over the same network infrastructure. This situation leads to a complex and dynamic mixture of network service requirements, including network QoS issues. For instance, such a network infrastructure can consist of four Cisco 7200 series routers that are either connected by ATM connections, by Fast Ethernet, or by Gigabit Ethernet connections. As a consequence, there can be a network tunnel established between this connections. Figure 3.8 shows this network infrastructure represented by a laboratory `Testbed` at the Research Centre Jülich.

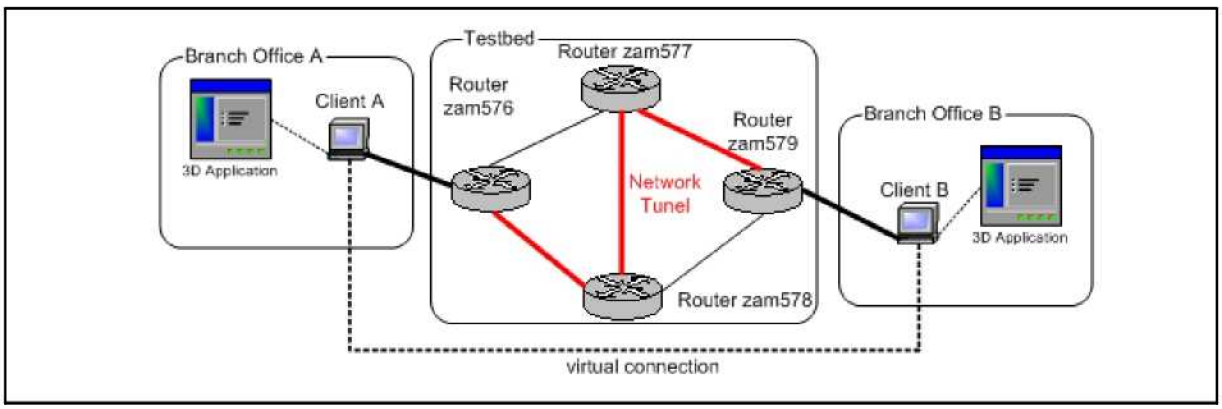

Figure 3.8: Network reservation between two endpoints.

One approach to provide network QoS is the *'reservation based'* approach that usually rely on specific capabilities of the underlying network infrastructure ([**Pab-Report**]). These specific capabilites can be exposed by a service provider that offers a network service to establish a connection through the network infrastructure between two endpoints. The [**WS-Agreement**] specification addresses the demand for exposed domain-specific service capabilities with agreement templates. These agreement templates are used by a service consumer, in this case the organization that wants to establish a connection between two applications, to specify their QoS requirements. For instance, such QoS requirements can be:

- a delay of 10 ms - 50ms
- a bandwidth of 15MBit/s

The service consumer specifies the QoS requirements during the agreement negotiation process and typically does not change the demand subsequently. The service providers in turn guarantee that the reservation will not change during the lifetime of the service. The [**WS-Agreement**] specification addresses the description of QoS requirements as guarantee terms inside a concrete agreement offer. The concrete agreement offer is used by the service consumer to specify a service request for the service provider. In addition, the agreement offer also includes functional terms that are necessary to establish a service. The following enumeration lists some functional terms that are necessary to establish a communication between the two mentioned 3D applications of the organization:

- IP-Address Client A
- IP-Address Client B
- Port Client A
- Port Client B
- used Protocol (UDP/TCP)

Such functional terms are defined inside the agreement offer as service description terms. Figure 3.9 shows some service description terms of an agreement offer that can be used for network reservation. Note that the namespace `job` defines an XML schema that defines elements related to network reservation. The complete listing of this agreement offer can be found in Appendix A.

```
<wsag:AgreementOffer xmlns:job=".." ...>
  <wsag:Name>...</wsag:Name>
  <wsag:Context>...</wsag:Context>
  <wsag:Terms>
    <wsag:All>
      <wsag:ServiceDescriptionTerm
                  wsag:Name="StartTime"
                  wsag:ServiceName="NetworkAgreement">
        <job:StartTime>2004-07-01T08:00:00</job:StartTime>
      </wsag:ServiceDescriptionTerm>
      <wsag:ServiceDescriptionTerm
                  wsag:Name="IPADDRESSA"
                  wsag:ServiceName="NetworkAgreement">
        <job:IPAddr>130.68.90.9</job:IPAddr>
      </wsag:ServiceDescriptionTerm>
      <wsag:ServiceDescriptionTerm
                  wsag:Name="PORTA"
                  wsag:ServiceName="NetworkAgreement">
        <job:Port>5000</job:Port>
      </wsag:ServiceDescriptionTerm>
      <wsag:ServiceDescriptionTerm
                  wsag:Name="BANDWIDTH"
                  wsag:ServiceName="NetworkAgreement">
        <job:Bandwidth>50MBit/s</job:Bandwidth>
      </wsag:ServiceDescriptionTerm>
      <wsag:ServiceDescriptionTerm
                   wsag:Name="DELAY"
                   wsag:ServiceName="NetworkAgreement">
         <job:Delay>10ms - 50ms</job:Delay>
      </wsag:ServiceDescriptionTerm>
      ...
    </wsag:All>
  </wsag:Terms>
</wsag:AgreementOffer>
```

Figure 3.9: Elements of an agreement offer related to network reservation.

3.4.2 Job Submission

The request for executing a computing job can be used to demonstrate agreement negotiations. The agreement provider provides an agreement template available to interested requestors. For a job submission scenario the template includes a list of applications to be executed and the software execution environment. Furthermore, the template includes guarantee terms to provide QoS for a service consumer, for instance a guarantee on the completion time. To submit a job, the service consumer retrieves such a template from the service provider, selects the application name, and provides the URL of the input and output files. The filled template is sent as an offer to the agreement provider. Whether the agreement specifying the job is accepted or rejected depends on the actual resource situation at the agreement provider. In this example, the situation depends on the queue of jobs waiting to be processed and the current allocation of resources. If the agreement provider accepts the agreement offer, a confirmation message is sent to the service consumer and the job is automatically allocated at the service provider, i.e. submitted with the given constraints. The service provider processes the job and writes the output file to the URL defined in the agreement. However, if the offer is rejected, the agreement provider sent a fault message to the service consumer.

3.4.3 Service Parameterization

This example uses an application service provided by a financial company. The application service consists of online banking and investment. The online banking service operations are accessible via a Web browser. Therefore, online banking operations, for instance `GetAcctBalance` or `GetTransactionHistory`, are exposed via a portal. On the other hand the invest banking operations, for example `BuyShares` or `SellShared`, are accessible as Web Services. As a consequence, these investment operations are exposed via WSDL. Suppose, the financial company offers several service levels, such as a Premium, Selected, and Basic service. Each service level requires a minimum investment amount or account balance. Additionally, each service level provides a certain QoS that is defined in an agreement template. Therefore, the template includes SDT, guarantee terms and the QoS options available to the customer. Hence, new customers of the financial company select a service level by customizing these options in the template. For instance, customers can add availability and response time guarantees to individual operations of the Web Service's interface. Note that these options can be seen as a service parameterization.

With the help of the template a customer creates a complete agreement offer and sends it to the service provider. Whether the agreement is accepted or rejected depends on the capacity limitations of the service provider. Hence, the consumer asks for 1 sec response time for up to 1000 requests per minute but the service provider might only have a capacity for up to 500 requests left, the agreement is rejected. The information about why the agreement is rejected can be included inside a fault that is send back to the customer. Hence, this is not a very precise way on how to inform the customer which capacities are left. Suppose, while the fault is sent back to the customer the resource situation at the service provider can change. For instance, their might only be a capacity for up to 300 requests left. After the reception of the fault, the customer has the possibility to create a new offer with different desired service levels. Hence, the customer can create an agreement that will be rejected again, because the fault gives no guarantees that the information inside the fault will be the actual resource situation at the service provider. On the other hand, if the agreement offer is accepted, the service provider provisions the service and exposes status information on guarantee compliance to the customer ([**WS-Agreement**]).

Chapter 4

Technology Frameworks

This chapter provides an overview about conditions in Grid environments and introduces techniques to create advanced services in such stateless environments.

4.1 Conditions of a Grid Environment

This section provides an overview about the conditions in Grid environments. Already propagated solutions to solve the statelessness in such stateless environments are analyzed for their use in Web Services. Furthermore, an introduction to Web Services Addressing (WS-Addressing) is given that provides a standard mechanism for addressing resources in stateless environments.

4.1.1 Stateless Environment

The statelessness of Web Services is solved by the factory approach defined in the OGSA specification. As a consequence, multiple instances can be created on the server-side to guarantee access to stateful resources and therefore stateful Web Services. This factory approach is one of the cornerstones of the improvements that were leading to Grid Services. However, OGSA does not define a particular solution for the implementation of such stateful Web Services. Web Services are normally using the stateless protocol HTTP for transportation of messages. HTTP provides no built-in way for a server to recognize that a sequence of Web Service requests were all originated from the same requestor ([**Java-Servlet**]). In particular, the server identifies only a series of requests and the server needs additional information to identify who is making a request. A well-known question in this context is, why the server cannot identify the requestor by the connecting machine's IP address. Note that the reported IP address could possibly be the address of a proxy server or the address of a server machine that hosts multiple users ([**Java-Servlet**]). However, each requestor needs to introduce itself in each request. The requestor needs to provide a unique identifier in the request or some information that the server can use to properly associate the request. To conclude, Web Services and the improvements for Grid Services with the factory approach allow stateful instances on the server-side, but the Services live within a stateless environment for transportation owing HTTP as transportation solution. Therefore, all messages are not directly related to each other, since HTTP provides no state of messages. The problem of statelessness in using HTTP leads to several *'session tracking techniques'* while the term session can be interpreted as the relation of different messages to each other.

One way to support session tracking is to use hidden form fields. This technique is used in HTML documents wherein hidden fields are added to a HTML form that are not displayed in the client's browser. The underlying idea is to automatically add parameters, for instance a `sessionID`, in the hidden fields to create unique identifiers in each submitted form. Figure 4.1 shows the syntax of this fields. In a sense, hidden form fields define variables for a form that were constant within a session ([**Java-Servlet**]). The disadvantage of this technique is that it is only useful when working with HTML, Java 2 Platform Enterprise Edition (J2EE) Servlets or JavaServer Pages (JSP). The advantage of hidden form fields is their ubiquity and support for anonymity. Therefore, this technique is supported in all the popular browsers demanding no special server requirements.

```
<HTML>
  ...
  <FORM Action="/servlet/Service" Method="Post">
    ...
    <INPUT Type="hidden" Name="sessionID" Value="1234">
    ...
  </FORM>
  ...
</HTML>
```

Figure 4.1: Hidden form fields as session tracking technique.

Another way to support session tracking is called URL-Rewriting. With URL-Rewriting, every URL is dynamically modified, or rewritten, to include extra information such as `sessionIDs`. Figure 4.2 shows an example of an URL that has been rewritten to pass the `sessionID` to the server. A disadvantage of this technique is that every user can see the rewrited information as part of the rewritten URL. Furthermore, the addition of session information to the URL can be an significant effort. This technique works also with all browsers. More information related to URL-Rewriting techniques can be found in [**Java-Servlet**].

```
http://server:port/servlet/Service
(Original)

http://server:port/servlet/Service?sessionID=1234
(Rewritten)

http://www.google.de/search?q=Agreement
(Google uses URL-Rewriting for passing search keywords)
```

Figure 4.2: URL-Rewriting as session tracking technique.

Persistent cookies are another technique to perform session tracking. When a browser receives a cookie, it saves it and thereafter sends the cookie back to the server each time it accesses a page on that server ([**Java-Servlet**]). Hence, cookies are stored on the client side. Cookies can contain a piece of information that can uniquely identify a client, therefore cookies are often used for session tracking and to pass different values to the server. The cookie technique is not a part of the official HTTP specification but they become a de facto standard supported by all the popular browsers. Inside cookies are usually strings of and numbers that can be difficult

to interprete by a human, including URL, `sessionIDs` or other pieces of information. However, cookies offer an elegant, efficient, easy way to implement session tracking. Noticeable is the automatism that provides a client `sessionID` and perhaps a list of the client's preferences for each request. Note that browsers do not accept cookies, because the user can be deactivate this mechanism for privacy reasons. More information about cookies can be found in [**Cookie-Spec**].

The session tracking API of J2EE Servlets is a way of working with sessions that automatically uses the techniques described above. The Servlet API provides several methods and classes specifically designed to handle session-tracking on behalf of Servlets. The portion of the Servlet API denoted to session tracking is called the session tracking API ([**Java-Servlet**]). Note that this API bases on the use of cookies with the ability to revert to using URL-Rewriting when cookies fail. There is also the possibility to store server-sided objects in relation to a specific session. More information about the capabilities of Servlets and their session tracking API can be found in [**Servlet-SPEC**].

All these *'session tracking techniques'* are difficult to use in the context of Web Services, but they are related to the problem of addressing instances or identifying successively requests inside stateless environments. While all techniques offer the basic functionality, there is no real standard that is incorporated in the Web Service architecture. They are all some kind of workarounds and no real standard for Web Services. Since, the Servlet API can be used if every request of a Web Service is forwarded to a responsible Servlet that works upon the request, the Web Service is not more than the interface to reach the Servlet beyond it. Therefore, the Web Service can be seen as a wrapper mechanism for the Servlet. Hence, the Servlet technology could be used alone, without a Web Service. While these techniques can be used in HTML, J2EE Servlets or JSP, there lack of support for Web Services demands another kind of technique that is achieved with WS-Addressing ([**WS-Adressing**]).

4.1.2 The solution of WS-Addressing

The WS-Addressing specification defines two interoperable constructs that convey information that is typically provided by transport protocols and messaging systems. These constructs are:

- Endpoint Reference (EPR)
- Message Information Header (MIH)

The basic idea is to bring underlying information into a uniform format that can be processed independently within transport or application layers. Note that a Web Service endpoint is a reference-able entity or resource where Web Service messages can be targeted. The EPR construct conveys information needed to access or reference a Web Service endpoint. Furthermore, an EPR provides addresses for individual messages sent to and from Web Services, using the MIH. The MIH is a construct that conveys end-to-end message characteristics including addressing for source and destination endpoints as well as message identity ([**WS-Adressing**]). This section introduces the two constructs of WS-Addressing and how they can be used to achieve stateful communication in a stateless environment.

Endpoint References

The definition of the construct EPR intends to facilitate the dynamic generation and customization of service endpoint descriptions. Very important is the identification and description of specific service instances that are created as the result of stateful interactions that is also supported by an EPR. Furthermore, the support of flexible and dynamic exchange of endpoint information in tightly coupled environments where communicating parties share a set of common assumptions about specific policies or protocols that are used during the interaction ([**WS-Adressing**]). The EPR construct is a lightweight and extensible mechanism to dynamically identify and describe service endpoints and instances. An EPR extends the WSDL description model, but does not replace it. As a consequence, the EPR construct and WSDL evolves coherently.

```
<wsa:EndpointReference>
  <wsa:Address>xs:anyURI</wsa:Address>
  <wsa:ReferenceProperties>...</wsa:ReferenceProperties>
  ...
</wsa:EndpointReference>
```

Figure 4.3: XML representation of an Endpoint Reference.

Figure 4.3 shows only the important parts of the EPR construct, because several elements within the EPR are optional. While the required element `<wsa:Address>` specifies the logical address for the service endpoint, the optional element `<wsa:ReferenceProperties>` contains the elements that convey the reference properties of a particular reference. Hence, this element and child elements can be used to convey information that identify the instances on the server-side. For instance a particular `ResourceID` can be deposed within the `<wsa:ReferenceProperties>` element that can be used to identify an object on the server with this specific `ResourceID`. Figure 4.4 shows an example of an EPR using reference properties to address one single instance on the server.

```
<wsa:EndpointReference>
  <wsa:Address>http://localhost:4400/wstk/services/Agreement</wsa:Address>
  <wsa:ReferenceProperties>
    <ns:ResourceID>177</ns:ResourceID>
  </wsa:ReferenceProperties>
  ...
</wsa:EndpointReference>
```

Figure 4.4: Example of an EPR using reference properties.

Message Information Headers

[**WS-Adressing**] also defines the model and syntax of a MIH. This header collectively augments a message with properties that enabling identification and location of the endpoints involved in an interaction. The MIH blocks provide end-to-end characteristics of a message that can be easily secured as a unit. Furthermore, the information in these headers is immutable and not intended to be modified along the message path. However, Figure 4.5 shows the XML representation of an MIH.

```
<headerXYZ>
   ...
   <wsa:MessageID>xs:anyURI</wsa:MessageID>
   <wsa:To>xs:anyURI</wsa:To>
   <wsa:Action>xs:anyURI</wsa:Action>
   <wsa:From>Endpoint Reference</wsa:From>
   <wsa:RelatesTo>xs:anyURI</wsa:RelatesTo>
   ...
</headerXYZ>
```

Figure 4.5: XML representation of a Message Information Header.

The `<wsa:MessageID>` element conveys the message id property that can be used to identify one single message within a collection of messages. Therefore a Universal Unique IDentifier (UUID) can be used to give the message a unique `messageID`. The mechanism used to guarantee that a UUID is unique is through combinations of hardware addresses, time stamps and random seeds. Such a UUID is a 128 bit number that is shown in the example in Figure 4.6. As a consequence, every message in the interaction process can be identified by this unique `messageID`. `<wsa:To>` is a required element that is used to provides the value for the destination property. In particular, this element provides the address of the intended receiver of the message. Normally, servers have a well-known address while many clients have no pre-known address, so the clients are addressed anonymously by the server as shown in Figure 4.6. The `<wsa:Action>` element contains an Uniform Resource Identifier (URI) ([**RFC2396**]) that identifies the semantics implied by this message. According to [**WS-Adressing**] it is recommended that the value of this property is a URI identifying an input, output or fault message within a WSDL portType. `<wsa:From>` provides a reference of the source endpoint where the message originated from. Finally `<wsa:RelatesTo>` indicate how a message relates to another message. The related message is identified by an URI that correspondent to the related message's `messageID` property, by using the UUID.

```
<SOAP:header>
   ...
   <wsa:MessageID>uuid:6fd0087e865ea94518d18b323b68eaba</wsa:MessageID>
   <wsa:To>
     http://schemas.xmlsoap.org/ws/2003/03/addressing/role/anonymous
   </wsa:To>
   <wsa:Action>
     http://www.ggf.org/namespaces/ws-agreement/createAgreementResponse
   </wsa:Action>
   <wsa:From>
     <wsa:Address>
       http://localhost:4400/wstk/services/AgreementFactory
     </wsa:Address>
   </wsa:From>
   <wsa:RelatesTo>uuid:1gd0084c865ea945550e318de42a518a</wsa:RelatesTo>
   ...
</SOAP:header>
<SOAP:body>
   ...
</SOAP:body>
```

Figure 4.6: A SOAP message using the MIH.

SOAP Binding of Endpoint References

The specification of [**WS-Adressing**] defines its functionality and concepts not for a specific protocol. Therefore, a protocol-specific binding defines how information of the specification is used in a corespondent protocol such as SOAP. To address a service endpoint using an EPR, the protocol-specific binding for SOAP defines how information in the EPR is copied to message and protocol fields. Therefore, the [**WS-Adressing**] specification defines a SOAP binding for an EPR. Figure 4.4 provides an example of an EPR that can be used to address a service endpoint. The first binding rule, is that the `<wsa:Address>` value of the EPR is copied into the `<wsa:To>` element of the header. All childrens of the `<wsa:ReferenceProperties>` element are copied into the header. As a consequence, the `<ns:ResourceID>` element is copied into the header. Finally, Figure 4.7 shows the results of the SOAP bindings related to the EPR as shown in Figure 4.4. To conclude, the binding allows the direct access to a resource at the service provider.

```
<SOAP:header>
   ...
   <wsa:MessageID>...</wsa:MessageID>
   ...
   <wsa:To>
     http://localhost:4400/wstk/services/Agreement
   </wsa:To>
   <ns:ResourceID>177</ns:ResourceID>
   ...
   <wsa:Action>...</wsa:Action>
   <wsa:From>...</wsa:From>
   <wsa:RelatesTo>....</wsa:RelatesTo>
   ...
</SOAP:header>
<SOAP:body>
   ...
</SOAP:body>
```

Figure 4.7: A SOAP message addressed using an EPR.

4.2 OGSI

The Open Grid Services Infrastructure (OGSI) specification ([**OGSI-Spec**]) defines a set of conventions and extensions on the use of the WSDL and XML schema to enable stateful Web Services. This specification includes definitions for creating, naming and managing the lifetime of instances of services. OGSI was the first proposal to implement the idea of stateful Web Services as stated in OGSA. Furthermore, the specification defines approaches for declaring and inspecting service state data and approaches for asynchronous notification of service state change. This section describes the ideas beyond OGSI, lists its advantages and disadvantages, turning into a refactoring and evolution process of OGSI, which ends up in the new WSRF. WSRF retains essentially all of the functional capabilities present in OGSI. Note that for a better understanding of WSRF the ideas beyond OGSI might be helpful.

4.2.1 The ideas beyond OGSI

The OGSI specification defines the creation, addressing, inspection and lifetime-management of stateful Grid Services as proposed by the OGSA architecture. Furthermore, the specification defines a Grid Service as a Web Service that conforms to a set of conventions that define how a client interacts with a Grid Service ([**WS-Refactor**]). OGSI can be seen as a solution for the many issues that are commonly required in distributed applications, including the controlled, fault-resilient and secure management of the distributed and often long-lived states of resources.

The concept of stateful Web Services is introduced using extended WSDL and XML schema. These WSDL extensions have analogous support in WSDL 2.0, which is still an working draft ([**WSDL 2.0**]). The WSDL constructs are for representing, querying and updating meta-data and state data of a service. [**OGSI-Spec**] specifies a Grid Service handle and Grid Service references that are used to address Grid Services. Furthermore, it specifies a common fault format without modifying the WSDL fault message model as defined in [**WSDL 1.1**]. OGSI defines a set of operations for creating and destroying Grid Services, including explicit destruction and implicit garbage collection of expired services. The creation and addressing of collections of Web Services is also a part of OGSI. Mechanisms for requesting asynchronous notifications of state changes are also provided by OGSI.

According to [**WS-Refactor**] there are at least six different implementations of the OGSI specification leading to a broad set of experiences that have been gained with the use of OGSI constructs in applications. Note that various efforts have started to develop higher-level specifications, based on OGSI constructs. Figure 4.8 shows the relationship of OGSA, OGSI and Grid Services.

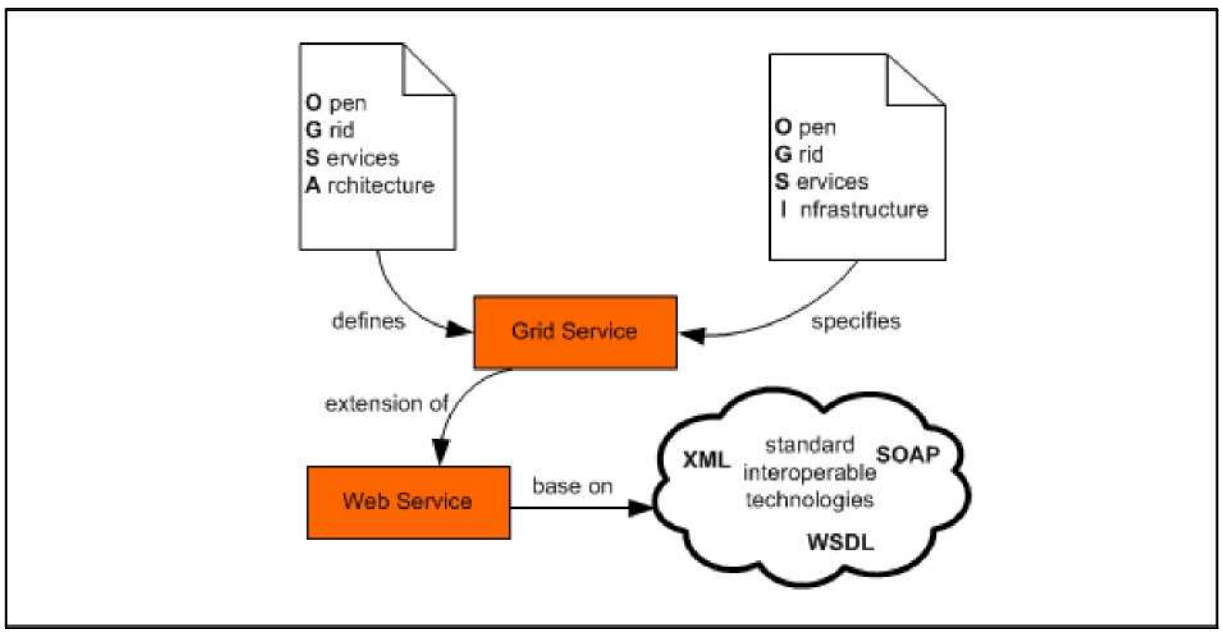

Figure 4.8: Relationship between OGSA, OGSI and Grid Services.

4.2.2 OGSI Evolution

Since development started on OGSI in early 2002, the Web Services world has evolved significantly but independently ([**WS-Refactor**]). Hence, many new specifications and use patterns emerged that simplified and clarified some ideas expressed in OGSI. For instance the transport-neutral mechanisms to address Web Services with [**WS-Adressing**] or [**WS-MetaDataExchange**] that provides a collection of mechanisms for obtaining information about a published service, such as its WSDL description, its XML schema definitions, and any other policy information necessary to use the service. Therefore, it is better to exploit those new Web Services specifications rather than maintaining a specification that defines the same functionality within OGSI redundantly. However, the idea of Grid Services is one of the core parts of OGSI. [**OGSI-Spec**] specifies a Grid Service as a Web Service that conforms to a set of conventions for Grid purposes, such as service lifetime management, inspection and notification of service state changes. The OGSI core part with Grid Services is heavily related with the Web Services technology. Therefore it was time to consider how the functional capabilities of OGSI exploit functionality provided by other new specifications, for instance the new WS-Addressing specification. Also the alignment of OGSI functions with the emerging consensus on the Web Services architecture ([**WS-Arch**]) had to be reconsidered. The composed approach of the OGSI specification includes many functions that are independently useful, for example event notification ([**WS-Refactor**]). That's why it was appropriate to refactor OGSI leading to a framework of independently useful Web Services standards.

The evolution of OGSI and the Web Services technology goes further than the refactoring process addresses. Besides the integration of new standards, also four criticisms of OGSI from the Web Services community were addressed ([**WS-Refactor**]):

- *'Too much stuff in one specification.'*
- *'does not work well with existing Web Services and XML tooling.'*
- *'too object oriented'*
- *'Introduction of forthcoming WSDL 2.0 capability as unsupported extensions to WSDL 1.1 '*

'Too much stuff in one specification.' This is the first critique of OGSI and means that the OGSI specification has not a clean separation of functions to support incremental adoption. For instance the event notification is a useful function, but independent of coupling with service data. This notification event could be separated from the specification leading to an approach that can be seen as a composition of the different Web Service specifications. *'does not work well with existing Web Services and XML tooling.'* This describes the second critique that addresses the aggressively use of XML schema, for example substantial use of `xsd:any` and other document oriented WSDL operations. This approach of OGSI cause problems with, for example, JAX-RPC. The Java API for XML-based RPC (JAX-RPC) can be used to build Web applications and Web services, incorporating XML-based RPC functionality according to the SOAP 1.1 specification ([**JAX-RPC**]). This critique can be seen as an interoperability problem for existing XML tooling, usually used in the Web Services community. OGSI also extends the WSDL portType definition, that implies problems with existing Web Services. OGSI is also *'too object oriented'*. This means that pure Web Services have no state or instances, but the specification encapsulates the resource's state with the identity and lifecycle of the service. There is an demand for explicit distinction between the service itself and the stateful entities, which are acting upon that service. [**WS-Refactor**] named the last critique point *'Introduction of forthcoming WSDL 2.0*

capability as unsupported extensions to WSDL 1.1 '. OGSI exploits constructs from the proposed WSDL 2.0 draft specification, which publication is delayed. This implies that it is difficult to support OGSI with existing Web Services tooling and runtimes.

4.3 WSRF

This section introduces the ideas beyond the WSRF and how the framework fits the problems and criticism of OGSI.

4.3.1 Mastering the problems of OGSI

The general approach and underlying motivations for the factoring and evolution process that takes OGSI to the WSRF was divided into three evolutionary steps. The first step was the introduction of the WS-Resource concept that is used in all the specifications of the WSRF. In particular, WSRF refactors the OGSI architecture to make an explicit distinction between the service and the stateful entities used by that service ([**WS-Refactor**]). Therefore, WSRF defines the means by which a Web Service and a stateful resource are composed. Hence, the OGSI critique *'too object oriented'* services, is fixed, because the identity and the lifecycle of the service and resource state is not coupled anymore. That leads to the possibility of establishing pure Web Services with the WSRF without any kind of resource states.

The second step in the evolution process aims on a better separation of the functionality. As a consequence, the OGSI critique *'too much stuff in one specification'* is fixed in WSRF by partitioning OGSI functionality into a family of separate specifications that allow flexible composition. The second step also includes better exploitation of other Web Services specifications ([**WS-Refactor**]). This aim also focuses on the OGSI critique *'does not work well with existing Web Services and XML tooling'*. Therefore, WSRF uses standard XML schema mechanisms ([**XML-Schema**]) that are familiar to developers and are also supported by existing tools. Furthermore, the framework uses only WSDL 1.1 to associate the XML schema information model of the resources with the Web Service operations. Hence, there is no need for extending the WSDL 1.1 definition, like in OGSI. This also leads to address the OGSI critique *'Introduction of forthcoming WSDL 2.0 capability as unsupported extensions to WSDL 1.1 '*. Since WSRF expresses the capabilities of OGSI with WSDL 1.1. Another example of better exploitation of other Web Services specifications is the use of **[WS-Adressing]** in WSRF. Instead of using a proprietary Grid Service reference, WSRF uses the generic EPR to identify a stateful resource associated with the Web Service at the designated endpoint.

The last step in the evolution process is to provide a broader view of the notification model that is a general Web Service requirement upon which state change notification can be built ([**WS-Refactor**]). Hence, notification is a broad concept and not all events relate to changes in the state of a resource. As a consequence the notification model is not a part of WSRF. However, the functionality of the WS-Notification family can be build on top of WSRF.

4.3.2 The idea beyond WSRF

The basic idea beyond WSRF is a straightforward refactoring of the concepts and interfaces developed in OGSI in a manner that exploits recent developments in Web Services architecture ([**WSRF**]). It is important to know that WSRF uses the WS-Resource approach that is modeling

the state of resources in the context of Web Services([**WS-ModelingResources**]). Furthermore, WSRF is defined by five normative specifications as shown in table 4.1

Name	Description
WS-ResourceLifetime	Mechanisms for WS-Resource destruction, including message exchanges that allow a requestor to destroy a WS-Resource, either immediately or by using a time-based scheduled resource termination mechanism.
WS-ResourceProperties	Definition of a WS-Resource, and mechanisms for retrieving, changing, and deleting WS-Resource properties.
WS-RenewableReferences	A conventional decoration of a WS-Addressing endpoint reference with policy information needed to retrieve an updated version of an endpoint reference when it becomes invalid.
WS-ServiceGroup	An interface to heterogeneous by-reference collections of Web services.
WS-BaseFaults	A base fault XML type for use when returning faults in a Web services message exchange.

Table 4.1: The WS-Resource Framework is defined by five normative specifications.

Stateful Web Service Resources

The WS-Resource construct is an approach to model stateful resources in a Web Services framework. More specifically the WS-Resource is composed of a Web Service and a stateful resource, located on the server side. This stateful resource can be created and destroyed and is used in the execution of Web Services message exchanges. Furthermore, the definition of a stateful resource can be associated with the interface descriptions of a Web Service to enable well-formed queries against the state of a WS-Resource. This state can be also modified using Web Service message exchanges ([**WS-ModelingResources**]).

A stateful resource is defined to have a specific set of state data expressible as an XML document. The resource can be accessible through different Web Services sharing the same underlying resource. Furthermore, resources have a well-defined lifecycle. Suppose a stateful resource that models files in a file system, rows in a relational database, encapsulated objects such as Enterprise Java Beans (EJB), or a collection of other stateful resources. It is important to know that the WS-Resource approach concerns how a stateful resource is modeled, not how it is implemented or represented ([**WS-ModelingResources**]). As a consequence, the implementation can be a XML document, describing the resource properties, stored in memory or in a database. Alternatively, the same stateful resource can be implemented as a logical projection over data constructed from specific programming language objects such as J2EE EJB.

The WS-Resource approach also defines a specific form of stateful resource identity that can be used by one or more service implementations to identify the stateful resource used in the execution of a given message exchange ([**WS-ModelingResources**]). This specific relationship between a Web Service and one or more stateful resources is called the *'implied resource pattern'*.

Definition
*The implied resource pattern refers to the mechanisms used to associate a stateful resource with the execution of message exchanges implemented by a Web Service ([**WS-ModelingResources**]).*

The term implied is used because the stateful resource is implicitly associated with the execution of the message exchange. The requestor does not provide the stateful resource identifier as an explicit parameter in the body of the request message. The term pattern is used to indicate that the relationship between Web Services and stateful resources is codified by a set of conventions on existing Web Services technologies, in particular XML, WSDL and WS-Addressing ([**WS-ModelingResources**]). As defined in [**WS-Adressing**], an endpoint reference is an XML serialization of a network-wide pointer that represents the address of a Web Service deployed at a given network endpoint. Note that this endpoint reference can also contain, in addition to the native endpoint identification, other meta-data associated with the Web Service such as service description information and reference properties. Reference properties within the endpoint reference play an important role in the implied resource pattern.

Definition
*When a stateful resource is associated with a Web Service and participates in the implied resource pattern, one can refer to the component resulting from the composition of the Web Service and the stateful resource as a WS-Resource ([**WS-ModelingResources**]).*

WS-Addressing provides the EPR component that can include information about a stateful resource identifier such as an `objectID`. This identifier represents the stateful resource to be used in the execution of the requested message ([**WS-Adressing**]). As a consequence, the WSRF calls such an EPR a WS-Resource qualified endpoint reference.

Web Services Resource Lifetime

The lifecycle of a WS-Resource is defined as the period between its instantiation and its destruction. The specification of [**WS-ResourceLifetime**] focuses on how Web Services can be destroyed. A service requestor's interest in a WS-Resource is often limited in time. The lifetime is typically not indefinite, therefore the client should have the possibility to destroy the WS-Resource immediately or after a defined period of time. The WS-ResourceLifetime specification includes both. The immediate destruction of a WS-Resource can be accomplished by using the `Destroy` operation defined in the WSDL file of the [**WS-ResourceLifetime**] specification. Figure 4.9 shows the portType `ImmediateResourceTermination`, including the `Destroy` operation.

In a distributed environment, the client can become disconnected from the service provider. Hence, the client cannot reach the endpoint of the service provider to cause an immediate destruction of the WS-Resource. Therefore the [**WS-ResourceLifetime**] specification defines the `SetTerminationTime` operation by which any client of a WS-Resource may establish and extend a scheduled termination time of a WS-Resource. When this time expires, the WS-Resource performs a self-destruction without the need for an explicit destroy request message from the client. Figure 4.10 shows the portType `ScheduledResourceTermination`, including the `SetTerminationTime` operation.

```
<wsdl:definitions
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsrl=
    "http://www.ibm.com/xmlns/stdwip/Web-services/WS-ResourceLifetime"
  targetNamespace=
    "http://www.ibm.com/xmlns/stdwip/Web-services/WS-ResourceLifetime">
  ...
  <wsdl:portType name="ImmediateResourceTermination">
    <wsdl:operation name="Destroy">
      <wsdl:input message="wsrl:DestroyRequest" />
      <wsdl:output message="wsrl:DestroyResponse" />
      <wsdl:fault name="UnknownResource"
                  message="wsrl:UnknownResourceFault" />
      <wsdl:fault name="UnableToDestroyResource"
                  message="wsrl:UnableToDestroyResourceFault" />
    </wsdl:operation>
  </wsdl:portType>
  ...
</wsdl:definitions>
```

Figure 4.9: WS-ResourceLifetime portType `ImmediateResourceTermination`.

```
<wsdl:definitions
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsrl=
    "http://www.ibm.com/xmlns/stdwip/Web-services/WS-ResourceLifetime"
  targetNamespace=
    "http://www.ibm.com/xmlns/stdwip/Web-services/WS-ResourceLifetime">
  ...
  <wsdl:portType name="ScheduledResourceTermination"
     wsrp:ResourceProperties ="wsrl:ScheduledResourceTerminationRP">
    <wsdl:operation name="SetTerminationTime">
      <wsdl:input message="wsrl:SetTerminationTimeRequest" />
      <wsdl:output message="wsrl:SetTerminationTimeResponse" />
      <wsdl:fault name="UnknownResource"
                  message="wsrl:UnknownResourceFault" />
      <wsdl:fault name="TerminationTimeChangeRejected"
                  message="wsrl:TerminationTimeChangeRejectedFault" />
      <wsdl:fault name="UnableToSetTerminationTime"
                  message="wsrl:UnableToSetTerminationTimeFault" />
    </wsdl:operation>
  </wsdl:portType>
  ...
</wsdl:definitions>
```

Figure 4.10: WS-ResourceLifetime portType `ScheduledResourceTermination`.

Web Services Resource Properties

The [**WS-ResourceProperties**] specification defines how properties of a WS-Resource are declared as part of the Web Service interface. This declaration of the properties represents a projection of the WS-Resource's state. Therefore this projection is defined in terms of a resource

properties document that defines a basis for access to the resource properties through a Web Service interface. As shown in the example in Figure 4.11 the resource properties document type is associated with a service specific WSDL portType definition. This provides the declaration of the exposed resource properties of the WS-Resource. Note that the specification does not define the means by which a service implements a resource properties document. As a consequence, the implementation of the resource properties inside the WS-Resource can be an XML document, stored in memory or in a database. Other service implementations can dynamically construct the resource property elements and their values from data held in programming language objects such as J2EE EJB.

```
<wsdl:definitions xmlns:tns="http://example.com/diskDrive"...>
  ...
  <wsdl:types>
    <xsd:schema targetNamespace="http://example.com/diskDrive" ... >
      <!-- Resource property element declarations -->
      <xsd:element name="NumberOfBlocks" type="xsd:integer"/>
      <xsd:element name="BlockSize" type="xsd:integer" />
      <!-- Resource properties document declaration -->
      <xsd:element name="GenericDiskDriveProperties">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element ref="tns:NumberOfBlocks"/>
            <xsd:element ref="tns:BlockSize" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </wsdl:types>

  <!-- Association of resource properties document to a portType -->
  <wsdl:portType name="GenericDiskDrive"
     wsrp:ResourceProperties="tns:GenericDiskDriveProperties" >
    <operation name="start"/>
    <operation name="stop" />
    ...
  </wsdl:portType>
</wsdl:definitions>
```

Figure 4.11: Resource properties document and its association with a WSDL portType.

Furthermore, the **[WS-ResourceProperties]** specification defines a standard set of message exchanges that allow a requestor to query or update the resource properties. Important is the fact that the set of properties defined in the mentioned resource properties document defines the constraints on the valid contents of these message exchanges. However, there are several operations on resource properties defined, but only the `GetResourceProperty` operation is required when working with resource properties document type declarations. This operation can be used to obtain one single property of a WS-Resource. On the other hand, the `GetMultipleResourceProperties` operation allows a requestor to retrieve the values of multiple resource properties of a WS-Resource. Note that the properties of a WS-Resource must not be read only. The `SetResourceProperties` operation allows the processing of a single request message to make multiple changes to the resource properties document. These

changes can be an insert operation, wherein a new resource property element is inserted into the resource properties document, or a delete operation, wherein an existing resource property element is removed. Furthermore, `SetResourceProperties` supports an update operation, wherein existing resource property elements can be modified.

In addition, the **[WS-ResourceProperties]** specification supports a message exchange that allows a requestor to query the resource properties document of a WS-Resource. This operation is called `QueryResourceProperties` and uses a query expression such as XPATH ([**XPath**]) to query the resource properties. Finally, Figure 4.12 summarizes all the portTypes with their operations as defined in the official WSDL files of the **[WS-ResourceProperties]** specification.

```
<wsdl:definitions
  xmlns:wsrp=
  "http://www.ibm.com/xmlns/stdwip/Web-services/WS-ResourceProperties"
  targetNamespace=
  "http://www.ibm.com/xmlns/stdwip/Web-services/WS-ResourceProperties">
  ...
  <wsdl:portType name="GetResourceProperty">
    <wsdl:operation name="GetResourceProperty">...</wsdl:operation>
  </wsdl:portType>

  <wsdl:portType name="GetMultipleResourceProperties">
    <wsdl:operation name="GetMultipleResourceProperties">
      ...
    </wsdl:operation>
  </wsdl:portType>

  <wsdl:portType name="SetResourceProperties">
    <wsdl:operation name="SetResourceProperties">...</wsdl:operation>
  </wsdl:portType>

  <wsdl:portType name="QueryResourceProperties">
    <wsdl:operation name="QueryResourceProperties">...</wsdl:operation>
  </wsdl:portType>
  ...
</wsdl:definition>
```

Figure 4.12: WS-ResourceProperties portTypes with operations.

Web Services Renewable References

The specification of WS-RenewableReferences is not yet released. For instance, the specification will aim at the process when a service provider wants to move a WS-Resource between hosts. As a consequence, the EPR that is addressing the WS-Resource becomes invalid. WS-RenewableReferences should provide a mechanism for the client to find the new address of the WS-Resource. Furthermore, an EPR can contain not only addressing but also policy information concerning interactions with the service. Typically, an EPR is constructed by an authorized source of the addressing and policy information ([**WSRF**]). An EPR made available to a client represents a copy of that information. Hence, if the authoritative source changes the policy assertions that are governing the message exchanges with the service, the EPR of the client become inconsistent. Therefore, WS-RenewableReferences specification should provide a mechanism to renew an EPR. To conclude, WS-RenewableReferences should define a specific

policy assertion for the purpose of decorating an EPR with information necessary to retrieve a new EPR in the event the reference becomes invalid ([**WSRF**]).

Web Services Service Groups

The [**WS-ServiceGroup**] specification defines how Web Services and WS-Resources can be aggregated or grouped together for a domain-specific purpose. The idea is that requestors can form meaningful queries against the contents of the service group. As a consequence, the membership in the group must be constrained in some fashion. The constraints for membership are expressed by intension using a classification mechanism ([**WS-ServiceGroup**]). Furthermore, all members of each intension must share a common set of information over which queries can be expressed. [**WS-ServiceGroup**] uses the WSRF resource property model ([**WS-ResourceProperties**]) to express membership rules and membership constraints. Service groups are defined as a collection of members that meet the constraints of a correspondent group. To conclude, a service group is a WS-Resource that represents a collection of other Web Services. More information about grouping services can be found in [**WS-ServiceGroup**].

Web Services Base Faults

Designer of Web Services often use interfaces that are defined by other Web Service developers. Therefore, managing faults in Web Services is difficult when each interface uses a different convention for representing common information in fault messages ([**WS-BaseFaults**]). But when the information available in faults from various interfaces is consistent, it is easier for service requestors to understand faults. Furthermore, it is then possible that common tooling can be created to assist in the handling of faults. The aim of [**WS-BaseFaults**] is to specify Web Services fault messages in a common way to allow better support for problem determination and fault management. As a consequence, the specification defines an XML schema type for a base fault and rules for how base faults can be used in Web Services. More information about the base fault type can be found in [**WS-BaseFaults**].

4.3.3 WS-Notification on top of the WSRF

A commonly used pattern for inter-object communications, such as Web Services, is the event-driven or notification-based interaction pattern, also called notification pattern. Examples exist in many domains, for example in publish/subscribe systems provided by message oriented Middleware vendors, or in systems and device management domains ([**WS-PubSub**]). The idea beyond the notification pattern is that a Web Service disseminates information to a set of other Web Services or entities, without prior knowledge of these other Web Services. This means the consumers are registered dynamically with the Web Service that is capable of distributing information. Note that the Web Service that sends the information is called the notification provider while the other Web Services that receive notifications are called notification consumers. The notification provider disseminates information by sending one-way messages to the notification consumers that are registered to receive the correspondent information. The notification pattern also includes the possibility that more than one notification consumer can register to consume the same information. Hence, each notification consumer receives a separate copy of the pieces of information. Finally, the term WS-Notification is used to refer to a whole family of specifications, related to the notification pattern. Table 4.2 shows an overview of these specifications. Note that WS-Notification can be built on top of the WSRF, by creating subscriptions to state changes within the WS-Resource to monitor and report on property changes.

Name	Description
Web Services Base Notification	Defines the Web Services interfaces for notification producers and notification consumers.
Web Services Topics	Defines a mechanism to organize and categorize items of interest for subscription known as topics .
Web Services Brokered Notification	Defines the Web Services interface for the NotificationBroker that is an intermediary which allows publication of messages from entities that are not themselves service providers.

Table 4.2: The WS-Notification family consists of three specifications.

Web Services Base Notification

The first specification of the WS-Notification family is called WS-Base Notification and is the base specification on which all the other specifications in the family depend. WS-Base Notification defines the interfaces for the notification roles. That includes standard message exchanges to be implemented by service providers that wish to act in these roles.

As defined in [**WS-Base Notification**], a notification producer accepts incoming subscribe requests and each subscription identifies one or more topics of interest and a reference to a notification consumer. Hence, a subscriber sends a `Subscribe` message to a notification producer in order to register the interest of a notification consumer for notification messages related to topics. A notification producer maintains a list of subscriptions while each entry contains the topics of interest and a reference to the notification consumer. Furthermore, it maintains additional information if needed. Hence, if there is a notification to distribute, the notification producer uses this list to match the notification against the interest registered in each subscription and notifies the consumers associated with that subscription ([**WS-Base Notification**]). A notification producer uses a `Notify` message addressed to the notification consumer in order to deliver one or more notification messages. These messages requires a Web Service hosting on the client to receive the messages. Normally, no response is expected from the notification consumer upon receipt of the `Notify` message. Note that topic classifications and topic expressions are defined in another specification of the WS-Notification family, called WS-Topics ([**WS-Topics**]). However, WS-Base Notification defined a configuration where a notification consumer is subscribed directly to the notification provider. This configuration is called as the direct or point-to-point notification. Note that there is another configuration, where the notification consumer is subscribed to an intermediary notification broker service. This configuration is defined in [**WS-Brokered Notification**]. Both specifications based on the WS-Base Notification specification. More pieces of information, for instance the XML structure of the `Subscribe` and `Notify` methods can be found in [**WS-Base Notification**].

Web Services Topics

The [**WS-Topics**] specification defines a mechanism to organize and categorize items of interest for subscription known as topics. Three topic expression dialects are defined that can be used as subscription expressions in a `Subscribe` request message. Furthermore, WS-Topics specifies an XML model for describing meta-data associated with topics. To understand WS-Topics there are two roles beside the notification producers and notification consumers roles. The role of the publisher creates notification messages. Hence, a Web Service that implements the message exchanges associated with notification producer can be a publisher if it creates the notification messages itself. On the other hand, the notification producer can be a notification broker that distributes notification messages that were produced by a separate publisher ([**WS-PubSub**]). The other role is called subscriber that acts as a service requestor to send a `Subscribe` request message to the notification producer. It is possible that a subscriber is different from the notification consumer, which actually receives the notification messages. However, a collection of related topics is used to organize and categorize a set of notification messages that provides the convenient means by which subscribers can reason about notifications of interest ([**WS-Topics**]). The publisher associates a notification message with one ore more topics. In addition, the subscriber associates a subscription with one or more topics too. As a consequence, the list of all subscribers related to topics can be used if the notification provider plans to send a notification message. A notification message is delivered to a notification consumer if the list of topics associated with the subscription has a nonempty intersection with the list of topics associated with the notification message ([**WS-Topics**]).

Every topic is assigned to a separate XML namespace to avoid naming collisions. This facilitates inter-operation between independently developed notification parties. [**WS-Topics**] specifies a topic space as a set of topics associated with one given XML namespace. It is important to know that WS-Topics follows up an hierarchical approach. The outcome is that each topic can have child topics while a child topic can contain further child topics. A root topic is a topic without a parent and all descendents from such a root topic build a topic tree. A topic space is thus a collection of topic trees. Such an approach allows subscribers to subscribe against multiple topics to reduce the number of subscription requests if it is interested in a large sub-tree. However, a topic space contains additional meta-data related to its member topics that can be defined as an XML document. All root topics must have unique names within their topic spaces ([**WS-Topics**]). As a consequence, a root topic can be uniquely referenced by the XML namespace associated with the topic space and the unique name of the root topic. Note that child topics can be refered relatively to their ancestor root topics using a topic expression dialect. More information about such dialects or more detailed information related to topics can be found in [**WS-Topics**].

Web Services Brokered Notification

A notification broker is an intermediary between the two notification parties. The broker allows publication of messages from entities that are not themselves service providers. The Web Service interface of such a notification broker is defined in [**WS-Brokered Notification**]. The specification includes standard message exchanges that can be implemented by a notification broker service. Furthermore, [**WS-Brokered Notification**] defines operational requirements expected of service providers and service requestors that participate in brokered notifications. A notification broker is responsible for disseminating messages produced by one or more publishers to zero or more notification consumers. Furthermore, a notification broker can pause its subscription whenever there are no active subscribers for the topics provided by the publisher.

Certainly, if the broker has active subscribers, it is obliged to resume the subscription to the publisher. The detailed definition of a set of message exchanges to describe the function of a broker can be found in [**WS-Brokered Notification**].

4.4 Conclusion

Using WSRF to implement the case study, seems to be the best perspective for now. The well-defined specifications of the framework and its support for new Web Services techniques like WS-Addressing indicate it importance for the future. Furthermore, the WS-Notification family allows many functionality that can be built on top of *'WSRF compliant services'*. Even if WSRF is not a standard yet, it will substitute OGSI in the future. Note that this thesis is only a snapshot of currently available specifications of work that is in progress. Organization for the Advancement of Structured Information Standards (OASIS) is an international consortium that drives the development, convergence, and adoption of e-business standards, like W3C or others standardization organizations ([**OASIS-Home**]). The purpose of the OASIS WSRF Technical Commitee ([**OASIS-WSRF**]) is to define a generic and open framework for modeling and accessing stateful resources using Web Services. The aim is to standardize mechanisms to describe views on the WS-Resources state, to support management of the state through properties associated with the Web Service, and to describe how these mechanisms are extensible to groups of Web Services.

Chapter 5

WSRF Hosting Environments

The creation of *'WSRF compliant Grid Services'* relies on the use of a WSRF hosting environment. This chapter analyzes the requirements of an advantageous hosting environment. Furthermore, an overview on existing WSRF hosting environments is given.

5.1 Requirements

5.1.1 Hosting Environments

Grid Services and their stateful resources are instantiated within a specific execution environment, named hosting environment. A hosting environment defines:

- the implementation programming model
- a particular programming language
- development and debugging tools

In particular, a hosting environment defines a framework for instantiating components adhering to environment-defined interface standards ([**Physiology**]). These components can be composed to build complex applications. Furthermore, hosting environments offer superior programmability, manageability, flexibility, and safety. Therefore, they are suited for building e-business services. In the OGSA context, the hosting environment has the responsibility for ensuring that the services it supports adhere to the semantics of a Grid Service. In this case, the hosting environment has the responsibility to support WSRF semantics. In addition, a hosting environment provides traditional Web server functionality such as transport mechanisms, for instance HTTP.

Definition
*A hosting environment addresses the mapping of Grid-wide names (references and handles) into implementation specific entities (C pointers, Java object references). Furthermore, a hosting environment is responsible for the dispatch of Grid invocations and notification events into implementation-specific actions (events, procedure calls). In addition, it performs the protocol processing and the formatting of data for network transmission. Finally, the lifetime management of Grid Services resources is controlled by a hosting environment.([**Physiology**]).*

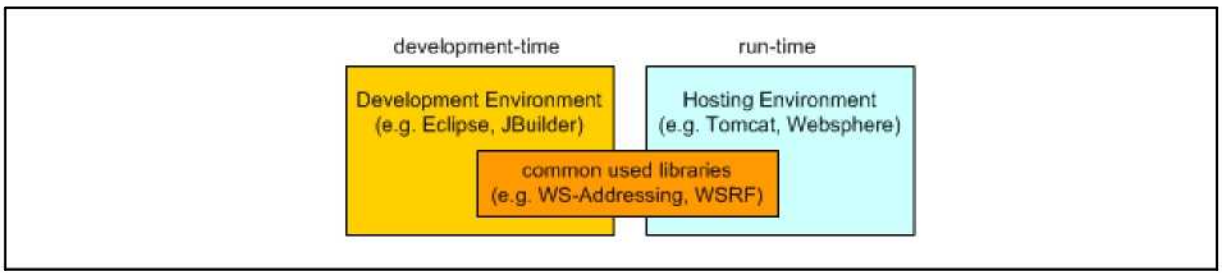

Figure 5.1: Relationship between hosting and development environment.

The term hosting environment is often confused with the term development environment. Figure 5.1 shows a simplified picture of the relationship between a development environment, libraries and a hosting environment. Note that a development environment such as Borland JBuilder or the Eclipse Platform is something different. Development environments aim at aiding the developer in implementing services. A simple hosting environment, on the other hand, is a set of resources located within a single administrative domain that is supporting facilities for service management ([**Physiology**]). As a consequence, both hosting and development environments need to have access to the same libraries that are used by a service. However, there are many synonyms for the term hosting environment as shown in the following enumeration:

- application server
- execution environment
- (Web Service) container

Finally, table 5.1 provides an overview of common hosting environments.

Name	**Producer**
Tomcat	Apache
JBoss	JBoss Incorporated
.NET	Microsoft
One	Sun
WebSphere	IBM

Table 5.1: Common hosting environments.

5.1.2 SOAP Engine

Beside a hosting environment Web Services require some infrastructure to manage SOAP compliant service requests. This infrastructure is called a SOAP engine, SOAP processor or also Web Services runtime. On the server side, the SOAP engine takes an incoming request and determines which Web Service to invoke. The SOAP engine uses the data of the SOAP message for invocation of the service and send the results back to the client ([**WS-DEVETTK**]). The following enumerations shows the key aspects of a SOAP engine:

- Support for designated programming languages.
- Tools available for the creation of services.
- The concept how services are managed and deployed.
- Concepts for advanced performance.

Figure 5.2 shows a simplified picture of the relationship between a hosting environment, libraries and a SOAP engine. Note that a SOAP engine is only a small part of the hosting environment that supports functionality for Web Services.

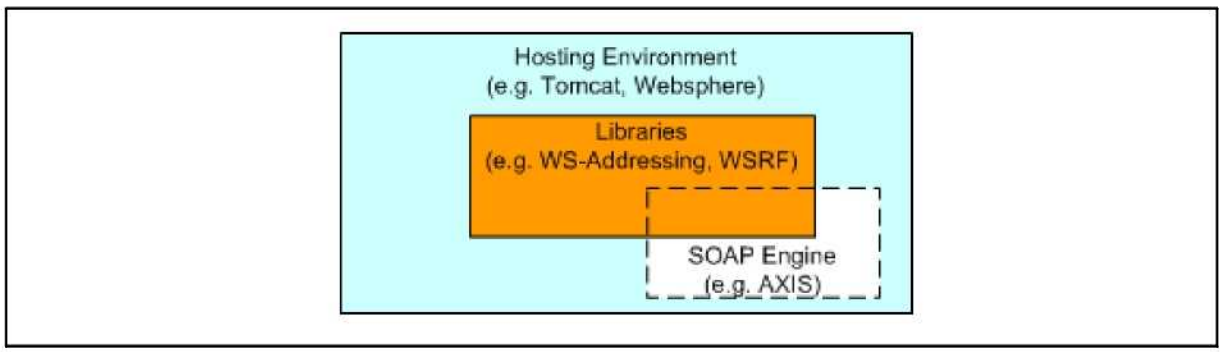

Figure 5.2: Hosting environment, SOAP engine and libraries.

To give an example, the open-source Apache Apache eXtensible Interaction System (AXIS) represents such a SOAP engine. According to **[WS-DEVETTK]**, AXIS is probably the most widely used Java based SOAP engine currently available. AXIS provides an implementation of the SOAP protocol (**[AXIS-Home]**). Therefore, AXIS can be seen as a framework for constructing SOAP compliant clients and servers. The current version of AXIS is written in Java, but a C++ implementation of the client side of AXIS is being developed. According to **[AXIS-User]**, AXIS is not only a SOAP engine, because it also includes:

- a simple stand-alone server
- a server which plugs into Servlet engines such as Apache Tomcat
- emitter tooling that generates classes from WSDL
- sample programs and documentation
- a tool for monitoring TCP/IP packets, called `TCPMonitor`

To conclude, AXIS can be seen as a SOAP toolkit.

One of the key concepts of AXIS is the incredible pluggability it offers its users. The AXIS engine allows the replacement of many predefined functionality by customized components. The most important component is a handler. Handlers allow a cleanly separation of the business logic from the SOAP processing logic. To give a small example, adding security to a SOAP message transfer can be only the configuration of a security handler. Hence, if a new security specification is defined, this handler can be replaced by a new handler that represents an implementation of the new specification. The benefit of this approach is that replacements of features have minimal impact on the business logic code and configuration. By keeping a

clean separation between the service and these handlers, the add-on features can be added or removed as needed. In addition, third party handlers can be plugged into the configuration without any changes to the service. Handlers have access to the SOAP messages and also to all AXIS-specific data, such as configuration and administrative data. Hence, there is no restriction on what a handler can do ([**WS-DEVETTK**]). When the central AXIS processing logic runs, a series of handlers are each invoked in order. Note that this ordered series of handlers is also called the chain of handlers. The order of such chains is determined by two factors, the deployment configuration and whether the engine is a client or a server ([**AXIS-Arch**]).

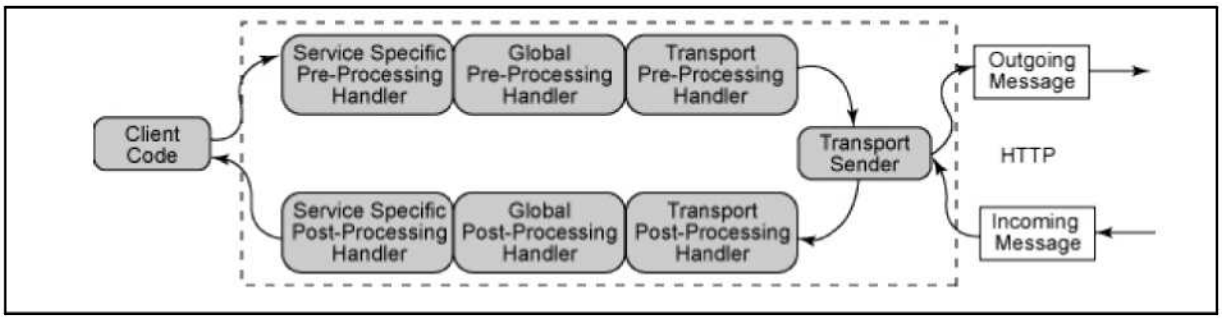

Figure 5.3: Chain of handlers on the client side ([**WS-DEVETTK**]).

Figure 5.3 shows the concept of the handlers on the client side with the use of HTTP. The client code will invoke the AXIS client that constructs an outgoing message and send it to the SOAP server. When an incoming message is received it will processed by the AXIS client that returns any results to the client code. The handlers are pluggable components through which the message will pass. Each handler can modify the message or can perform some action based on the message. Furthermore, AXIS allows to put handlers into three different stages of a message's processing flow ([**WS-DEVETTK**]):

- Transport-specific - The handler is only invoked when a certain transport is used, for instance HTTP
- Global - The handler is invoked for all services
- Service-specific - The handler is only invoked when a specific service is invoked.

The transport sender is a simple handler to send the message to the requested SOAP server or to receive messages.

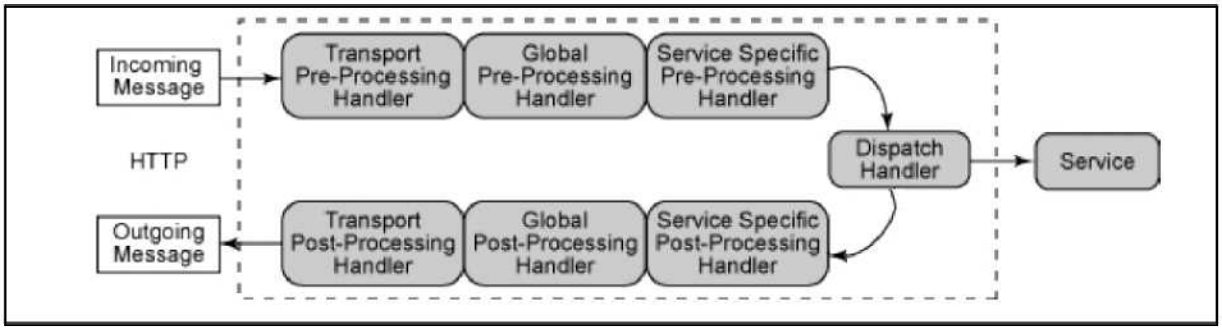

Figure 5.4: Chain of handlers on the server side ([**WS-DEVETTK**]).

Figure 5.4 shows the concept of the handlers on the server side with the use of HTTP. The AXIS server does the same things as the AXIS client, but in reverse order ([**WS-DEVETTK**]). As a consequence, the message runs through the same types of handlers. Instead of the transport sender handler, the AXIS server uses a dispatch handler to invoke the requested Web Service. More information about the advanced mechanisms of handlers and their extensibility can be found in [**AXIS-Arch**].

5.1.3 Libraries and Tooling

An advantageous hosting environment includes many libraries to support the development of *'WSRF compliant Grid Services'*. This can be accomplished by supporting Java packages or any other kind of libraries that include functionalities that are related to WSRF. For instance, a library for WS-Addressing, WS-Security or for the WS-Notification family. Such libraries include well-defined APIs that are correspondent to the well-defined specifications of the related technologies. In addition, a library is very useful if it defines, beside interfaces, also implementations of the interfaces to minimize the efforts of a developer. Hence, the support of interfaces is also important to allow the creation of domain-specific implementations of such interfaces. This libraries can be used to create *'WSRF compliant Grid Services'* at development-time. In addition, this libraries are used by the hosting environment at the run-time of the service.

Another important requirement is any kind of tooling that is supported by the hosting environment. This tooling can be:

- support for special functionalities, for instance an AXIS handler that provides support for WS-Addressing or WS-Security.
- support for code generation, for instance stub or skeleton generators that create complicated transportation functionality.

The aim of such tooling is to minimize the efforts of a developer. Furthermore, such tools can be used thousand times and they work every time flawless. Instead of human developers that are less flawless.

To give a small example, AXIS provides a WSDL tools subsystem that contains tools like `WSDL2Java` and `Java2WSDL`. Note that the AXIS engine does not depend on these tools. They are just to make life easier for the developers of services or clients. However, the `WSDL2Java` is an emitter tool that generates Java classes from WSDL that can be used to access the correspondent Web Service. The `Java2WSDL` tool on the other hand takes a Java interface and generates a WSDL file, including all operations that are defined inside the Java interface. More information about the configuration of this tools and their possibilities can be found in [**AXIS-Ref**].

5.2 Globus Toolkit

The Globus Alliance conducts research and development to create fundamental Grid technologies. According to [**Globus-Home**], the open source Globus Toolkit is a fundamental enabling technology for the Grid that lets people share computer power, databases, and other resources. The toolkit includes software services and libraries for resource monitoring, discovery and management. The toolkit also provides security functions to share resources securely across corporate and geographic boundaries. To conclude, the Globus Toolkit includes the following aspects:

- information infrastructure
- resource management
- software for security
- data management
- communication
- fault detection
- portability

The toolkit is packaged as a set of components that can either be used independently or together to develop Grid applications. The collaboration between multiple virtual organizations is often handicapped by incompatibility of resources. Therefore, the toolkit is conceived to remove obstacles such as incompatibility, to provide a better collaboration. The core services, interfaces, and protocols of the toolkit allow users to access remote resources as if they were located local. However, from version 1.0 in 1998 to the 2.0 release and the 3.0 version, the Globus Toolkit has evolved rapidly into the de facto standard for Grid computing ([**Globus-Home**]). Several large-scale e-science projects relying on the Globus Toolkit such as European Data Grid, the Network for Earthquake Engineering and Simulation (NEES) and technologies that are used by physicists that are designing the Large Hadron Collider (LHC) at Centre Européen de Recherche Nucléaire (CERN). In addition, the toolkit drawn attention from many companies, including Fujitsu, Hewlett-Packard, IBM, NEC, Oracle and SUN.

5.2.1 Globus Toolkit and OGSI

Globus Toolkit 3 (GT3) is based on a core infrastructure component that is compliant with the OGSA and is the first full-scale implementation of OGSA specifications. Important is the fact that GT3 is an open source implementation of the OGSI that was intended to serve as a proof of concept for OGSI. Furthermore, the GT3 can be seen as a reference implementation of OGSI for other implementations or related work. The core infrastructure component of the toolkit is called GT3 core. The GT3 core offers a run-time environment capable of hosting Grid Services ([**GT3-Core**]). This run-rime environment mediates between an application and an underlying network. Furthermore, GT3 core depends on other components such as a hosting environment that provides transport mechanisms. Note that GT3 ships with a lightweight embedded and standalone hosting environment, called `GlobusContainer`. But GT3 also works with standard Java Servlet Engines, e.g. Apache Tomcat, JBoss or WebSphere. GT3 core functionality also depends on the AXIS SOAP engine that is responsible for implementing XML messaging. Finally, development support in terms of programming models for the work with Grid Services are also provided by GT3. More information about the OGSI implementation of the toolkit can be found in [**GT3-Core**].

5.2.2 Future Globus Toolkit and WSRF

The WSRF represents a refactoring and evolution of OGSI that delivers essentially the same capabilities in a manner that is more aligned with Web Service technologies. Therefore, the Globus Alliance has already started work on moving the GT3 to WSRF. A WSRF-enabled version of the Globus Toolkit will be released as Globus Toolkit 4 (GT4) by modifying the OGSI based GT3. Note that according to [**Globus-Home**], the WSRF-based GT4 cannot interoperate

at the message level with GT3, because the change from OGSI to WSRF represents a change in the fundamental message exchanges and XML definitions. However, all of the capabilities embodied in the GT3 OGSI-compliant services will evolve into GT4 WSRF-compliant services ([**Globus-Home**]). A first release of GT4 final for Java is scheduled for the end of January 2005. Note that this overview of GT4 depends on a work-in-progress.

Hosting Environment

According to [**Globus-Home**], GT4 Web Services will work with the standard Apache AXIS SOAP engine without any kind of modifications. In addition, the toolkit provides support for WS-Addressing and WS-Security support. To implement WSRF and WS-Notification services, the toolkit also provides a set of libraries and handlers that implement WSRF and WS-Notification behaviors. It is important to know that these libraries and handlers plug into standard AXIS. As a consequence, this architecture makes it easy to write a single AXIS service that can interface efficiently with many resources. Note that GT4 will also shipped with an lightweight hosting environment like the `GlobusContainer` within GT3.

Installation efforts

The installation of GT4 is quite simple. There are only some prerequisites to respect such as a Java Java Developer Kit (JDK) in version 1.3 or higher. Furthermore, the Apache Ant tool is needed as well as JUnit to run tests on the services. Note that the installation of some development releases of actual toolkit demand certificates of the old GT3. Therefore, a completely new Globus developer must install first GT3 to get certificates from it. However, a development release of the toolkit which size is approximately 26 MB can be downloaded at [**Globus-Home**]. The actual version of the latest toolkit is called Globus Java WSRF Core 3.9.2, but the final release will be called Globus Toolkit 4.

WS-Addressing

The WSRF uses the Web Service standard WS-Addressing for addressing WS-Resources and services. Therefore, the implementation of GT4 uses the WS-Addressing implementation of the Apache Addressing library that is published by IBM, Microsoft and BEA as a joint-specification on top of Apache AXIS ([**Apache-Addressing**]). The library uses the handler functionality of AXIS, therefore most of the work is done in the `AddressingHandler`. To give a small example, Figure 5.5 shows the creation of an EPR, including the use of `ReferenceProperties`. Remember, that `ReferenceProperties` inside the EPR are used to carry domain-specific WS-Resource identity information, for instance a `ResourceKey`. Normally the WS-Resource identity can be anything as long as it serializes as an XML element, but Apache Addressing only allows `DOM` or a `SOAPElement`. A `VersionServiceAddressingLocator` class can be generated by the modified AXIS `WSDL2Java` tool provided by Apache Addressing. An `AddressingLocator` class can be used to get a stub for a service by passing an EPR. More details about the use of the API can be found in [**Apache-Addressing**].

```
...
// this will be a reference property
SimpleResourceKey key = new SimpleResourceKey(Counter.KEY, "123");

ReferencePropertiesType props = new ReferencePropertiesType();

// convert to SOAPElement and add to the list
props.add(key.toSOAPElement());

EndpointReferenceType epr = new EndpointReferenceType();
epr.setAddress(
  new Address("http://localhost:8080/axis/services/Version")
);
epr.setProperties(props);

VersionServiceAddressingLocator locator =
  new VersionServiceAddressingLocator();

VersionServicePortType port = locator.getVersionPort(epr);

port.getVersion();
...
```

Figure 5.5: Creation of an EPR using Apache Addressing ([**Globus-Home**]).

Representation of Resources

In the current prerelease, the resources are managed by an object that implements the `ResourceHome` interface ([**GT4-Design**]). Note that the interface is not the representation of a resource, it is more like the management interface that provides methods that operate on a set of resources, such as resource discovery. Therefore, the `ResourceHome` interface provides methods for finding and removing resources as well as methods related to a resource key. A resource key is represented by the `ResourceKey` interface that is a combination of a key name and the actual key value. As a side-remark, this resource key can be used for the reference properties element inside an EPR as introduced in [**WS-Adressing**]. Finally, Figure 5.6 shows both interfaces.

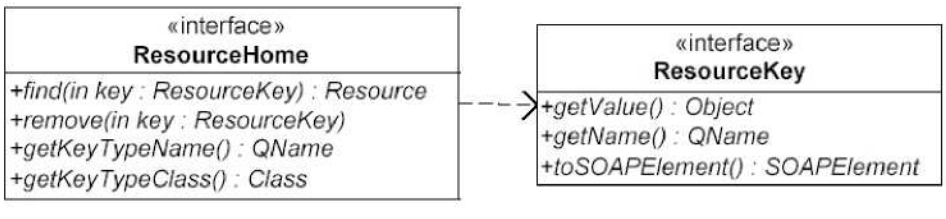

Figure 5.6: Representation of resources in GT4 ([**GT4-Design**]).

An implementation of the `ResourceHome` interface is used when an operation on a resource is invoked. Therefore, the WS-Addressing SOAP headers and the implied EPR informations need to be resolved into an actual resource instance. The resolution process, as defined in [**GT4-Design**], is as follows:

1. look up the ResourceHome instance associated with the service
2. discovering the WS-Addressing information of the SOAP header element containing the resource key
3. de-serialization of the resource key into an instance of the resource key
4. the resource key instance is used to look up the resource using the `find` method of the `ResourceHome`

All implementations that implement the `ResourceHome` interface will also provide a `create` method. Note that this method is not specified by the `ResourceHome` interface, because the signature of the method can be implementation-specific. As a consequence resources are created by invoking the `create` method on an implementation of the `ResourceHome` interface. Therefore the `ResourceHome` instance can be seen as a factory as defined in [**WSRF**]. The scheduled destruction of a resource, as defined in [**WS-ResourceLifetime**], is provided by the `ResourceLifetime` interface. The `ResourceLifetime` interface allows generic mechanisms for removing expired resources and is shown in Figure 5.7.

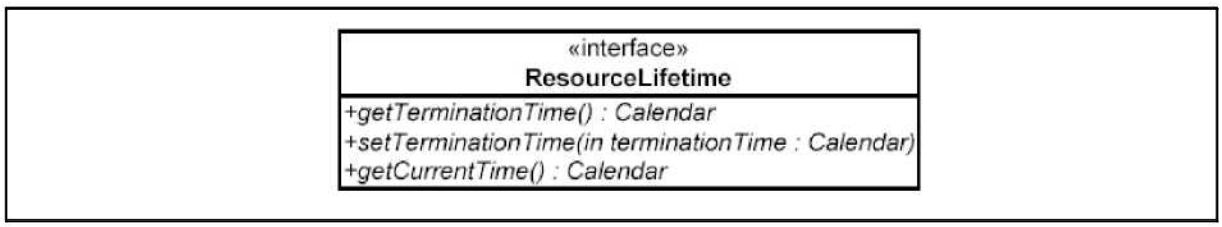

Figure 5.7: Scheduled destruction of resources in GT4 ([**GT4-Design**]).

The implementation of [**WS-ResourceProperties**] is provided by different interfaces. Within the prerelease, every resource that exposes resource properties is required to implement an interface called `ResourceProperties`. The `ResourceProperties` interface contains a single accessor method for retrieving the `ResourcePropertySet` from a resource. An implementation of the `ResourcePropertySet` interface is the representation of the resource property document associated with the resource. Remember, resource properties are declared in the WSDL file of the service as elements of a resource property document. However, the `ResourcePropertySet` interface contains methods for managing the set of resource properties, for instance the removing of resource properties. There is also an interface defined for each resource property in the set, called `ResourceProperty`. The `Resource Property` interface contains methods for managing the values associated with the resource property. Furthermore, it also contains methods for serializing the resource property to an array of SOAP or Document Object Model (DOM) elements. Finally, the `QueryEngine` interface provides a dialect-neutral mechanism for evaluating queries on implementations of the `ResourcePropertySet` interface. It determines the dialect from the query expression and calls the implementation of the `ExpressionEvaluator` registered for the dialect to perform the dialect-specific query operation ([**GT4-Design**]). Figure 5.8 shows the overview of all interfaces in GT4 related to the functionality of resource properties.

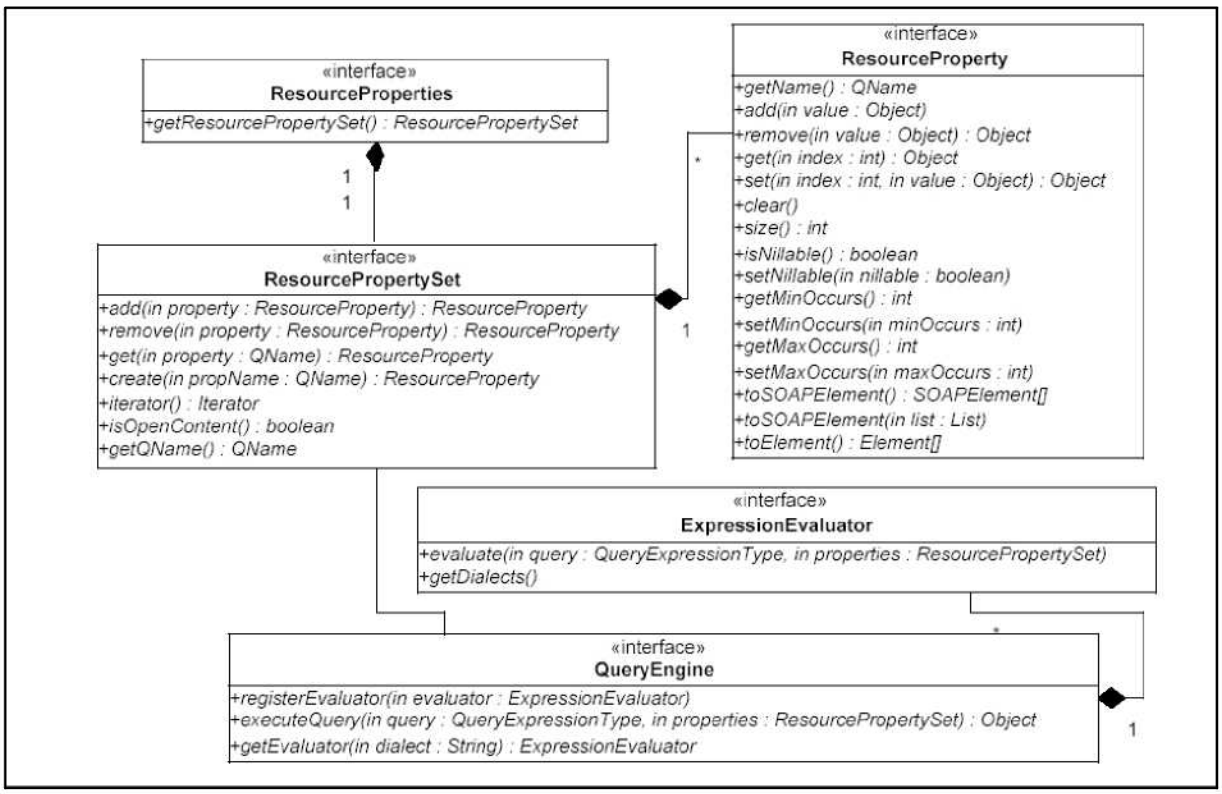

Figure 5.8: Resource properties in GT4 ([**GT4-Design**]).

Important is the fact, that loading and storing of resources will be provided within GT4. As a consequence, the `PersistentResource` interface is provided to be used by a resource that needs to be persisted to and loaded from permanent storage. Every resource that implements the `PersistentResource` interface must also define one `create` operation. While the `create` operation can be used to create new resource instances the `load` operation can be used to retrieve the resource state from persistent storage. Figure 5.9 illustrate the `PersistentResource` interface.

Figure 5.9: Persistent resources in GT4 ([**GT4-Design**]).

WS-Notification

The GT4 also provides many interfaces related to the WS-Notification family as introduced in [**WS-PubSub**]. For the work-flow of subscription, the toolkit defines an interface called `Subscription` that can be used to create a subscription resource. This created subscription resource can be associated with the topics the subscription applies to. A situation, for instance a resource property change, is represented by an object that implements the `Topic` interface. There is also and `TopicListener` interface which implementations can be used for notifi-

cation of changes in the topic value. The toolkit will also provide hierarchies of topics by defining a `TopicList` and `TopicListenerList` interface. More information about the WS-Notification family and their implementation in GT4 can be found in [**GT4-Design**].

5.3 WSRF.NET

This section provides an overview of a WSRF implementation in the .NET environment.

5.3.1 Introduction to Microsoft .NET

The Microsoft .NET Framework is an important new component of the Microsoft Windows family of operating systems. It is the foundation of the next generation of Windows-based applications ([**DotNET-Home**]). .NET allows easier building, deployment, and integration with other networked systems. Usual Windows users will not notice that the .NET Framework is running on their desktop computer, but they may appreciate the ability to connect to other systems. The aim of the .NET Framework is to establish a new approach to building software in a Windows environment. The framework provides developers with a single approach to build both desktop applications and Web-based applications. Furthermore, it enables developers to use the same tools and skills to develop software for a variety of systems ranging from hand-held smart-phones to large server installations ([**DotNET-Home**]). Therefore, the work-flow of developing applications for different systems is nearly the same. The .NET framework allows easier deployment and maintenance of conventional software and consists of two main parts: the Common Language Runtime (CLR) and the .NET Framework class library.

Common Language Runtime

The CLR is a run-time environment that manages the execution of code and provides services that simplify the development process ([**MSDN**]). In particular, the CLR is responsible for run-time services, for instance language integration and security enforcements. It also provides the infrastructure for memory, process, and thread management. The CLR aids the developer at development time with features like strong type naming, life-cycle management or cross-language exception handling. However, the CLR includes much more features that are listed in the following enumeration. Note that detailed information about each feature can be found in [**MSDN**] or [**DotNET-Home**].

- Complete object-oriented design. Everything is an object, also String or Integer data types.
- Garbage collection that manages object lifetime. Therefore reference counting is unneccesary and memory leaks are automatical eliminated.
- Cross-language integration that is possible because language compilers and tools that target the runtime use a Common Type System (CTS) defined by the runtime. That leads to a support of over 20 different programming languages.
- Self-describing objects that make using a kind of IDL unnecessary.
- Compile once and run on any operating system that supports the CLR.

.NET Framework class library

The second part of the .NET Framework is the class library that includes prepackaged sets of software functionality and components. Table 5.2 provides an overview of the three key components of the .NET Framework class library.

Name	Description
ASP.NET	Components to build Web applications and Web Services.
Windows Forms	Components to facilitate smart client user interface development.
ADO.NET	Components to help connect applications to databases.

Table 5.2: The three key components of the .NET Framework class library.

The first part, ASP.NET, is more than the next version of Active Server Pages (ASP), because ASP.NET provides a unified Web development model. An ASP.NET application can be a Web form application or a Web Service application. Web forms can be used to build form-based Web pages by using the ASP.NET server controls to create common User Interface (UI) elements ([**MSDN**]). Web forms can take full advantage of all the features of the .NET Framework. Web Services in .NET are supported by a very good infrastructure, because they are built on top of the .NET Framework and the CLR. For instance, the performance, state management, and authentication supported by ASP.NET can be seen as advantages in the process of building Web Services ([**MSDN**]).

The second part of the .NET Framework is called Windows Forms that provides components which can be used for building Windows client applications. The fundamental idea of this Windows Forms is that they utilize the CLR. Among other advantages, Windows Forms offer full support for quickly and easily connecting to Web Services. ADO.NET is the third part of the framework that provides consistent access to data sources such as databases or data sources exposed through XML. Data-sharing applications can use ADO.NET to connect to these data sources and retrieve, manipulate, and update data. Much more information about parts of the .NET Framework class library and the .NET Framework in general can be found in [**MSDN**].

5.3.2 WSRF compliant Grid Services in .NET

WSRF.NET is an implementation of the WSRF specifications on the Microsoft .NET platform. The implementation includes software libraries, tools and applications that allow easy authoring of *'WSRF compliant Grid Services'*. WSRF.NET integrates many Microsoft technologies such as Web Services Enhancements (WSE) and ADO.NET ([**WSRF.NET-Home**]). The WSE is a supported add-on to the Microsoft .NET Framework that provides developers the advanced Web Services capabilities such as message-based security or policy-based administration. ADO.NET can be used for consistent access to data sources such as Microsoft SQL Server as well as data sources exposed through XML ([**ADO.NET-Home**]). However, WSRF.NET builds upon the success of the OGSI.NET project that provides a container framework on which *'OGSI compliant Grid Services'* were created in the world of Microsoft Windows ([**OGSI.NET-Home**]).

Hosting Environment

ASP.NET is the hosting environment that enables developers to use the .NET Framework to target Web-based applications. Web Services in .NET are using the Internet Information Services (IIS) and ASP.NET as the publishing mechanism for applications. IIS is a powerful Web server that provides a highly reliable, manageable, and scalable Web application infrastructure ([**IIS-Home**]). As a consequence, the combination of IIS and the ASP.NET runtime leads to a complex hosting environment, as depicted in Figure 5.10, that is needed to create Web Services in .NET. Note that Web Services in Microsoft .NET and therefore the hosting environment uses standards such as SOAP or WSDL to promote interoperability with non-Microsoft solutions ([**MSDN**]). Finally, WSRF.NET also includes a simple, light-weight HTTP server for clients for using WS-Notifications.

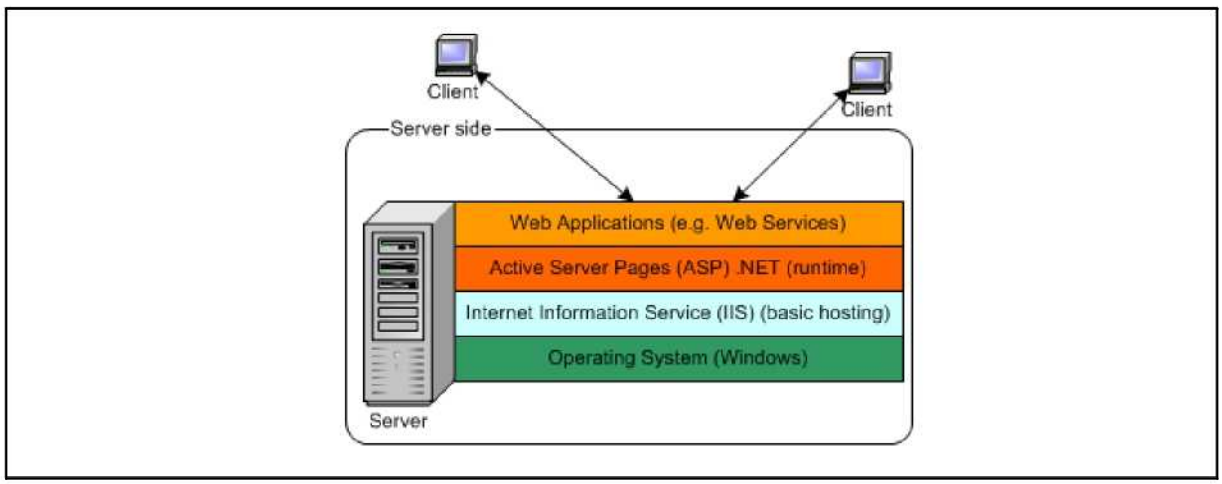

Figure 5.10: Layer model of the hosting environments needed for .NET Web Services

Installation efforts

The installation effort for WSRF.NET depends on the usual working environment of the developer. If the developer is already developing .NET Web Services solutions, the installation efforts can be reduced to a minimum. Otherwise, if a new .NET environment has to be created, there are many steps to take before beginning of building *'WSRF compliant Grid Services'*. Note that users of .NET services must only install the .NET redistributable package. The following enumeration provides an overview of the neccessary installation steps:

1. Installation and configuration of IIS
2. Installation of the .NET Framework 1.1
   (a) Redistributable Package that includes the run-time elements required to run applications based on the .NET Framework.
   (b) .NET Framework Software Development Kit (SDK) that includes tools to write, build, test, and deploy applications using the .NET Framework 1.1. Furthermore, it contains documentation, samples, command-line tools and compilers.
3. Installation of the WSE 2.0
4. Running of `setup.exe` of the WSRF.NET download bundle

5. Installation of an Open Database Connectivity (ODBC) compliant database such as MySQL or Microsoft SQL Server Desktop Engine (MSDE)

Hence, for an existing .NET environment one only needs to run the `setup.exe` of the WSRF.NET download bundle, because the usual environment for .NET is already installed. It is essential install at first the IIS and then the .NET Framework, otherwise IIS will not be integrated with the framework correctly ([**WSRF.NET-Home**]). Furthermore, the .NET Framework Redistributable Package must be installed prior to installing the .NET Framework. Finally, if all necessary steps are completed, WSRF.NET provides several services to test the installation.

WS-Addressing

WSRF.NET stores the resource in a database using an EPR of WS-Addressing. The EPR consists of an `<ReferenceProperties>` element defined by the `[ReferenceProperties]` attribute in WSRF.NET. This attribute allows the definition of a name and namespace for an XML element that will contain an unique value of a resource. The defined element will then be used in the `<ReferenceProperties>` element of the service's EPR ([**WSRF.NET-Reference**]). Figure 5.11 shows an example of a `[ReferenceProperties]` attribute. Note that more information of the internals about the realization of the [**WS-Adressing**] specification is not conveniently published yet.

```
[ReferenceProperties( TrackingNumber ,  http://shipping.com )]
public class PackageService {
  ...
}
```

Figure 5.11: Example of a `[ReferenceProperties]` attribute ([**WSRF.NET-Reference**]).

Representation of Resources

A WS-Resource is an abstraction for a collection of state that is manipulated by a particular Web Service. Therefore, WSRF.NET allows class-level data members to be declared as part of the stateful resource via the `[Resource]` attribute ([**WSRF.NET-Reference**]). An attribute in .NET is a keyword-like descriptive declaration to annotate programming elements such as types, fields, methods, and properties ([**MSDN**]). Important is the fact that these attributes can be used to describe code that affect application behavior at run time. The .NET Framework supplies many useful attributes, but there is also the possibility to design and deploy own attributes. More information about the use of attributes can be found in [**MSDN**]. A resource in WSRF.NET can be seen as the collection of values of the data members annotated with the `[Resource]` attribute. Figure 5.12 shows an example of such attributes. Two class-level data members have been added to contain the data about an agreement that will be loaded from the database, `gTerms` and `sdTerms`.

```
public class AgreementService : ServiceSkeleton
{
    [Resource]
    GuaranteeTerms gTerms;

    [Resource]
    ServiceDescriptionTerms sdTerms

    ...
}
```

Figure 5.12: Usage of the `[Resource]` attribute.

A service can have many resources, each containing a set of values and each being named by a unique identifier. The WSRF.NET architecture takes care of special database interactions to establish stateful resources as if they were class-level data members ([**WSRF.NET-Reference**]). Furthermore, that architecture approach automatically includes the persistence resources. This is accomplished by storing all resources in a database under an EPR that includes the URL of the Web Service and this unique identifier. When this EPR is used to invoke the Web Service the associated resource will be loaded from the database and the values are placed in the Web Service's `[Resource]`-annotated data members. After the invocation, all changed made to these data members are saved back to the database under the same EPR. Note that not all class-level data members must be part of the resource. Hence, only `[Resource]`- annotated data members will be automatically persisted und restored from the database.

To create a service logic that can take actions upon the stateful resources, an ASP.NET Web Service must be created. This service must be derived from the class `UVa.GCG.WSRF.Service.BaseTypes.ServiceSkeleton`. All usual .NET attributes can be used to annotate the service logic, for instance the `[WebMethod]` attribute. Attaching the `[WebMethod]` attribute to a public class method indicates that this method should be exposed as part of the Web Service. The addressable entry point for a Web Service is represented by a `.asmx` file in ASP.NET. With this annotated class method and with the class referenced in an `.asmx` file, ASP.NET can generate a complete service description using WSDL. As a side-remark, this tooling is very close related to the Apache AXIS tool `Java2WSDL`. However, Figure 5.13 shows a method that can be used to act upon the resources. The signature of the method is also a good example of the benefits of stateful resources in Web Services. Since any invocation on the agreement service will use the implied resource pattern ([**WSRF**]), there is no need to pass agreement terms as explicit parameters to the methods. The methods can use the stateful resources that are stored server-sided.

Another part of WSRF.NET that is related to the representation of resources are WS-Resource properties as defined in [**WS-ResourceProperties**]. This specification defines the description of stateful resources with an XML document called the resource property document. This document contains the exposed information about a particular resource and is composed of elements that are called resource properties. WSRF.NET uses an `[ResourceProperty]` attribute to declare resource properties. Hence, the resource property document can be thought of all the properties annotated with `[ResourceProperty]` declarations. There is one instance of a resource property document for every resource which contains the property values for that resource ([**WSRF.NET-Reference**]). The XML schema for this resource property document are automatical generated in WSRF.NET from the set of `[ResourceProperty]` attributes.

```
public class AgreementService : ServiceSkeleton
{
    [Resource]
    GuaranteeTerms gTerms;

    [Resource]
    ServiceDescriptionTerms sdTerms

    [WebMethod]
    public void clearGuaranteeTerms() {
      // possible changes of stateful resources
      ...
    }
    ...
}
```

Figure 5.13: Usage of the `[WebMethod]` attribute.

All this resource property documents are stored in a database, like the resources. Therefore the resource property documents are persistent between invocations on the correspondent Web Service. The `[ResourceProperty]` attribute can be placed on a .NET property with appropriate *'getters'* or *'setters'*. Note that in .NET, properties are a mechanism for writing accessor functions for data stored in a class's field. The `Name` property in the Web Service in Figure 5.14 shows the layout of a .NET property with a *'getter'*. As a side-remark, *'getter'* and *'setter'* methods can be interpreted like `getXYZ()` or `setXYZ()` methods in Java. Figure 5.14 also shows an example for the advanced development support, because the `[WSRFPortType]` attribute allows a Web Service to import functionality defined in other portTypes. As a consequence, all methods that are defined in the **[WS-ResourceProperties]** specification are already implemented and can be easily imported into any new service by using the `[WSRFPortType]` attribute. Note that there are also other implemented portTypes, for instance the immediate or scheduled destruction portTypes as defined in **[WS-ResourceLifetime]**. Furthermore, developers can create their own portTypes to import them into Web Services and to share central organized functionality over different Web Services. More information about using resource properties in WSRF.NET can be found in **[WSRF.NET-Reference]**.

```
[WSRFPortType(typeof(GetResourcePropertyPortType))]
public class AgreementService : ServiceSkeleton
{
    [Resource]
    GuaranteeTerms gTerms;

    [Resource]
    ServiceDescriptionTerms sdTerms

    [ResourceProperty]
    public String Name {
      get { return getName(); }
    }

    [WebMethod]
    public void clearGuaranteeTerms() {
      // possible changes of stateful resources
      ...
    }

    private String getName() {
      ...
    }
    ...
}
```

Figure 5.14: Usage of the `[ResourceProperty]` attribute.

WS-Notification

A WSRF.NET service uses an special attribute to receive notification messages on the server-side, namely `[WSRFPortType(typeof(NotificationConsumerPortType))]`. This is accomplished by using the infrastructure of both IIS and the ASP.NET framework to receive a notification message. Hence, such a notification call is just like any other invocation on a service. The more interesting question is how a notification message is received by the client-side without using the IIS. In particular, the client-side program need a light-weight hosting environment to receive notification messages. The class `UVa.GCG.WSRF.Common.HttpServer.NotificationServer` is a simple, light-weight HTTP server for clients. The creation of such a `NotificationServer` starts a new non-blocking thread. While the `start` method of the `NotificationServer` starts the listening of messages, the `stop` method stops the listening. There is also the possibility to define the address and the port where the `NotificationServer` is listening. Hence, there could be multiple `NotificationServer` with different addresses and ports. Both pieces of information are used to create an EPR when subscribing to receive notifications by the notification producer.

5.4 Emerging Technologies Toolkit

The Emerging Technologies Toolkit (ETTK) is a software development kit for designing, developing, and executing emerging autonomic and Web Service technologies ([**ETTK-Home**]). Autonomic technologies, for instance Autonomic Computing (AC) is an approach of automating the management of computing resources, but there is much more to find about this technology inside [**Autonomic**]. The ETTK provides an environment in which many emerging technology examples can be tested. These examples are related to recently announced specifications and prototypes from IBM's emerging technology development and research teams. Furthermore, the toolkit provides introductory material, tutorials and documentation about autonomic technologies and Web Services.

5.4.1 The Ideas beyond the ETTK

The ETTK evolved from a package known as Web Services Toolkit (WSTK). The WSTK for dynamic e-business was created in July 2000 to provide a simple infrastructure for creating, locating and invoking Web Services ([**WSTK**]). Sample source code, documentation, tutorials, and specifications were also supplied by the WSTK. The idea of the WSTK was to provide a showcase package for emerging technologies related to Web Services. Therefore, the WSTK aimed on supplying prototypes and technology previews. Since July 2000, several updates of the WSTK have been distributed, including new technology related to Web Services. The goal of the WSTK was to stay current with the latest Web Services technologies. For example, the WSTK was the first package to showcase an implementation of WS-Inspection as well as WS-Security ([**WSTK**]). Note that the WSTK was not a product, it was a preview technology package of emerging technologies, used by everybody who was interested. However, many functions first seen within the WSTK are now contained within IBM's product offerings ([**WSTK**]).

According to [**ETTK-Home**] the emerging technologies team has expanded the scope of the WSTK to include autonomic and Grid-related technologies. As a consequence, the WSTK was renamed to the ETTK. The ETTK will continue to offer leading-edge prototyping of emerging Web Services technologies, but will also integrate other emerging technologies from IBM research labs. Software components needed to experiment with Web Services and programs are provided. Furthermore, the toolkit includes a fully-functioning SOAP engine and an embedded application server. The ETTK can be used under Linux and Windows operating systems. The current version of the ETTK, version 2.0.1, was released at the end of April 2004. Finally, the following enumeration shows some specifications that are implemented inside the ETTK:

- WS-Addressing
- WS-Business Process Execution Language (BPEL)
- WS-Inspection
- WS-Notification
- WS-Policy
- WS-Resource Framework 1.1
- WS-Security

5.4.2 WSRF inside the ETTK

The ETTK includes an implementation of the WSRF version 1.1 . Therefore, this section provides an overview of this implementation and related implementations of specifications like WS-Addressing and WS-Notifications.

Hosting Environment

An excellent idea of the ETTK was the integration of an application server to simplify the installation and configuration process of examples. The ETTK ships with an embedded version of the IBM WebSphere Application Server - Express 5.1 that includes a WebSphere SOAP engine ([**ETTK-Doc**]). WebSphere is a software platform of products that are designed to work together to help deliver dynamic e-business quickly ([**WebSphere-Home**]). The underlying idea of WebSphere is to connect customers, systems, and applications with internal and external resources. Therefore, WebSphere is based on Middleware and infrastructure software designed for dynamic e-business. To conclude, the WebSphere product family provides a proven, secure, and reliable software portfolio and the WebSphere Application Server is one significant part of this family.

The WebSphere Application Server Express is a low-risk, affordable entry point to e-business. It is designed for end-to-end ease of use to get e-business projects up and running quickly ([**WebSphere-AppExpress**]). Many wizards are used for quick and simple installation and deployment. As a consequence, WebSphere Application Server - Express requires minimal administrative resources. Default settings help eliminate the complexities of configuring a Web server. As a side-remark, 1998 Apache reached an agreement with IBM, so that IBM includes the Apache Web Server code in its WebSphere server products ([**OpenSource**]). However, the express version supports the latest specifications for JSP, Java Servlets and Web Services. In addition it can be used on different operating system platforms such as OS/400, Linux, UNIX and Windows. WebSphere Application Server - Express allows a migration to more-advanced IBM WebSphere Application Server with no alterations. To conclude, WebSphere Application Server - Express is a simplified application server pre-configured inside the ETTK to support the demos of emerging technologies.

According to [**ETTK-WS**] the ETTK uses AXIS as SOAP engine. In particular, the support for WS-ReliableMessaging has been written into the modified version of the Apache AXIS SOAP engine that is shipped with the ETTK. Furthermore, the toolkit provides many handlers for the AXIS engine. For example, it provides two handlers related to a useful subset of [**WS-Security**]. For signing and encrypting outbound messages the `SecuritySender` handler can be used. The `SecurityReceiver` handler can be used for signature verification and decryption. The handlers provided by the toolkit can be established by inserting them into the client and server side chains as defined in [**AXIS-Arch**].

Installation efforts

The installation of the ETTK is very simple, because it can be downloaded at **[ETTK-Home]** and installed using a setup tool. The installation includes:

- documentations about emerging technologies
- many examples about emerging technologies
- the embedded application server
- libraries providing APIs of the emerging technologies

Note that before using the ETTK for implementing software, the shipped license of the ETTK should be considered.

WS-Addressing

The ETTK includes an implementation of the **[WS-Adressing]** specification. The implementation provides transport-neutral mechanisms to address Web Services and identify messages. The API that is needed to use WS-Addressing is contained within a a single Java package, namely `wsa.jar`. The API consists of two primary classes and an AXIS handler. The two primary classes are `MIHeader` and `EndpointReference` as shown in Figure 5.15.

- The `EndpointReference` class is used to represent an EPR and provides access to the fields that are specified for an EPR in **[WS-Adressing]**.
- The `MIHeader` class acts like a utility class that can read relevant WS-Addressing defined headers from a SOAP message. Furthermore, the class provides access to the fields that are specified for an MIH in **[WS-Adressing]**.

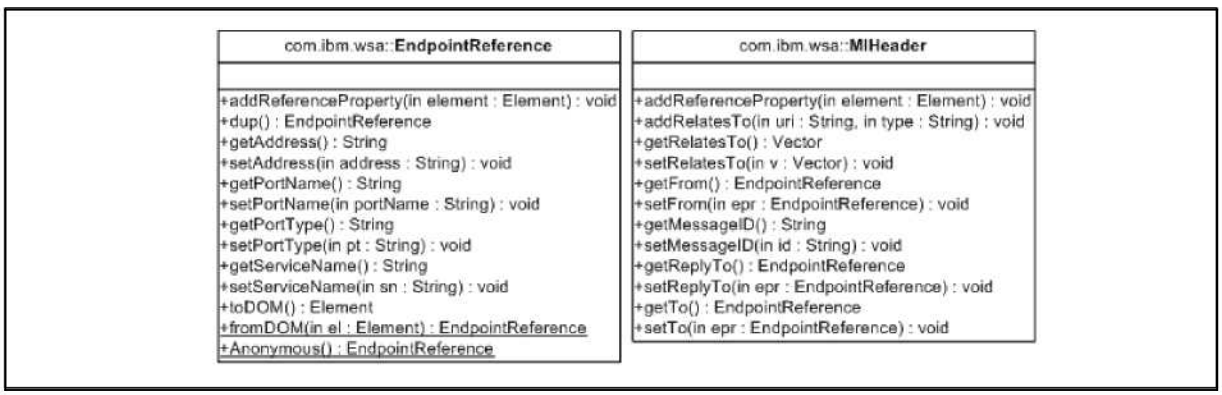

Figure 5.15: Implementation of the **[WS-Adressing]** specification.

The implementation of **[WS-Adressing]** also includes the `WSAHandler` class that provides the support to read and create WS-Addressing compliant message headers. Figure 5.16 shows how to deploy the handler in the transport chain of AXIS. If the header is deployed, messages will be routed and replied using WS-Addressing functionality. Furthermore, the `MIHeader` class can be used to access the information in the messages.

```
<handler name="WSAHandler" type="java:com.ibm.wsa.handler.WSAHandler"/>
  <transport name="http">
    <requestFlow>
      <handler type="URLMapper"/>
      <handler type="WSAHandler"/>
    </requestFlow>
    <responseFlow>
      <handler type="WSAHandler"/>
    </responseFlow>
  </transport>
</handler>
```

Figure 5.16: **[WS-Adressing]** handler for AXIS.

Representation of Resources

The ETTK implementation of the WSRF is contained within a single Java package, namely `wsrf.jar`. Note that ETTK WSRF uses the ETTK WS-Addressing implementation. Therefore, the handler for WS-Addressing must be deployed on the client and server side to enable this support. However, the implementation related to the representation of resources consists of a set of classes and interfaces. One of the important classes is the `WSResourceLifetimeManager`. The `WSResourceLifetimeManager` manages the association of EPR addressed operations to a resource. The `WSResourceLifetimeManager` is a singleton and can manage resources for multiple services (**[ETTK-Doc]**). This interface is shown in Figure 5.17.

```
public interface WSResourceLifetimeManager {
  public EndpointReference createInstance(
      Object resource,
      Date initialTerminationTime,
      URL url);

  public Object getResource();
}
```

Figure 5.17: Interface `WSResourceLifetimeManager` (**[ETTK-Doc]**).

WSRF uses a factory approach to generate stateful resources. The implementation of such a factory is using the `createInstance` method of the `WSResourceLifetimeManager` to register the resource instance. Furthermore, this method creates an EPR to represent this resource instance by using the supplied URL and by creating reference properties which uniquely identify the provided resource instance. Note that the ETTK also uses an UUID for uniquely identifying an resource instance. Therefore, reference properties are intended to be opaque from the service and client utilizing the EPR (**[ETTK-Doc]**). After the resource instance has been created, requests can be made to the stateful resource. The Web Service that is associated with the correspondent resource will receive the request and can ask the `WSResourceLifetimeManager` for the associated resource via a `getResource` method. As a consequence, the implementation of the `WSResourceLifetimeManager` accesses the EPR from the request message, performs a lookup, and retrieves the resource instance associated with this Web Service. Note that the manager also provides immediate and scheduled destruction of resources.

The ETTK uses a `Resource` interface that can be implemented by a class which represents the state of the service. All the methods that are specified within the [**WS-ResourceProperties**] specification are also defined by the `Resource` interface. To support the development of standard resources, the ETTK also provides a default implementation of resources. The internal data-model of this default implementation is a DOM tree ([**ETTK-Doc**]). On the other hand a custom resource class can be written that can support resource property requests by subclassing the default resource implementation. Alternatively, the resource property requests can be delegated to a contained instance of the default resource implementation class. To conclude, the ETTK supports advanced tooling for the creation of domain-specific resources. Finally, Figure 5.18 shows the methods of the interface `Resource`.

```
public interface Resource {
  public Element[] getResourceProperty(QName qname)
         throws ResourceUnknownFault, ...;

  public Element[] getMultipleResourceProperties(QName[] qnames)
         throws ResourceUnknownFault, ...;

  public void insertResourceProperty(Element el)
         throws ResourceUnknownFault, ...;

  public void updateResourceProperty(QName qname, Element el)
         throws ResourceUnknownFault, ...;

  public void deleteResourceProperty(QName qname)
         throws ResourceUnknownFault, ...;

  public Node[] queryResourceProperty(String expression, String dialect)
         throws ResourceUnknownFault, ...;

  public Object getDataModel();
}
```

Figure 5.18: Interface `Resource` ([**ETTK-Doc**]).

Another important class is the `ResourceClientProxy` that is an implementation of the `Resource` interface. This class can be used at the client-side to interact with a remote resource addressed by an EPR. The benefit of this class is that it provides methods for resource property operations, including the creation of appropriate messages. Finally, an important information related to the WSRF implementation is that fault messages do not conform to the WSRF specifications. Instead, usual SOAP faults are being returned.

WS-Notification

The implementation of the WS-Notification family is provided inside the Java package `wsrf.jar` within the ETTK. The notification implementation provides implementations of client side stubs for various interfaces ([**ETTK-Doc**]). This client side stubs can be used by specifying either the EPR or the URL of the endpoint being proxied. The implementation also provides skeletons for the server side. According to [**ETTK-Doc**] there is a skeleton to map to each interface defined in the WS-Notification specifications. The ETTK also includes implementations of a notification broker and notification producer. Other implementations of the interfaces can be used to create domain-specific parties in the notification process.

5.5 Other WSRF hosting environments

There are several other implementations of the WSRF. For instance, the University of Manchester provides an WSRF implementation to create Grid Services in Perl, called WSRF::Lite ([**WSRF.Lite-Home**]). *'Don't be mislead by the Lite suffix, this refers to the effort it takes to use the module, not its capabilities. '* ([**WSRF.Lite-Workshop**]). WSRF::Lite relates on experiences gained by OGSI::Lite which was formerly also developed by the University of Manchester. However, the Perl implementation of WSRF supports the implied resource pattern, WS-Resource lifetime and properties ([**WSRF.Lite-Introduction**]). Furthermore, the implementation uses WS-Addressing and can use Hypertext Transfer Protocol Secure (HTTPS) for security. Note that an implementation of WS-Security, WS-ServiceGroup, WS-BaseFault, and WS-Notification is not included in the actual version of WSRF::Lite. The state of the resources can be stored using a file or a database. The WSRF Perl services can be hosted using a stand alone script, using the WSRF::Lite Container or using Apache Tomcat. The installation of WSRF::Lite requires a number of other Perl modules, for instance XML::DOM and SOAP::Lite. WSRF:Lite ships with some test scripts for WSRF functionality.

A complete overview of all WSRF implementations is out of the scope of this thesis. Finally, table 5.3 provides an overview of some well-known implementations of the WSRF ([**WSRF-Interop**]).

Name	Language	Producer
Emerging Technologies Toolkit	Java	IBM
Globus Toolkit	Java	Globus Alliance
pyGridWare WSRF Toolkit	Python	Lawrence Berkeley National Laboratories
WSRF Implementation	Java	Indiana University
WSRF::Lite	Perl	University of Manchester
WSRF.NET	.NET	University of Virginia

Table 5.3: Different implementations of the WSRF.

5.6 Conclusion

Because a stable release of the GT4 final is scheduled for the next year, the use of the Globus implementation is out of scope here. Even if there are some beta-quality development releases planned for 2004. But it will be certainly one of the most sophisticated and comprehensive free WSRF hosting environments.

The WSRF.NET is an implementation of the WSRF in the world of Windows. An advantage of WSRF.NET is the architecture approach that provides persistent resources. Furthermore, WSRF.NET provides advanced tooling, e.g. import functionality of already implemented portType operations or automatic generations of WSDL and resource property documents. In addition, the generators, for instance the WSDL generator, can be precisely configured to guide the generation processes. To conclude, WSRF.NET is a promising implementation for platform-dependent solutions, in example for Microsoft Windows. As a side-remark, the Mono project aims to develop a .NET implementation for Linux ([**MONO**]). Hence, all platform-dependent solutions can be soon platform-independent solutions. On the other hand, WSRF.NET needs much more infrastructure to establish a WSRF hosting environment as other implementations, mainly because of all installations related to the rather complex .NET Framework. Finally, the light-weight HTTP server for the use of client-sided WS-Notification receptions is also one of the benefits of WSRF.NET.

Beside all other WSRF hosting environments the IBMs ETTK provides the sturdy and best WSRF hosting environment for now. The implementation and libraries include the main parts of the WSRF and a stable implementation of WS-Addressing. Furthermore, the hosting environment of the ETTK provides good tooling. For instance, an automatical invocation of a `WSDL2JavaEmitter` is called during the run-time of the service. This tool does nearly the same as the AXIS `WSDL2Java` tool, but instead on development-time, on run-time of the service.

Chapter 6

Realization of Flexible Service Offers in a Grid Environment

The Grid environment is an emerging distributed computing infrastructure that facilitates the routine interaction between service requestor and provider. The routine interaction relies on well-defined service interfaces that are exposed by the service provider. This chapter will concentrate on the realization of such service interfaces instead of their implementation. A full description of the implementation is out of scope of this thesis. Only very important pieces of source are mentioned that are useful to describe important parts of the implementations.

The prototypes are realizing the routine interaction between service requestor provider as defined in the [**WS-Agreement**] specification. Furthermore, the realization of prototypes based on the network reservation example, as introduced in Chapter 3. An advantageous technology framework to implement the prototypes in a Grid environment is the WSRF, as analyzed in Chapter 4. Therefore, the prototypes are realized as *'WSRF compliant Grid Services'*. Such Grid Services are realized with an WSRF hosting environment and corresponding libraries. Chapter 5 discussed that the ETTK provides an sophisticated WSRF hosting environment that will be used to implement the prototypes. Note that all prototypes are implemented in the Java programming language. Figure 6.1 provides an overview on how all components of the prototypes are working together.

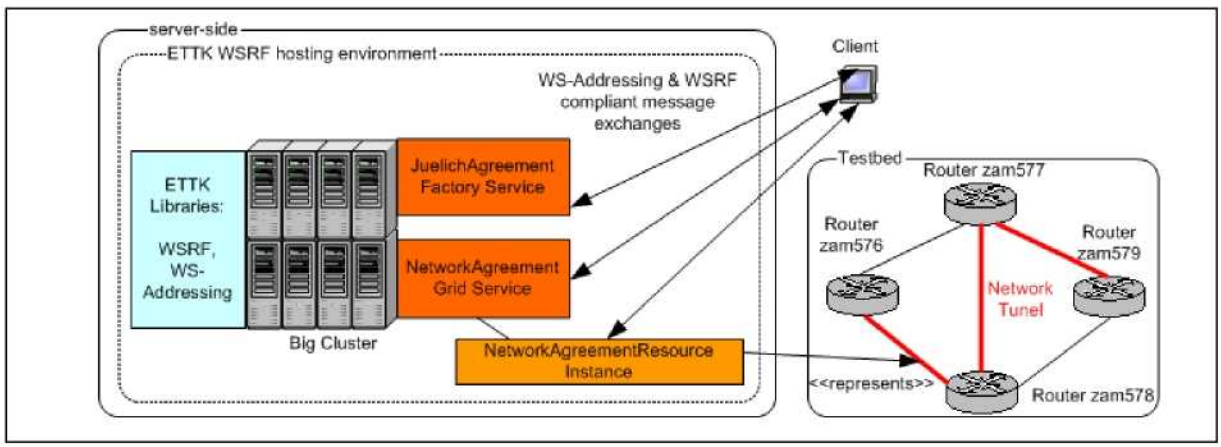

Figure 6.1: Network reservation scenario as *'WSRF compliant Grid Service'*.

Figure 6.1 shows both prototypes of services that are deployed within the ETTK hosting environment. The prototype of the `JuelichAgreementFactory` service is realized by using the *'autonomic factory approach'* that will be discussed later in this chapter. The basic idea of the prototypes is to demonstrate the power of this *'autonomic factory approach'* that is advancing the role of a factory. Therefore, this factory is modeled as an autonomic service that is responsible to create `NetworkAgreementResource` instances that can be accessed through the `NetworkAgreement` service. The intention of modeling an `NetworkAgreementResource` is to have an representation of a network tunnel that can be created inside the `Testbed`. The negotiated characteristics of the network tunnel are formal defined using the **[WS-Agreement]** compliant agreement structure.

6.1 Namespaces

The creation of service descriptions in distributed heterogenous environments is influenced by many elements that are defined in different namespaces. An XML namespace provides a method for qualifying element and attribute names used in XML documents and therefore also in WSDL documents. This is accomplished by associating the documents with namespaces that can be identified by URI references (**[XML-NS]**). Figure 6.1 shows an overview of all used namespaces in the prototypes.

Prefix	Namespace
soap	http://schemas.xmlsoap.org/wsdl/soap/
wsa	http://schemas.xmlsoap.org/ws/2003/03/addressing
wsag	http://www.ggf.org/namespaces/ws-agreement
wsbf	wsbf="http://www.ibm.com/xmlns/stdwip/Web-services/WS-BaseFaults
wsdl	http://schemas.xmlsoap.org/wsdl/
wsrl	http://www.ibm.com/xmlns/stdwip/Web-services/WS-ResourceLifetime
wsrp	http://www.ibm.com/xmlns/stdwip/Web-services/WS-ResourceProperties
wsnt	http://www.ibm.com/xmlns/stdwip/Web-services/WS-Notification
xs	http://www.w3.org/2001/XMLSchema

Table 6.1: Namespaces used in the prototypes.

6.2 Realization of a NetworkAgreement Service

The realization of the `NetworkAgreement` service is differentiated into the following aspects:

- Specification and implementation of the service.
- Specification and implementation of the WS-Resources used by this service.

The differentiated design follows the idea of the new WSRF framework: To allow easier composition between different aspects of functionality. On the one hand, this realization focuses on the implementation of the service that provides methods that are used at service invocation. On the other hand, the realization focuses on the implementation of a stateful WS-Resource that models an flexible agreement. As a consequence, the `NetworkAgreement` service implementation and the implementation of agreement resources are loosely coupled.

6.2.1 Concept

The `NetworkAgreement` service provides access and management capabilities for a number of WS-Resources that are used by different clients. In particular, the `NetworkAgreement` service is a *'WSRF compliant Grid Service'* that provides lifetime management for WS-Resources. Such a WS-Resource is a `NetworkAgreementResource` instance that represents a negotiated agreement for network reservation. A client uses an EPR of **[WS-Adressing]** to address an `NetworkAgreementResource` instance through the `NetworkAgreement` service. Figure 6.2 shows the basic concept of the `NetworkAgreement` Grid Service.

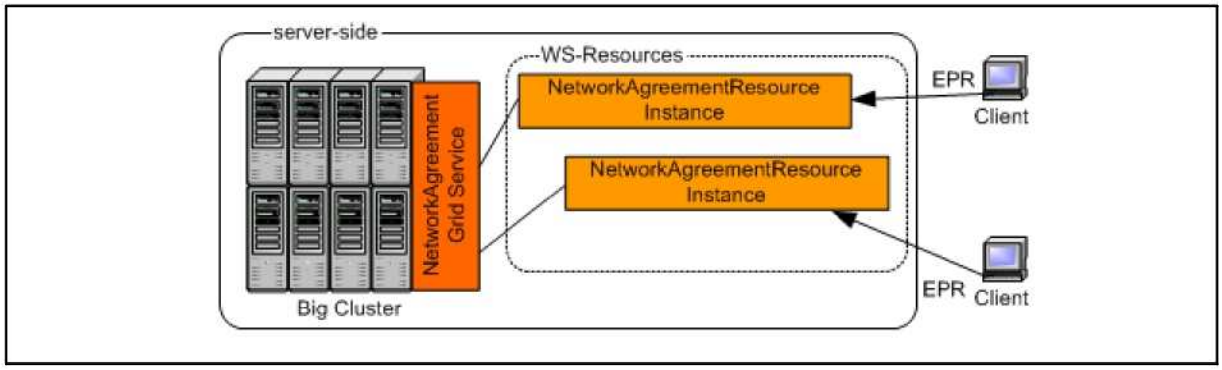

Figure 6.2: Basic concept of the `NetworkAgreement` Grid Service.

The need for a factory

In WSRF, a factory is used to create stateful WS-Resources and to create an EPR that can be used to address a WS-Resource on the server side through the service. A client can use the EPR to interact with the WS-Resource, in this case the `NetworkAgreementResource` instance. As a consequence, a factory must be established to create `NetworkAgreementResource` instances that are accessible through the `NetworkAgreement` service. Note when the service is invoked without using any kind of a factory, the `NetworkAgreement` service can be seen as a normal Web Service. The factory can be seen as an intermediary between the client and the service in the creation process of WS-Resources. Figure 6.3 shows the relationship between a client, a factory and `NetworkAgreementResource` instances.

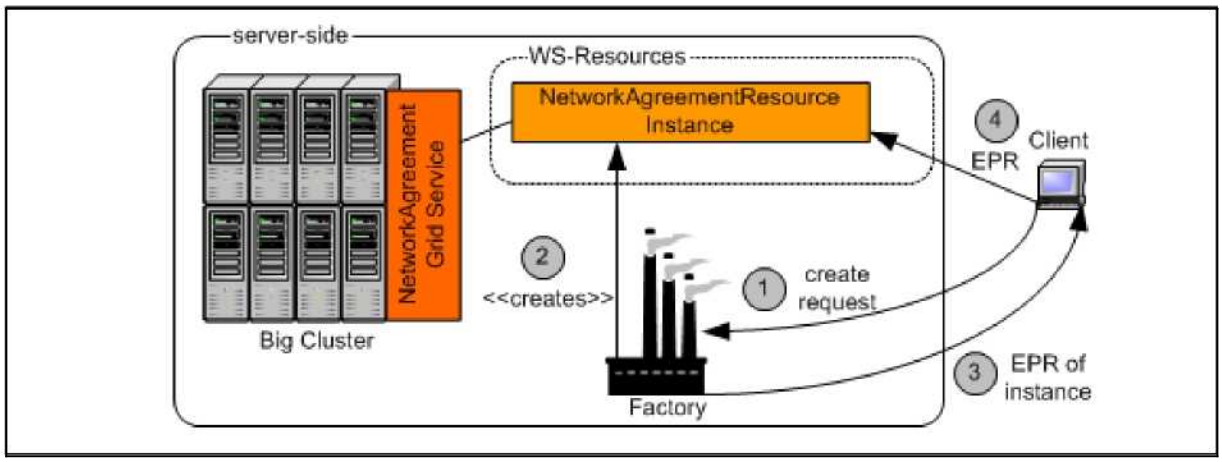

Figure 6.3: A factory that creates `NetworkAgreementResource` instances.

Network Reservation scenario as a case study

The network reservation example (see Chapter 3) relies on stateful WS-Resources that contain all the characteristics of an negotiated agreement inside their states. In particular, the negotiated characteristics of an agreement are well-defined by service description terms and guarantee terms ([**WS-Agreement**]) that represent the state of the WS-Resource. The stateful WS-Resources are realized by `NetworkAgreementResource` instances that are responsible to manage network reservation issues based on the negotiated characteristics. Network reservation issues can be the establishment, configuration, and destruction of a network tunnel inside a Grid. Therfore, the prototypes operate on a `Testbed` that simulates a Grid environment. Inside this `Testbed` is a network tunnel used for network reservation. To conclude, the behavior of one or more network tunnels inside the `Testbed` is controlled by a `NetworkAgreementResource` instance. Finally, Figure 6.4 shows the relationship between the `Testbed` and the stateful `NetworkAgreementResource` instances.

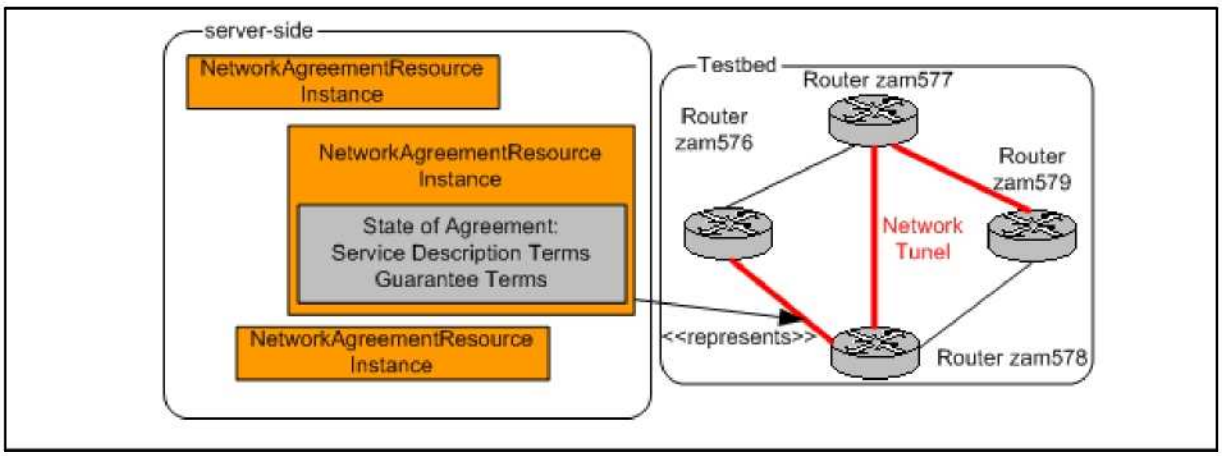

Figure 6.4: Network reservation inside the `Testbed`.

6.2.2 Description of the service's interface

The description of the `NetworkAgreement` service is splitted into three different WSDL files as shown in Figure 6.5. In addition, the WSDL files using several XML schema elements and types that are defined in the `agreement_types.xsd` XML schema file. More information about XML schema can be found in [**XML-Definitive**] or [**XML-Schema**]. Note that the XML schema and WSDL files are all located in the `wsag` namespace.

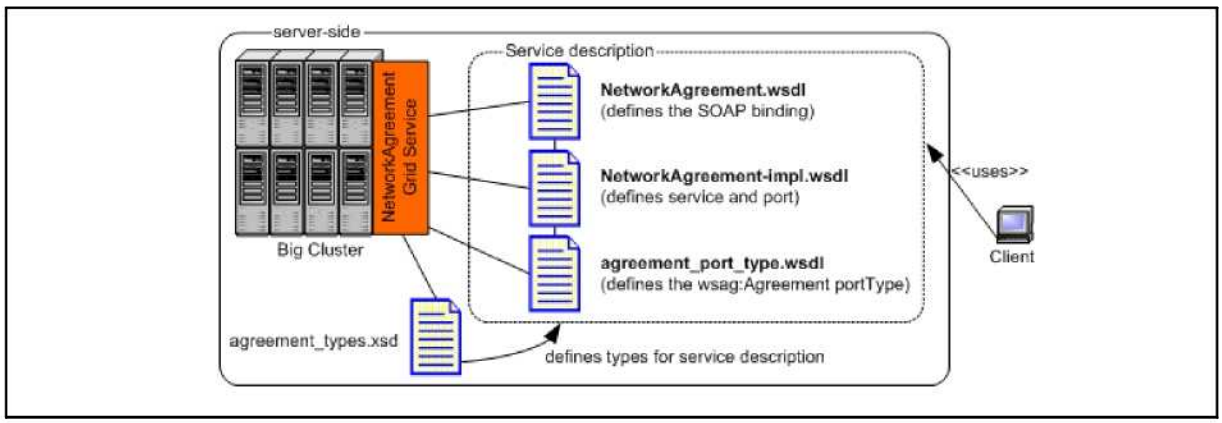

Figure 6.5: `NetworkAgreement` service description files.

agreement_port_type.wsdl

The **[WS-Agreement]** specification defines an `Agreement` portType that can be found inside the `agreement_port_type.wsdl` file. The portType is defined in the `wsag` namespace and includes the following two operations:

- `GetResourceProperty`
  This operation enables the portType to access resource properties of an WS-Resource as defined in **[WS-ResourceProperties]**. Hence, this operation provides access to agreement characteristics that are stored within the state of a `NetworkAgreementResource` instance.

- `Terminate`
  This operation can be used to force a termination of an agreement by the service requestor. As a side-remark, the `Terminate` operation is not related to service lifetime operations as defined in **[WS-ResourceLifetime]**. As a consequence, after the `Terminate` operation, the agreement still exists. On the other hand, after the `Destroy` operation as defined in **[WS-ResourceLifetime]**, the agreement exists not anymore.

Furthermore, the `Agreement` portType refers to exposed resource properties that are defined in the same WSDL file. The resource properties are named as `agreementProperties` and are also defined in the `wsag` namespace. The specification defines three different resource properties that are corresponding to the well-defined structure of an agreement inside the **[WS-Agreement]** specification, namely:

- `wsag:Name`
- `wsag:Context`
- `wsag:Terms`

The resource properties can be accessed through the `GetResourceProperty` operation of the `Agreement` portType. Note that these three exposed resource properties are the only allowed parameters for resource property requests. See **[WS-ResourceProperties]** for more information related to restrictions about the usage of resource properties. Finally, Figure 6.6 shows the `Agreement` portType and the related `agreementProperties`.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsag="http://www.ggf.org/namespaces/ws-agreement"
  xmlns:wsrp="http://www.ibm.com/xmlns/stdwip/Web-services/
              WS-ResourceProperties"
  targetNamespace="http://www.ggf.org/namespaces/ws-agreement" ...>
  ...
  <xs:element name="agreementProperties"
              type="wsag:AgreementPropertiesType"/>
  <xs:complexType name="AgreementPropertiesType">
    <xs:sequence>
      <xs:element ref="wsag:Name" minOccurs="0"/>
      <xs:element ref="wsag:Context"/>
      <xs:element ref="wsag:Terms"/>
    </xs:sequence>
  </xs:complexType>
  ...
  <wsdl:portType name="Agreement"
      wsrp:ResourceProperties="wsag:agreementProperties">
    <wsdl:operation name="GetResourceProperty">
       ...
    </wsdl:operation>
    <wsdl:operation name="Terminate">
       ...
    </wsdl:operation>
    <!-- domain specific extensions -->
    ...
  </wsdl:portType>
</wsdl:definitions>
```

Figure 6.6: `Agreement` portType and exposed resource properties ([**WS-Agreement**]).

The official specified `Agreement` portType includes only two methods, but there is the possibility to add other domain-specific relevant operations to the portType. Here, a well-known anti-pattern of Web Services motivates the addition of some more operations related the resource properties. Web Services anti-patterns are typical mistakes being made when doing Web Services and distributed systems in general ([**WS-SpeedStart**]). One of these anti-patterns is the fact that Web Services are often created on basics related to object-oriented design principles. Hence, every property of a WS-Resource on the server has its own *'getter'* or *'setter'* operation. In WSRF, the `GetResourceProperty` operation is used to get one property value from a specific property defined in the request of the operation. To give an example for this anti-pattern, all information about the agreement is needed at the client side. Therefore, the client invokes three times the `GetResourceProperty` operation in a row to get all the three different resource properties. Every invocation includes the usual transport overhead, for instance protocol-specific information. To avoid this anti-pattern, there should be an operation that provides the functionality to get all information in one single transfer. To conclude, avoiding multiple calls across the network end up in better performance.

To avoid the described anti-pattern and to give an example of domain-specific extensions two operations are added to the `Agreement` portType. According to [**WSRF**] the *'WS-Resource*

Property Composition' approach can be used for this. The idea of this approach is that the various specifications related to WS-Resources have been designed to be composable. As a consequence, the design of a Web Service interface can be composed by copy-and-paste of the operations defined in the constituent portTypes used in the composition ([**WSRF**]). In this case, the operations defined in the `Agreement` portType can be combined with some operations defined in the [**WS-ResourceProperties**] specification, namely:

- `GetMultipleResourceProperties`
  This operation allows a requestor to retrieve the values of multiple resource properties of a WS-Resource. Hence, all three exposed resource properties of an `NetworkAgrementResource` instance can be sent by one transfer to the requestor.

- `QueryResourceProperties`
  This operations allows a requestor to query the resource properties document of a WS-Resource using a query expression such as [**XPath**]. As a consequence, there can be also more than one properties sent back as a result of this operation.

As a side-remark, this approach provides the same functionality as the old OGSI approach of inheriting portTypes and operations. However, the demand for other [**WS-ResourceProperties**] operations is only one example of domain-specific extensions. There can be also other demands, for instance operations related to [**WS-ResourceLifetime**] or user-defined operations to support domain-specific functionality. Finally, Figure 6.7 shows the `Agreement` portType with domain specific extensions.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsag="http://www.ggf.org/namespaces/ws-agreement"
  xmlns:wsrp="http://www.ibm.com/xmlns/stdwip/Web-services/
              WS-ResourceProperties"
  targetNamespace="http://www.ggf.org/namespaces/ws-agreement" ...>
  ...
  <wsdl:portType name="Agreement"
                 wsrp:ResourceProperties="wsag:agreementProperties">
    <wsdl:operation name="GetResourceProperty">
      ...
    </wsdl:operation>
    <wsdl:operation name="Terminate">
      ...
    </wsdl:operation>
    <!-- domain specific extensions -->
    <wsdl:operation name="GetMultipleResourceProperties">
      ...
    </wsdl:operation>
    <wsdl:operation name="QueryResourceProperties">
      ...
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

Figure 6.7: `Agreement` portType with domain-specific extensions.

NetworkAgreement.wsdl

The description of operations inside the `agreement_port_type.wsdl` is not related to a specific protocol. Therefore, the `NetworkAgreement.wsdl` file contains a binding for the SOAP protocol, named `NetworkAgreementBinding`. The `NetworkAgreementBinding` defines message format and protocol details for operations and messages defined by the `Agreement` portType ([**WSDL 1.1**]). Hence, the definition of the `Agreement` portType is imported into the `NetworkAgreement.wsdl` file as shown in Figure 6.8. Furthermore, the binding references the portType that it binds using the `type` attribute. Note that there can be any number of bindings for a given portType. Important is that the name, `NetworkAgrementBinding`, provides a unique name among all defined bindings inside the service description. When using SOAP binding it is necessary to include an `<soap:binding>` element. The purpose of the `<soap:binding>` element is to signify that the binding is bound to the SOAP protocol format ([**WSDL 1.1**]). The `transport` attribute indicates which transport of SOAP this binding corresponds to. In this case `http://schemas.xmlsoap.org/soap/http` corresponds to the HTTP binding in the [**SOAP**] specification. Note that there can be also other transports used, for instance Simple Mail Transfer Protocol (SMTP) or FTP. The style of the SOAP binding of the `NetworkAgreement` service is document-oriented which means that the messages use documents as parameters and result values. There is also the possibility to change to the RPC-oriented style that uses concrete types for parameters and return values. Because every specification related to WS-Agreement and WSRF use XML, the document-oriented style is a sophisticated choice for the binding of the `NetworkAgreement` service. More detailed information about the elements of a WSDL file can be found in [**WSDL 1.1**]. Furthermore, the complete WSDL file can be found in Appendix B. Finally, Figure 6.8 shows the important parts of the SOAP binding.

Figure 6.8 shows the `GetResourceProperty` operation in more detail. Note that all other operations are defined in the same style as the `GetResourceProperty` operation. However, the `soapAction` attribute of the `<soap:operation>` element specifies the value of the SOAPAction header for the correspondent operation. The SOAPAction header is a single additional HTTP header containing an URI. The client that uses the service description of an operation, should send the value of the SOAPAction header unchanged back to the service provider. The value of the additional HTTP header indicates the purpose of the corresponding SOAP message. In this case, the URL `http://www.ibm.com/xmlns/stdwip/Web-services/WS-ResourceProperties` is used to indicate that a `GetResourceProperty` request can be found inside the SOAP message. This additional information can be interesting for the configuration of firewalls that can react and perform any kind of conditional processing based on the presence of the SOAPAction header ([**XML-Prof**]). Finally, the `literal` attribute of the `<soap:body>` element indicates that each part of the input or output references a concrete schema definition. In this case the concrete schema definitions of the operations can be found inside the `agreement_port_type.wsdl` file.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsag="http://www.ggf.org/namespaces/ws-agreement"
  targetNamespace="http://www.ggf.org/namespaces/ws-agreement" ...>
  ...
  <import location="http://localhost:4400/wstk/
                    wsag/agreement_port_type.wsdl"
          namespace="http://www.ggf.org/namespaces/ws-agreement"/>

  <binding name="NetworkAgreementBinding" type="wsag:Agreement">

    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http"/>

    <wsdl:operation name="GetResourceProperty">
      <soap:operation
          soapAction="http://www.ibm.com/xmlns/stdwip/Web-services/
                      WS-ResourceProperties"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    ...
  </binding>
</definitions>
```

Figure 6.8: SOAP binding of the `Agreement` portType.

NetworkAgreeement-impl.wsdl

The unique name of the service is defined inside the `<service>` element. The `<service>` element can be found in the `NetworkAgreement-impl.wsdl` file. Furthermore, the `service` element consists of a collection of related ports. The `<port>` element references the unique `<binding>` element, in this case `AgreementBinding`. Therefore, the `NetworkAgreement.wsdl` is imported for the access of the `AgreementBinding` definition. However, the `<port>` element defines an service endpoint. Such a service endpoint is related to the SOAP address binding and provides one address for the correspondent `port`. In this case, the service can be found at the URL `http://localhost:4400/wstk/services/NetworkAgreement`. Finally, Figure 6.9 shows the `<service>` and `<port>` element. Note that the complete listing of `NetworkAgreement-Impl.wsdl` can be found in Appendix B.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions...>
  <import location="http://localhost:4400/wstk/wsag/NetworkAgreement.wsdl"
    namespace="http://www.fz-juelich.de/namespaces/
               wsag/NetworkAgreement"/>

  <service name="NetworkAgreement">
    <documentation>Simple Agreement Service to illustrate
    GGF GRAAP agreement negotiations.</documentation>
    <port binding="interface:NetworkAgreementBinding"
          name="AgreementPort">
       <soap:address location=
         "http://localhost:4400/wstk/services/NetworkAgreement"/>
    </port>
  </service>
</definitions>
```

Figure 6.9: Naming the service as `NetworkAgreement` service.

6.2.3 Specification of the service's implementation

The specification of the service's implementation is very close related to the description of the service interface. Remember, the service description defines [**WS-Agreement**] compliant functionality, but also operations to address domain-specific requirements. Therefore, the specification use the following two packages to organize all functionality in relation to their intention:

- `ggf.wsag` package
  Provides [**WS-Agreement**] compliant interfaces.

- `fzjuelich.wsag` package
  Provides domain-specific functionality and interfaces.

Figure 6.10 provides an overview of both packages.

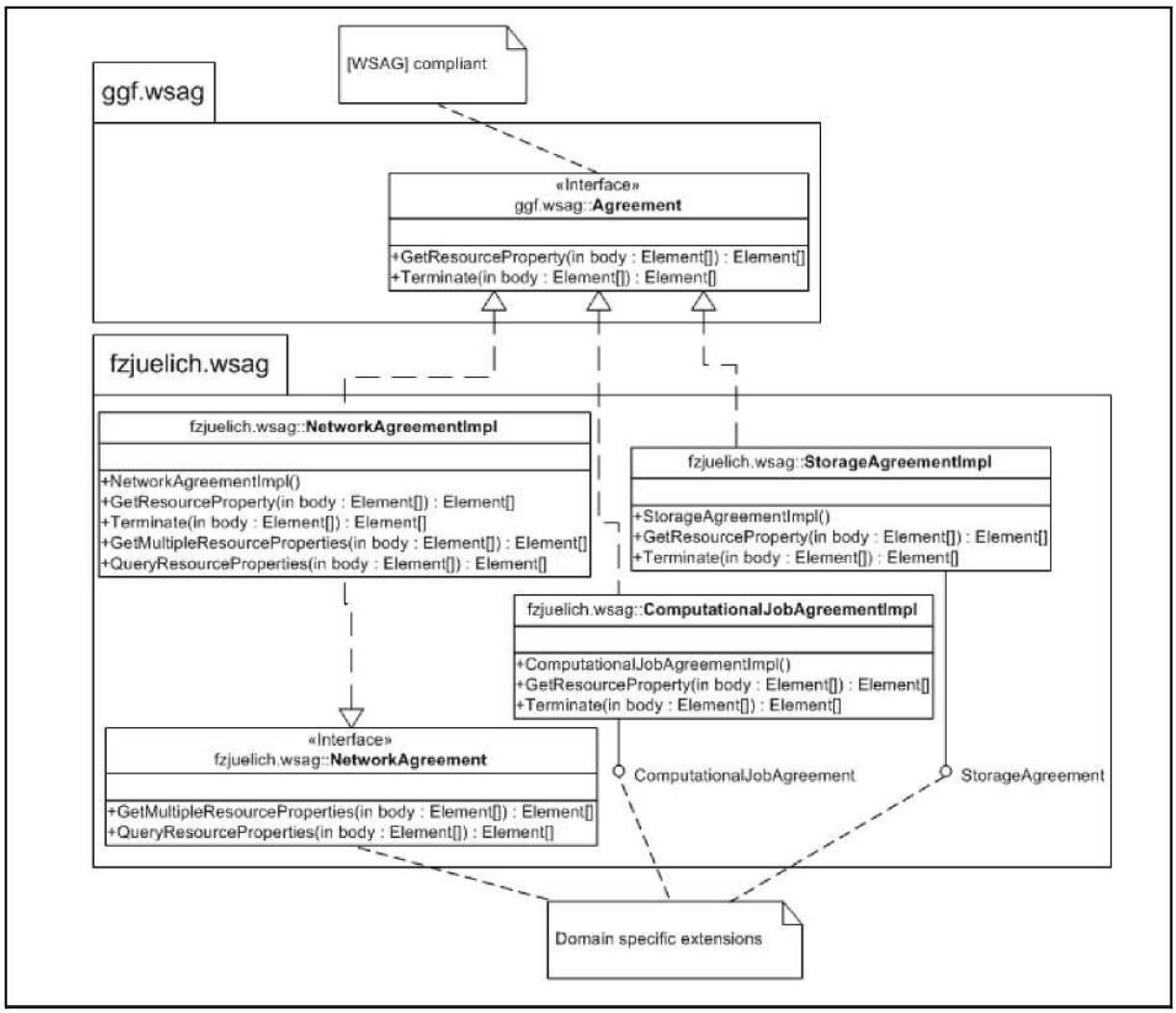

Figure 6.10: Specification of the `NetworkAgreement` service implementation.

Figure 6.10 shows an interface `Agreement` that contains the two methods that are defined for an **[WS-Agreement]** compliant agreement, namely:

- `GetResourceProperty`
- `Terminate`

Hence, this interface is part of the **[WS-Agreement]** compliant `ggf.wsag` package. Furthermore, the interface `NetworkAgreement` is specified that should contain any kind of domain-specific methods related to the implementation of the `NetworkAgreement` service. Therefore, the interface includes both methods that were introduced in the WSDL service description:

- `GetMultipleResourceProperties`
- `QueryResourceProperties`

Note that this interface can also include local domain-specific methods that are not exposed as Web Service operations in the WSDL file. For instance, some methods related to the calculation of best paths through the network that can be used by other implementation parts of the `NetworkAgreement` service, but not from a service requestor. To conclude, the design approach also allows the definition of local operations that cannot be invoked remotely.

The benefit of the differentiated interfaces approach is that implementations have both domain-specific and **[WS-Agreement]** compliant identity. Every class that implements the `Agreement` interface provides **[WS-Agreement]** compliant functionality. To provide well-defined domain-specific functionality related to network reservation, the same class can also implement a domain-specific interface, for instance the `NetworkAgreement` interface.
The `NetworkAgreementImpl` class represents the implementation of the `NetworkAgreement` service and implements both the `Agreement` and `NetworkAgreement` interfaces. As a consequence, every operation invocation of the `NetworkAgreement` service invokes one method of the `NetworkAgreementImpl` class.

In addition, Figure 6.10 shows alternative implementations of **[WS-Agreement]** compliant agreement services that are also included in the domain specific `fzjuelich.wsag` package, namely:

- `ComputationalJobAgreementImpl`
- `StorageAgreementImpl`

All these implementations of agreement services implement the `Agreement` interface of the `ggf.wsag` package. Furthermore, the class `ComputationalJobAgreementImpl` implement an interface called `ComputationalJobAgreement` that included domain specific extensions of this agreement type. The interface `ComputationalJobAgreement` can be used to define methods that are related to computational jobs. The class `StorageAgreementImpl` implements the interface `StorageAgreement` that contains domain specific extensions of the correspondent agreement type. The interface `StorageAgreement` can be used to define methods that are related to storage aspects.

6.2.4 Specification of the resource's implementation

The design of the resource's implementation is influenced by the existing infrastructure that is provided by the ETTK. For instance, already implemented interfaces or functionality related to automated requests as defined in **[WS-ResourceProperties]**. The implementation of the WS-Resource for an network agreement is partitioned into three parts as shown in Figure 6.11. Instead of putting all WS-Resource functionality together in one class, the three classes are partitioned for a better understanding of their intentions and for a better composable design. Furthermore, the delegation approach for resource properties functionality allows extensions such as more than one XML resource property tree. Hence, the `NetworkAgreementResource` needs maybe also some private information about the agreement that are related to network issues such as best paths or router configuration details. The delegation model allows the composition of another WS-Resource that provides private resource properties that are not accessible through the Web Service.

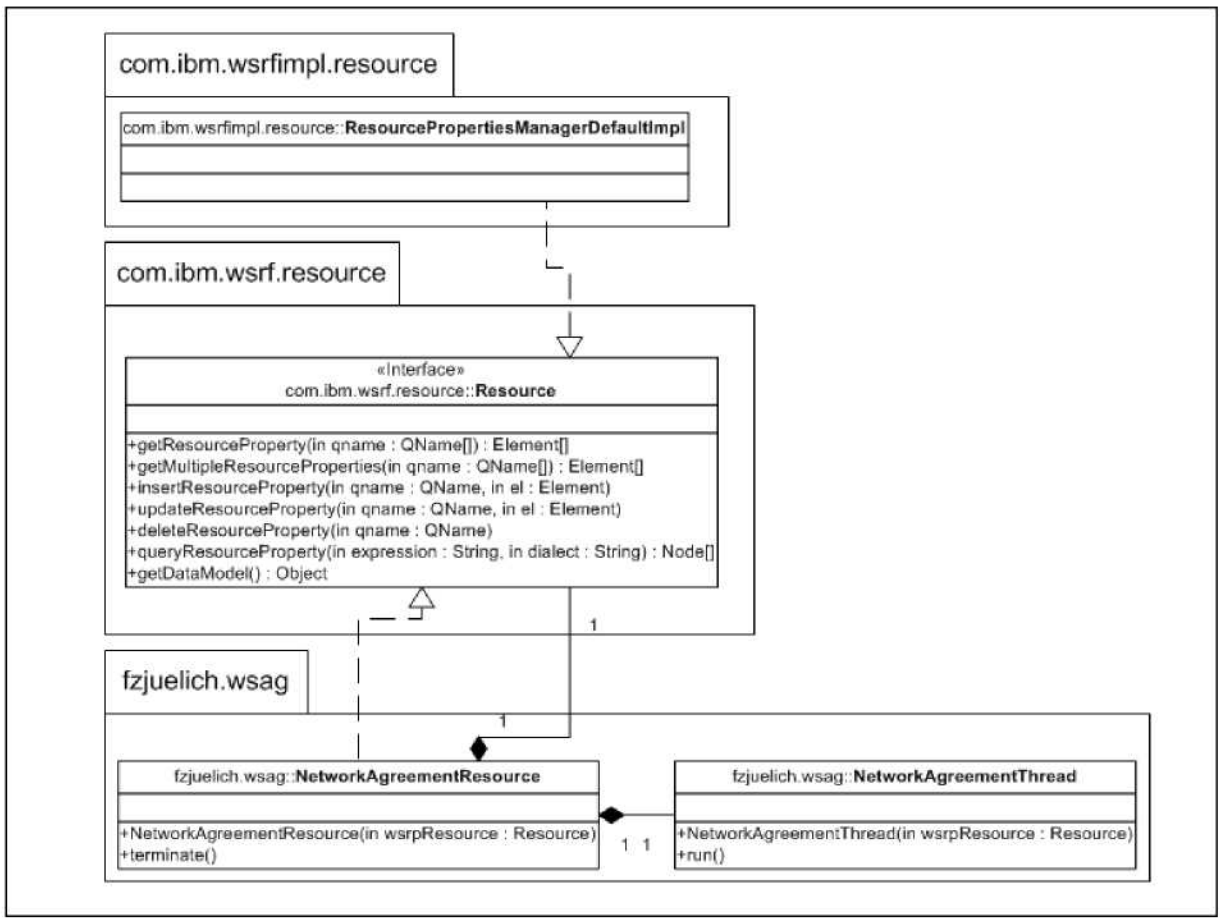

Figure 6.11: Specification of the partitioned `NetworkAgreementResource`.

The `NetworkAgreementResource` is the central part of the whole WS-Resource. As a consequence, an instance of this class is addressed by the EPR of WS-Addressing. The constructor of the `NetworkAgreementResource` takes an implementation of the `Resource` interface. This implementation contains the agreement state as resource properties. Hence, the factory should provide the `NetworkAgreementResource` with such an implementation, because the factory takes over the negotiation process and finally accepts an agreement offer that is converted into the resource properties of the implementation. Furthermore, this implementation is used in every request related to **[WS-ResourceProperties]**. This is accomplished by simple delegation to this composed implementation of the `Resource` interface. An default implementation of the `Resource` interface is also provided by the ETTK, namely: `ResourcePropertiesManagerDefaultImpl`. The default implementation provides default functionality to handle **[WS-ResourceProperties]** requests. Figure 6.12 shows the sequence of an incoming `getResourceProperty` method call and its delegation to an instance of the `ResourcePropertiesManagerDefaultImpl`.

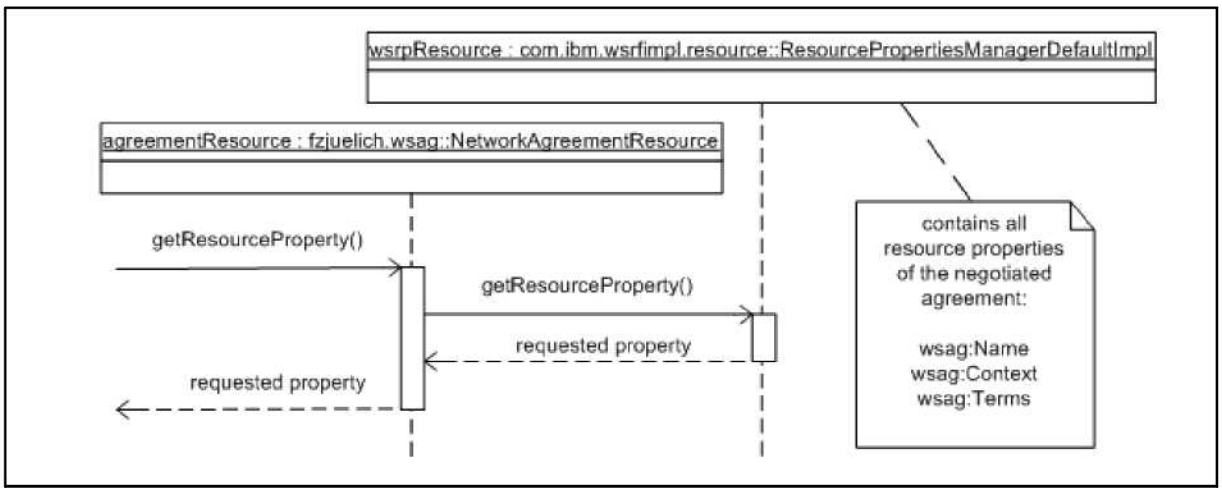

Figure 6.12: `getResourceProperty` call with delegation.

Note that four methods of the `NetworkAgreementResource` are not exposed as Web Service operations. Hence, the following methods cannot be invoked remotely:

- `insertResourceProperty`
- `updateResourceProperty`
- `deleteResourceProperty`
- `getDataModel`

These methods are defined to support the `SetResourceProperty` operation as defined in **[WS-ResourceProperties]**. Because the agreement is designed to be read-only after the negotiation process the `SetResourceProperty` operation is not exposed as an Web Service operation. Therefore, the listed methods will never be used, but note that the support of the `SetResourceProperty` operation can be easily integrated later. Instead of these methods, the defined `terminate` method of the `NetworkAgreementResource` is exposed as a Web Service operation.

The realization of prototypes are related to the network reservation example as introduced in Chapter 3. Therefore, the `NetworkAgreementResource` must be able to create or remove network tunnels in the `Testbed`. An implementation of the `NetworkAgreementResource` must be able to check agreement terms during the service lifetime to invoke specific operations that are necessary to meet the terms. For instance, one of the service description terms defines a start-time for the service inside the agreement. Hence, the `NetworkAgreementResource` has to check the terms in relation to the time. When the start-time of the service is reached the `NetworkAgreementResource` must establish a network tunnel until the possible end-time of the service is reached.

To address the demand of dynamic control of network tunnels inside the `Testbed` a thread oriented architecture is used. In particular, the `NetworkAgreementResource` composes a `NetworkAgreementThread` that takes over this dynamic control. As a consequence the `NetworkAgreementThread` is the only class that interacts directly with the `Testbed`. Therefore, the constructor of this class takes the implementation of the `Resource` interface to access

the resource properties including the terms. As a side-remark, this allows also the dynamical change of network tunnel agreement terms in relation to the real established tunnel in the `Testbed`. Finally, Figure 6.13 shows the creation and remove of a network tunnel inside the `Testbed`. Note that the instance `testbed` represents the access to the `Testbed`.

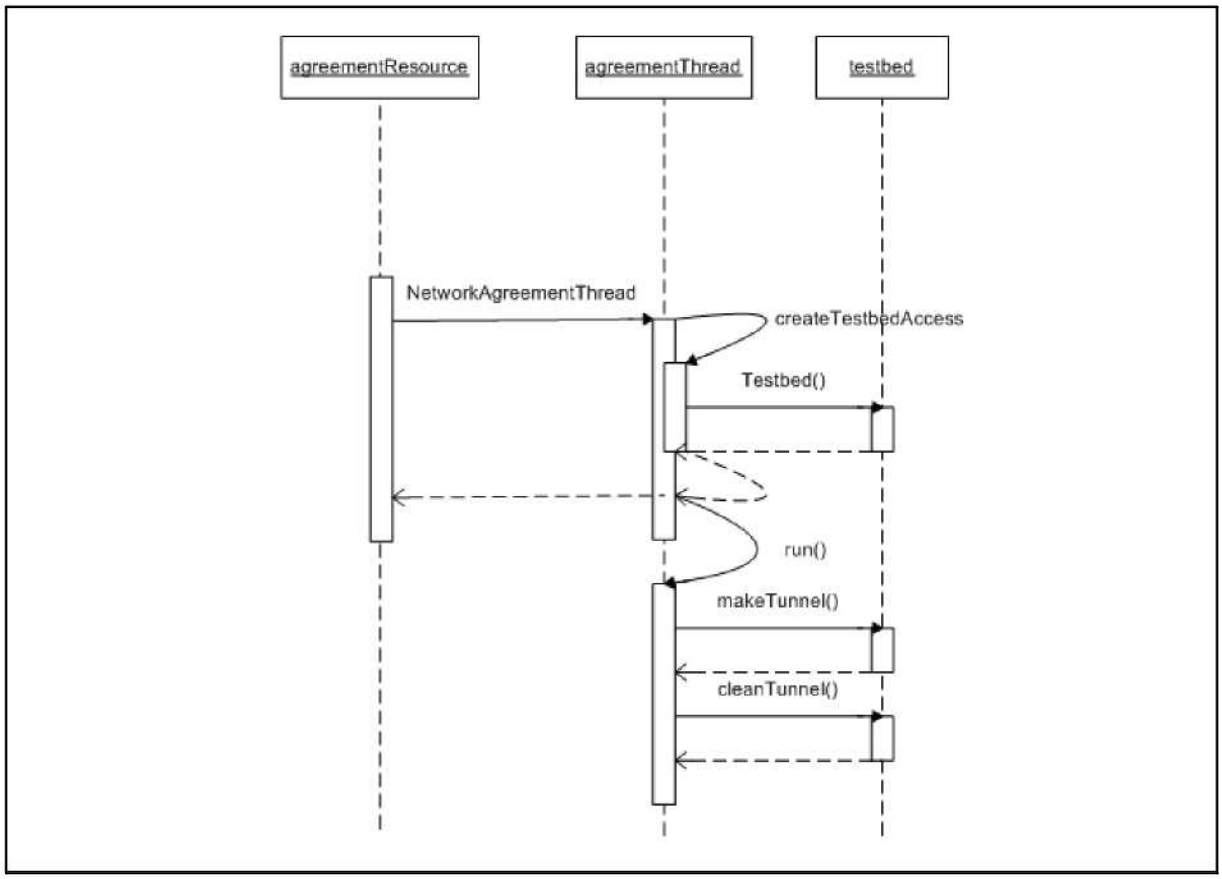

Figure 6.13: The thread architecture allows tunnel-control at service lifetime.

6.2.5 Implementation of the service

Because this thesis focuses on the specification, the implementation details of all classes are out of scope of this thesis. Therfore, only the important interfaces and important source elements are mentioned. Any agreement class that implement the `ggf.wsag.Agreement` interface is an **[WS-Agreement]** compliant agreement. This interface is shown in Figure 6.14.

```
package ggf.wsag;

import org.w3c.dom.Element;

public interface Agreement
{
  public Element[] GetResourceProperty(Element[] body)
  throws Exception;

  public Element[] Terminate(Element[] body)
  throws Exception;
}
```

Figure 6.14: Java interface `ggf.wsag.Agreement`.

To support domain specific extensions another interface is used, called `NetworkAgreement`. Note that this interface is not a part of the official `ggf.wsag` package, because this interface can include methods for network agreements that are domain-specific. Therefore, the domain-specific package `fzjuelich.wsag` contains this interface. The interface is shown in Figure 6.15.

```
package fzjuelich.wsag;

import org.w3c.dom.Element;

public interface NetworkAgreement
{
  public Element[] GetMultipleResourceProperties(Element[] body)
  throws Exception;

  public Element[] QueryResourceProperties(Element[] body)
  throws Exception;
}
```

Figure 6.15: Java interface `fzjuelich.wsag.NetworkAgreement`.

The implementation of the service is called the `NetworkAgreementImpl` class. This class implements the **[WS-Agreement]** compliant interface `Agreement` and also the domain-specific `NetworkAgreement` interface. Figure 6.16 shows the implemented methods of both interfaces.

```
package fzjuelich.wsag;

public class NetworkAgreementImpl
implements Agreement, NetworkAgreement
{
  ...
  public Element[] GetResourceProperty(Element[] body)
  throws Exception {...}

  public Element[] Terminate(Element[] body)
  throws Exception {...}

  public Element[] GetMultipleResourceProperties(Element[] body)
  throws Exception {...}

  public Element[] QueryResourceProperties(Element[] body)
  throws Exception {...}

  ...
}
```

Figure 6.16: `NetworkAgreementImpl` class.

Already Implemented Functionality

The implementation of the `NetworkAgreementImpl` class uses everywhere it is possible already implemented functionality. For instance, the `request` object mechanism related to [**WS-ResourceProperties**] that is provided by the ETTK. The `request` object mechanism encapsulates incoming requests and handles conversion to the appropriate RPC style java method on a resource. This is accomplished by a `request` object that is created from a raw incoming request. The benefit of this object is that it can be interrogated to examine an incoming request without need to parse the message. Figure 6.17 shows the use of this mechanism for the `GetResourceProperty` method. Note that the implementation of the `GetMultipleResourceProperties` and the `QueryResourceProperties` methods uses also nearly the same approach as in the `GetResourceProperty` method.

The ETTK provides an internal `WSRFFactory` that is used to create a `GetResourcePropertyRequest` instance. Note that the parameter `body` of the method is forwarded to the `GetResourcePropertyResource` create call. If the request should be fulfilled, the `process` method may be called on the `GetResourcePropertyRequest` instance. The `process` method obtains the appropriate resource from the `LifetimeManager` and invokes the method appropriate for this message request. This mechanism is one of the benefits of the ETTK.

```
public class NetworkAgreementImpl
implements Agreement, NetworkAgreement
{
  ...
  public Element[] GetResourceProperty(Element[] body)
  throws Exception {

    GetResourcePropertyRequest request;
    request = getFactory().createGetResourcePropertyRequest(body);

    Response rsp = null;

    rsp = request.process();

    return rsp.toMessage();
  }
  ...
}
```

Figure 6.17: Implementation parts of the `GetResourceProperty` method.

Terminate

Another method of the `NetworkAgreeementImpl` class is the `Terminate` method as defined in the **[WS-Agreement]** specification. The `LifeTimeManager` is used to get a reference to the resource that is addressed in the message. The `getFactory` method provides a reference for an internal factory that is responsible for global references, for instance the `LifeTimeManager` reference. `Terminate` uses not the predefined functionality of `request` object mechanism, because the method is not a part of the **[WS-ResourceProperties]** specification. The reference to the resource is casted into an instance of the `NetworkAgreementResource` class. Finally, the method `terminate` is called using this instance. To conclude, the real functionality of the `terminate` method is in the `NetworkAgreementResource` class. Figure 6.18 shows the implementation of the `terminate` method.

```
public class NetworkAgreementImpl
implements Agreement, NetworkAgreement
{
  ...
  public Element[] Terminate(Element[] body)
  throws Exception {

  Object o =
    getFactory().getWSResourceLifetimeManager().getResource();

  NetworkAgreementResource agreementResource =
    (NetworkAgreementResource)o;

  agreementResource.terminate();

  ...}
}
```

Figure 6.18: Implementation parts of the `Terminate` method.

6.2.6 Implementation of the Resources

The service implementation class `NetworkAgreementImpl` uses three different classes that are used to represent one WS-Resource, namely:

- NetworkAgreementResource
- NetworkAgreementThread
- any class that implements the `Resource` interface

Implementation of the NetworkAgreementResource

The `NetworkAgreementResource` class can be seen as the central manager for the complete agreement resource. It manages **[WS-ResourceProperties]** compliant requests that are forwarded to a composed instance that implements the `Resource` interface. Furthermore, the `NetworkAgreementResource` class composes a `NetworkAgreementThread`. The thread is responsible for the creating and the remove of network tunnels.

```
public class NetworkAgreementResource
implements Resource
{
  private NetworkAgreementThread agreementThread;
  private Resource wsrpResource;

  public NetworkAgreementResource(Resource  wsrpResource) {
    this.wsrpResource = wsrpResource;
    createAgreementThread();
  }

  private void createAgreementThread()
  {
    agreementThread = new NetworkAgreementThread(this);
    agreementThread.start();
  }
  ...
  public Element[] getResourceProperty(QName qname)
  throws Exception {
    return wsrpResource.getResourceProperty(qname);
  }
}
```

Figure 6.19: Source elements of the `NetworkAgreementResource`.

The constructor of the `NetworkAgreementResource` class takes the resource properties for the agreement resource. Furthermore, the thread is started which is responsible to check the agreement terms that are stored inside the resource properties. Note that the `Resource` interface is implemented by the `NetworkAgreementResource` class. Therefore, all methods as defined in **[WS-ResourceProperties]** are implemented in the class. Figure 6.19 shows, as an example, the `getResourceProperty` method that is used to forward an property request to a composed instance that implements the `Resource` interface using simple delegation.

Implementation uses a thread oriented design

The whole agreement resource is the representation of an agreement tunnel in the `Testbed`. A thread oriented design is chosen to control the tunnel dynamically.

```
public class NetworkAgreementThread
extends Thread {

  public NetworkAgreementThread(Resource wsrpResource)
  {
    this.wsrpResource = wsrpResource;
    createTestbedAccess();
  }

  private void createTestbedAccess()
  {
   // creates the access to the testbed using a special lib
  }
  ...
}
```

Figure 6.20: Source elements of the `NetworkAgreementThread`.

Figure 6.20 shows the constructor that also takes an instance that implements the `Resource` interface. This instance can be used to check agreement terms while the thread is running. Therefore, the `run` method calls periodically the `checkTerms` method that is responsible for checking all agreement terms and taking necessary actions such as the creation or remove of a network tunnel. Figure 6.21 shows the `run` method that uses a while loop to check all agreement terms with the use of the `checkTerms` method.

```
public class NetworkAgreementThread
extends Thread {
  ...
  public void run()
  {
    ...
    while (true) {
      if ( isInterrupted() ) {
        break;
      }
      ...
      checkTerms();
      ...
    }
    ...
  }
}
```

Figure 6.21: Source elements of the `NetworkAgreementThread`.

6.2.7 Deployment

Usually the deployment of a Web Service is done with a few mouse clicks within a development environment such as WebSphere Application Developer or the Eclipse Platform. For a better understanding of the whole deployment process, all steps are shown without the help of a development environment. That gives the opportunity to understand better how all the components are related to each other. To work properly with the embedded hosting environment of the ETTK it is necessary to define an environment variable, namely:

- `$ETTK_HOME`
  used to refer to the home of the ETTK installation

Note that the embedded application server is located in `$ETTK_HOME/appserver/`. It is essential to know that the root directory for deploying services is located in `$ETTK_HOME/appserver/installedApps/DefaultNode/wstkdemos.ear/wstk.war`. This directory is refered as `$DEPLOY_HOME`. The directory name `wstk.war` goes back to the old times where a special configuration tool of the old WSTK installed services as part of a `wstk.war` file created for the hosting environment. This file is a Web Application Archive (WAR) as defined in [**SunWAR**] and has a specific directory structure.

Deployment of the service

The hosting environment of the ETTK uses the AXIS SOAP engine. Therefore, the deployment of a service within the embedded hosting environment is done by using the XML-based Web Service Deployment Descriptor (WSDD) language that is defined by AXIS ([**AXIS-Ref**]). In particular, all services that are deployed within the embedded application server are described within the `$DEPLOY_HOME/Web-INF/server-config.wsdd` file. Figure 6.22 shows the deployment description of the `NetworkAgreement` service.

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment
  xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  ...
  <service name="NetworkAgreement" provider="java:MSG">
    <parameter
        name="className"
        value="fzjuelich.wsag.NetworkAgreementImpl"/>
    <parameter name="allowedMethods" value="*"/>
  </service>
  ...
</deployment>
```

Figure 6.22: Deployment of the `NetworkAgreement` service.

The `<deployment>` element is the root element of the deployment document that contains namespace definitions such as `xmlns:java`. The embodied `<service>` element is used to deploy an AXIS service. The `name` attribute represents the name of the service, in this case `NetworkAgreement`. In addition, the `provider="java:MSG"` attribute tells AXIS which dispatcher is used ([**WS-DEVETTK2**]). Furthermore, it tells AXIS that this service is a message service and no RPC service that could be established with the attribute value `provider="java:RPC"`. Hence, the `NetworkAgreement` service is a message service. In particular, all

operations of the service take in an array of DOM elements ([**DOM**]) that corresponds to the SOAP body element. The result value of each operation is also an array of DOM elements. Unlike the RPC case where AXIS will convert the XML to and from Java programming language elements automatically, in messages AXIS leaves the XML untouched ([**WS-DEVETTK2**]).

As shown in Figure 6.22, the first parameter defines the class that implements the Web Service interface. The `fzjuelich.wsag.NetworkAgreementImpl.class.` is used to implement the Web Service interface. The second parameter can be used to specify methods which are allowed as Web Service methods. It is also possible to specify for this parameter the value `*` which means that all methods are Web Service methods. As a side-remark, that allow methods functionality similar to public and private methods in Java. Note that all elements referred in this section are in the WSDD namespace, namely `http://xml.apache.org/axis/wsdd/`. More information about the WSDD language can be found in [**AXIS-Ref**].

The configuration of the WS-Addressing handler is also included in the `$DEPLOY_HOME/Web-INF/server-config.wsdd` file. As shown in Figure 6.23 the `<handler>` element can be used to configure a handler inside the AXIS handler chain ([**AXIS-Arch**]). As a consequence, the WS-Addressing handler is invoked during the request or response to implement transport-specific functionality as defined in [**WS-Adressing**]. The configuration inside the `globalConfiguration` means that every Web Service of this embedded application service uses the WS-Addressing handler. More details about the configuration and behavior of handler chains can be found in [**AXIS-Ref**].

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment
  xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  ...
  <globalConfiguration>
    ...
    <requestFlow>
      <handler type="WSAddr"/>
      ...
    </requestFlow>
    <responseFlow>
      <handler type="WSAddr"/>
      ...
    </responseFlow>
  </globalConfiguration>
  ...
</deployment>
```

Figure 6.23: Configuration of the WS-Addressing handler.

Beside the deployment description with the use of the WSDD language, there must be also a deployment location for the description of the service that is written in WSDL. The `$DEPLOY_HOME/wsag` location is used to deploy all WSDL or related XML schema files. Hence, this directory contains all the files that are related to the description of the `NetworkAgreement` service. This files are shown in the following enumeration:

1. NetworkAgreement.wsdl
2. NetworkAgreement-impl.wsdl
3. agreement_port_type.wsdl
4. agreement_types.xsd

Deployment of the resources

The specific directory structure of a WAR defines a `$DEPLOY_HOME/Web-INF` directory directly beneath the WAR directory level. Inside this very important subdirectory is the Web application deployment descriptor `Web.xml` file located and also some further directories. The following implemented classes are deployed into the `$DEPLOY_HOME/Web-INF/classes` subdirectory as follows:

- `/ggf/wsag/Agreement.class`
- `/fzjuelich/wsag/NetworkAgreement.class`
- `/fzjuelich/wsag/NetworkAgreementImpl.class`
- `/fzjuelich/wsag/NetworkAgreementResource.class`
- `/fzjuelich/wsag/NetworkAgreementThread.class`

Remember that packages in Java are corresponding to normal directories. For example, the interface `NetworkAgreement.java` belongs to the `fzjuelich.wsag` Java package, so inside the `classes` directory must be a directory structure `fzjuelich\wsag` present that includes all classes used server-sided by the `NetworkAgreement` service. As a side-remark, instead of using normal copy and paste functionality of the used operation system a good Ant script can be defined to simplify the described deployment procedure. Ant is a java build tool and very similar to the classic UNIX make command ([**Ant**]). The individual steps are described in one `build.xml` file, also called the modern 'makefile'. This file can be used to direct Ant what sources should be compiled and in which order and of course the copy command is one of the `CoreTasks` of Ant. All the described steps of deploying for better understanding which files belong in which directory can be reduces to one step, running an Ant script.

Deployment of used libraries

Inside the `Web-INF` directory is also the `lib` directory. That directory contains Java Archive (JAR) archives of utility libraries, which are used by server-side classes. The implementation of the service uses some libraries that must be deployed inside this directory:

- `wsrf.jar`
  API for the WSRF functionality
- `wsa.jar`
  API for the WS-Addressing functionality

Testing the deployment

Surely the test if all steps are correctly done is also a part of the deployment procedure. By using the shell script `$ETTK_HOME/bin/startserver.sh` the embedded application server is started. There are some outputs written and finally the server is started and its process ID is shown. The message `Server server1 is ready for e-business` tells the developer that the server offers now all the installed Grid or Web Services. Note that all logings are inside the directory `$ETTK_HOME/appserver/logs/server1` that include log files for `SystemOut` and `SystemErr` messages. Furthermore, this directory includes a `server1.pid` file that contains the actual Process ID (PID) of the server.

The URL `http://localhost:4400/wstk/services/NetworkAgreement` can be visited by a normal Webbrowser for testing the server and also the service. `AXIS error - No service is available at this URL` means that there is no service at this URL. This implies that configurations in the `server-config.wsdd` file are wrong or maybe some typo's are inside the WSDL files. On the other hand the message `Service NetworkAgreement is deployed.` tells the developer that his configurations and deployment procedure was correctly done. With the help of the server URL the description of the Grid Service can be found with using the URL `http://localhost:4400/wstk/wsag/NetworkAgreement.wsdl`. Of course working with a remote server demands the exchange of `localhost` with the obversely Server IP address. In fact in this scenario a remote server can be good tested by a good old command, named `telnet`. The command `telnet ipaddress 4400` can be used to easily check if the server is available from the client-side. Note that ipaddress is the IP of the machine, which runs the server. After some seconds the HTTP response `HTTP/1.1 408 Connection timedout while reading request`
`Server: WebSphere Application Server/5.1` is shown. Remember that this small command just tests the reachability of the server and not the service specific reachability, like in the first test, where the AXIS sevice deployed screen is shown. When changing the WSDL or class files the server does not need to be restarted. The re-read of the configuration file `server-config.wsdd` with the common `kill -HUP PID` command does not work for the embedded WebSphere Application Server. So changes on this file require a complete new start of the server. This works for maybe for syslog ([**AdminTasks**]) daemons, but unfortunately not for this server.

6.3 Realization of an autonomic AgreementFactory Service

This section introduces the *'autonomic factory approach'* and provides specifications for the creation of a prototype.

6.3.1 Autonomic factory approach

The WSRF uses WS-Resources to provide stateful Grid Services. Such WS-Resources are created by a factory, a concept that has been introduced in OGSA. But the service-specific factory requirements vary too much to allow a general-purpose factory message exchange to be usefully defined. Hence, different services need different parameters for the creation of the service. As a consequence, the [**WS-ResourceLifetime**] specification that defines the lifecycle and identity of WS-Resources defines no explicit message exchanges representing the function of a WS-Resource factory. Instead the specification refers to the creation of WS-Resources by a *'factory pattern'*.

The *'factory pattern'* defines an abstract factory that is capable of bringing a WS-Resource into existence. According to **[WS-ModelingResources]** such a WS-Resource factory can be any kind of Web Service capable of bringing a WS-Resource into existence, which includes:

- creating a new stateful WS-Resource
- assigning an identity to the new WS-Resource
- creating the association between the new stateful WS-Resource and its associated Web Service

The response message of a WS-Resource factory operation contains a WS-Resource qualified endpoint reference, containing a stateful resource identifier that refers to the new WS-Resource ([**WS-ModelingResources**]). To conclude, the *'factory pattern'* can be encoded in a variety of different Web Service operations. Important is that these operations create one or more WS-Resources.

This *'factory pattern'* leads to a common design of including one `create` method into every service to implement the *'factory pattern'*. This `create` method is responsible for the instantiation of WS-Resources that are used by the correspondent service. To conclude, the usual approach of the *'factory pattern'* is that the factory is modeled as a part of the Grid Service. Figure 6.24 shows that the factory is a part of the same service that provides access for the WS-Resources.

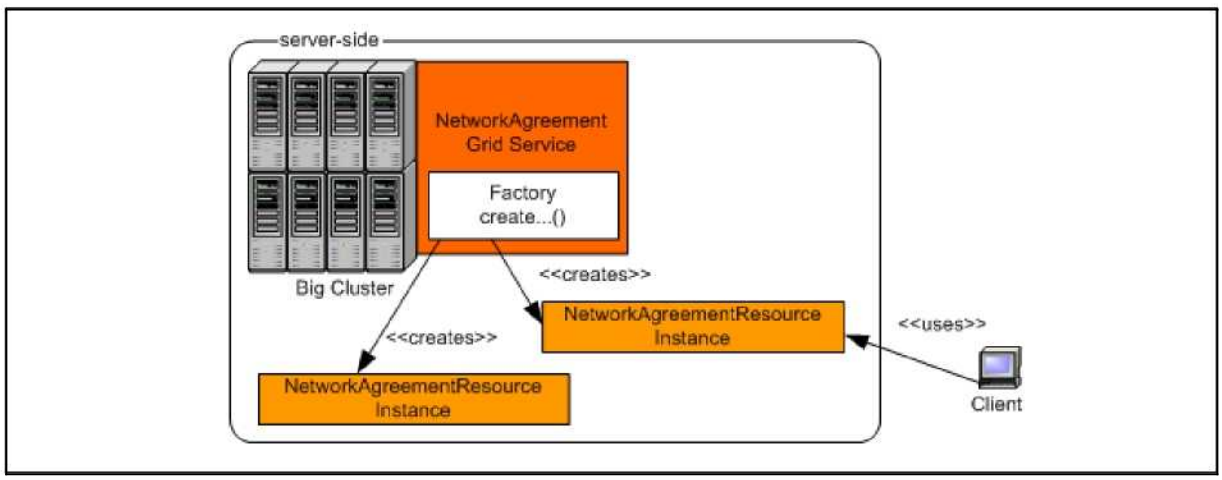

Figure 6.24: The factory pattern of WSRF.

The underlying idea of the *'autonomic factory approach'* is an enlargement of the normal WSRF *'factory pattern'*. This enlargement includes more functionality and more responsibility to advance the role of a factory in WSRF. The approach supports the creation of one central autonomic factory for multiple different services that are using one central policy decision point. The question on how much responsibility an autonomic factory should and must have has to be analyzed.

The *'autonomic factory approach'* is WSRF compliant, because it follows the demand for a WS-Resource factory that can be any Web Service capable of bringing a WS-Resource into existence ([**WS-ModelingResources**]). The slightly different design approach that is taken here is that the factory becomes more powerful by getting more functionality and responsibility. For instance, an autonomic factory has also properties as defined in [**WS-ResourceProperties**] and not only a create method for creating WS-Resources for a specific service. Note that all added functionality is related to creation issues, otherwise the idea of a factory would have been violated. The *'autonomic factory approach'* is modeled as an autonomic service that is not only responsible for the creation of WS-Resources related to one specific service. The autonomic factory provides create methods for several services and their WS-Resources. The motivation for establishing an autonomic factory is to have one central factory for different services. The motivation for establishing a factory that implements the *'factory pattern'* on the other hand, is that every service needs his own factory. Figure 6.25 shows that the autonomic factory is not a part of the service that is associated with the WS-Resources.

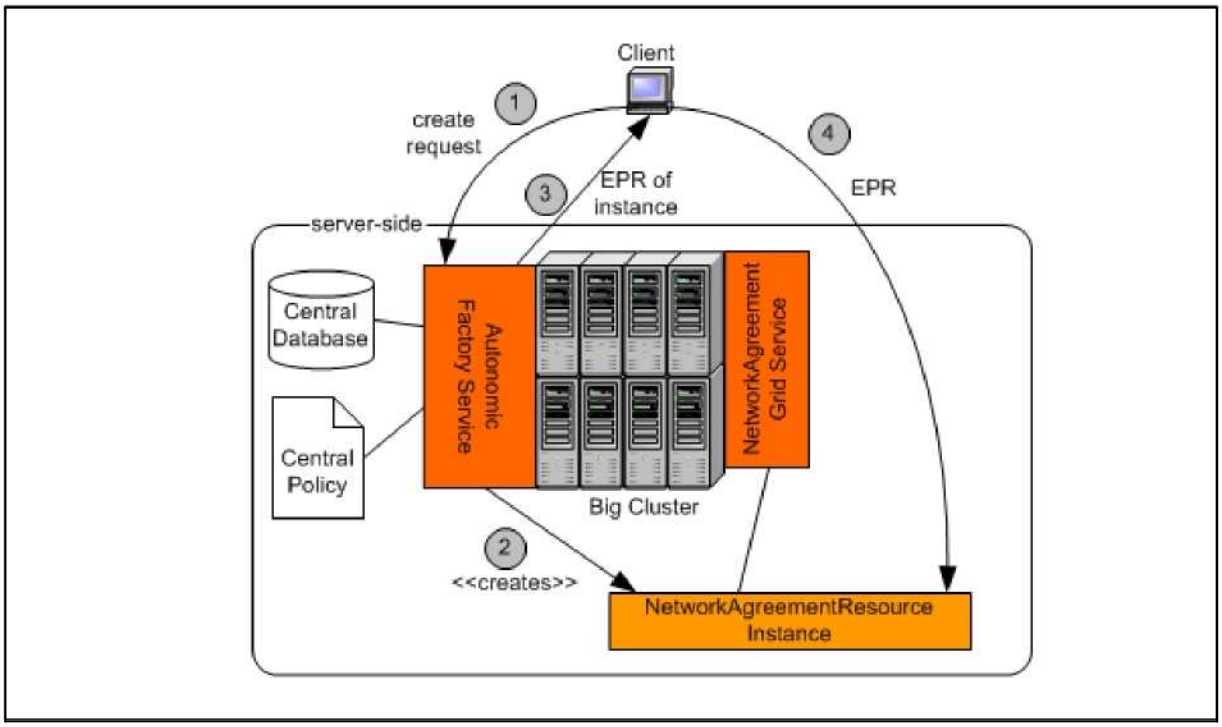

Figure 6.25: Example of an autonomic factory service.

The following enumeration provides an overview of the different approach and his posibilities:

- the factory is realized as an autonomic service
- the factory can provide read-only properties as defined in **[WS-ResourceProperties]**
- the factory can be deactivated without the change of services while all WS-Resources can be further used
- the factory provides a central location for hosting one central policy or database for different services

An autonomic factory service leads to the demand of changing the established EPR of the created WS-Resource, because another service provides the access for the WS-Resource. The service address inside the EPR can be easily updated to the correspondent service address to associate the created WS-Resource with another service. Suppose the party that hosts the autonomic factory service is different from the party that provides the service. This is impossible, because the factory service creates instances that must be reachable by the other services. To conclude, the factory and the service must be deployed in the same hosting environment. In addition the central autonomic factory allows the administration of all different service in the same hosting environment. Furthermore, it can store pieces of information related to evaluations of services use.

The autonomic factory service is a plain Web Service without any kind of state. As a consequence, the implementation of the `GetResourceProperty` method for example can be seen as a static implementation, because there is no WS-Resource that can be used for it. Hence, if the autonomic factory uses a WS-Resource it must implement the *'factory pattern'* leading to a factory that needs a factory. The lack of a WS-Resource to manage state leads to problems related to statelessness as described in Chapter 2. As a consequence, all properties of the autonomic factory must be read-only. In particular, an autonomic factory should not implement the `SetResourceProperty` method as defined in **[WS-ResourceProperties]**. But the addition of reduced **[WS-ResourceProperties]** functionality to a factory is still an important factor. The factory can expose information about the actual resource situation to give a service requestor the possibility to determine if it is usefull to request the service. Furthermore, the autonomic factory can expose information such as templates that can support the invocation process of services. Note that this is used in the **[WS-Agreement]** specification. Finally, the service can expose creation constraints for different services.

An advantage of an autonomic factory is that it can be deactivated for a well-defined period while all WS-Resources and other services are still working. For instance, the resource situation on a server cannot provide sophisticated QoS or the a kind of requested services and associated WS-Resources are deprecated and should not be instantiated aynmore. In such cases it might be usefull to deactivate the factory service or parts of the service, without changing other services. To conclude, no new WS-Resources can be created while old WS-Resources can be further used. This is usefull if the service provider established a service that is based on a negotiated agreement. Even if the service provider will not allow new agreements related to this agreements, the service provider must provide the service until old agreements are expired.

Another usage scenario of an autonomic factory is the establishment of one central policy or database. The maintainability of the central policy or database is easier, because only the autonomic factory should use them instead of each different service. For instance, it might be

usefull to define restrictions inside a policy for several service requestors that can be identified by an EPR. For instance, a create request of specific service requestors can be rejected with the help of such a policy. The reason can be underpayment by a well-known service requestor. Policies or database access are also possible with implementations of the *'factory pattern'*, but the autonomic factory provides one single policy or database for all services instead of one single service. Furthermore, the central decision point inside the autonomic factory allows an advantageous design of WSRF architectures. Finally Figure 6.26 shows the autonomic factory as a central decision entity in relation to other WS-Resources and services.

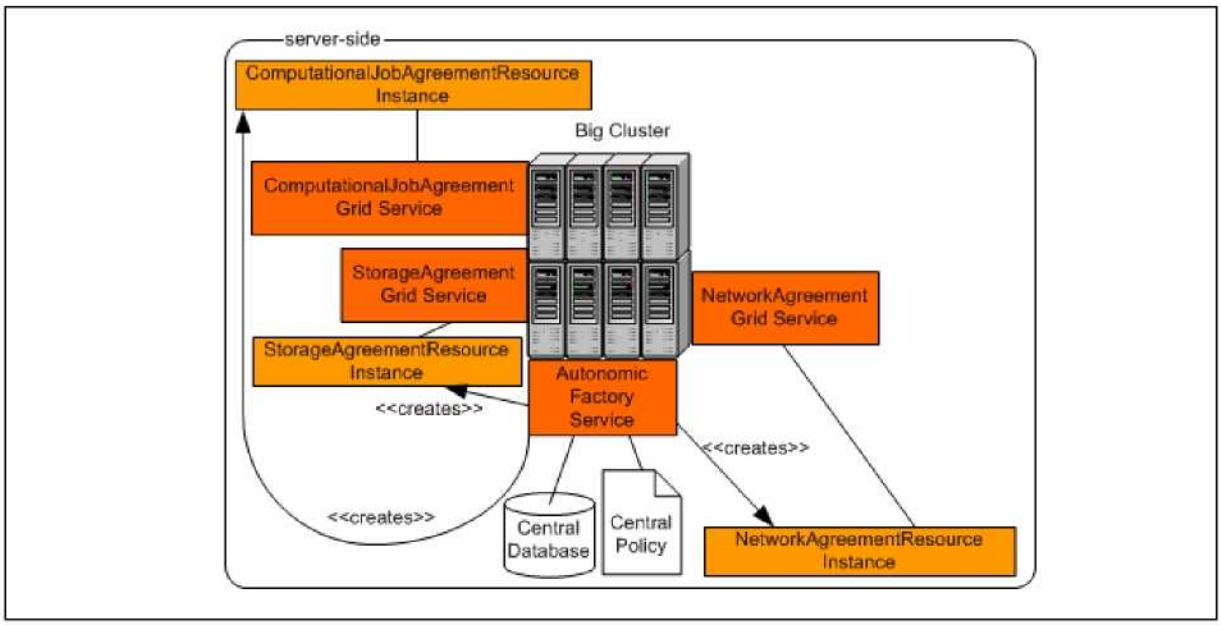

Figure 6.26: Autonomic factory service as central creation and decision point.

6.3.2 Description of the service's interface

The description of the `JuelichAgreementFactory` service is splitted into three different WSDL files as shown in Figure 6.27. In addition, the WSDL files using several XML schema elements and types that are defined in the `agreement_types.xsd`.

The name of the `JuelichAgreementFactory` service is used to emphasize that this kind of factory represents the *'autonomic factory approach'*. Hence, this factory can be used to create not only `NetworkAgreement` service instances. This factory can be also responsible to create instances for the `StorageAgreement` or `ComputationalJobAgreement` services. To conclude, the name `JuelichAgreementFactory` service is used to emphasize that the factory is an domain specific implementation of an autonomic factory that is responsible for service creation in the domain `Juelich`.

agreement_factory_port_type.wsdl

The **[WS-Agreement]** specification defines an `AgreementFactory` portType that can be found inside the `agreement_factory_port_type.wsdl` file. The portType is defined in the `wsag` namespace and includes the following two operations:

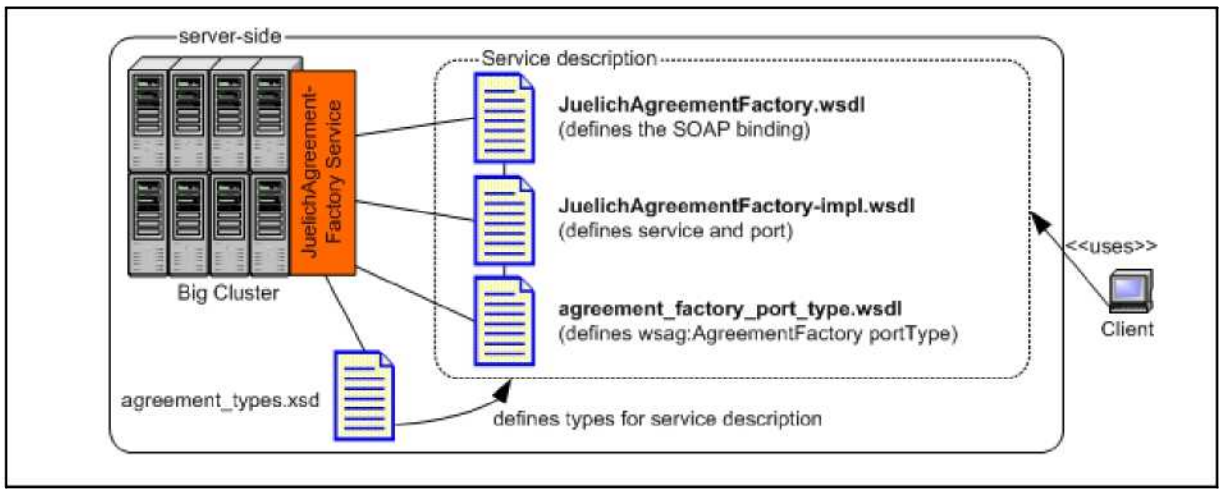

Figure 6.27: `JuelichAgreementFactory` service description files.

- `createAgreement`
  This operation is used to directly generate an agreement without any intervening negotiation ([**WS-Agreement**]).

- `GetResourceProperty`
  Operation as defined in the [**WS-ResourceProperties**] specification.

Furthermore, the `AgreementFactory` portType refers to exposed resource properties that are defined in the same WSDL file. In the `wsag` namespace are the resource properties for the portType defined, namely `AgreementFactoryProperties`. The specification defines only one resource property:

- `wsag:Template`

This resource property can be accessed through the `GetResourceProperty` operation of the `AgreementFactory` portType. Finally, Figure 6.28 shows the `AgreementFactory` portType and the related `AgreementFactoryProperties`.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsag="http://www.ggf.org/namespaces/ws-agreement"
  xmlns:wsrp="http://www.ibm.com/xmlns/stdwip/Web-services/
              WS-ResourceProperties"
  targetNamespace="http://www.ggf.org/namespaces/ws-agreement" ...>
  ...
  <xs:element name="AgreementFactoryProperties"
              type="wsag:AgreementFactoryPropertiesType"/>
  <xs:complexType name="AgreementFactoryPropertiesType">
    <xs:sequence>
      <xs:element ref="wsag:Template"/>
    </xs:sequence>
  </xs:complexType>
  ...
  <wsdl:portType name="AgreementFactory"
      wsrp:ResourceProperties="wsag:AgreementFactoryProperties">
    <wsdl:operation name="GetResourceProperty">
      ...
    </wsdl:operation>
    <wsdl:operation name="createAgreement">
      ...
    </wsdl:operation>
    <!-- domain specific extensions -->
    ...
  </wsdl:portType>
</wsdl:definitions>
```

Figure 6.28: `AgreementFactory` portType ([**WS-Agreement**]).

The specified `AgreementFactory` portType includes only two operations, but there is the possibility to add other domain-specific relevant operations to the portType. To keep it simple, the same domain-specific extensions like within the `NetworkAgreement` service definition are used, namely:

- `GetMultipleResourceProperties`
  Operation as defined in [**WS-ResourceProperties**].
- `QueryResourceProperties`
  Operation as defined in [**WS-ResourceProperties**].

Note that also other user-defined operations can be defined to support domain-specific functionality for the factory. For instance, operations that provide access to domain-specific templates for service description.

JuelichAgreementFactory.wsdl

The description of operations inside the `agreement_factory_port_type.wsdl` is not related to a specific protocol. Therefore, the `JuelichAgreementFactory.wsdl` file contains a binding for the SOAP protocol, named `JuelichAgreementFactoryBinding`. This binding defines message format and protocol details for operations and messages defined by the `AgreementFactory` portType ([**WSDL 1.1**]). The definition of the `AgreementFactory` portType is imported into the `JuelichAgreementFactory.wsdl` file as shown in Figure 6.29. Furthermore, the binding references the portType that it binds using the `type` attribute. The purpose of the `<soap:binding>` element is to signify that the binding is bound to the SOAP protocol format ([**WSDL 1.1**]). The `transport` attribute indicates which transport of SOAP this binding corresponds to. In this case HTTP. The SOAP binding of the `JuelichAgreementFactory` service is document-oriented. The complete WSDL file can be found in Appendix C. Finally, Figure 6.29 shows the important parts of the SOAP binding.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsag="http://www.ggf.org/namespaces/ws-agreement"
  targetNamespace="http://www.ggf.org/namespaces/ws-agreement" ...>
  ...
  <import location="http://localhost:4400/wstk/
      wsag/agreement_factory_port_type.wsdl"
      namespace="http://www.ggf.org/namespaces/ws-agreement"/>

  <binding name=JuelichAgreementFactoryBinding"
           type="wsag:AgreementFactory">

    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http"/>

    <wsdl:operation name="createAgreement">
      <soap:operation
          soapAction="http://www.ggf.org/namespaces/
                      ws-agreement/createAgreement"/>
        <wsdl:input>
          <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
          <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    ...
  </binding>
</definitions>
```

Figure 6.29: SOAP binding of the `AgreementFactory` portType.

Figure 6.29 shows also the `createAgreement` operation in detail. Note that all other operations are defined in the same style as the `createAgreement` operation. This service definition uses also the `literal` attribute of its operations for `<soap:body>` elements. Remember, the attribute indicates that each part of the input or output references has a concrete schema definition. In this case the concrete schema definitions of the operations can be found inside the `agreement_factory_port_type.wsdl` file.

JuelichAgreementFactory-impl.wsdl

The unique name of the service is defined inside the `<service>` element. The `<service>` element can be found in the `JuelichAgreementFactory-impl.wsdl` file. The `<port>` element references the unique `JuelichAgreementFactoryBinding` binding. For the access of the `JuelichAgreementFactoryBinding` definition the `JuelichAgreementFactory.wsdl` is imported. Furthermore, the service can be found at the URL `http://localhost:4400/wstk/services/JuelichAgreementFactory`. Finally, Figure 6.30 shows the `<service>` and `<port>` element. Note that Appendix C contains a complete listing of `JuelichAgreementFactory-Impl.wsdl`.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions...>
  <import location="http://localhost:4400/wstk/
     wsag/JuelichAgreementFactory.wsdl"
    namespace="http://www.fz-juelich.de/namespaces/
                          wsag/JuelichAgreementFactory"/>

  <service name="JuelichAgreementFactory">
    <documentation>Simple AgreementFactory Service to illustrate
    GGF GRAAP agreement negotiations.</documentation>
    <port binding="interface:NetworkAgreementBinding"
              name="AgreementFactoryPort">
       <soap:address location=
         "http://localhost:4400/wstk/services/JuelichAgreementFactory"/>
    </port>
  </service>
</definitions>
```

Figure 6.30: Naming the service as `JuelichAgreementFactory` service.

6.3.3 Specification of the service's implementation

The specification of the service's implementation is very close related to the description of the service interface. Remember, the service description defines [**WS-Agreement**] compliant functionality, but also operations to address domain-specific requirements. Therefore, this service implementation also organizes the functionality in both `ggf.wsag` and `fzjuelich.wsag` package. Figure 6.31 provides an overview of both packages.

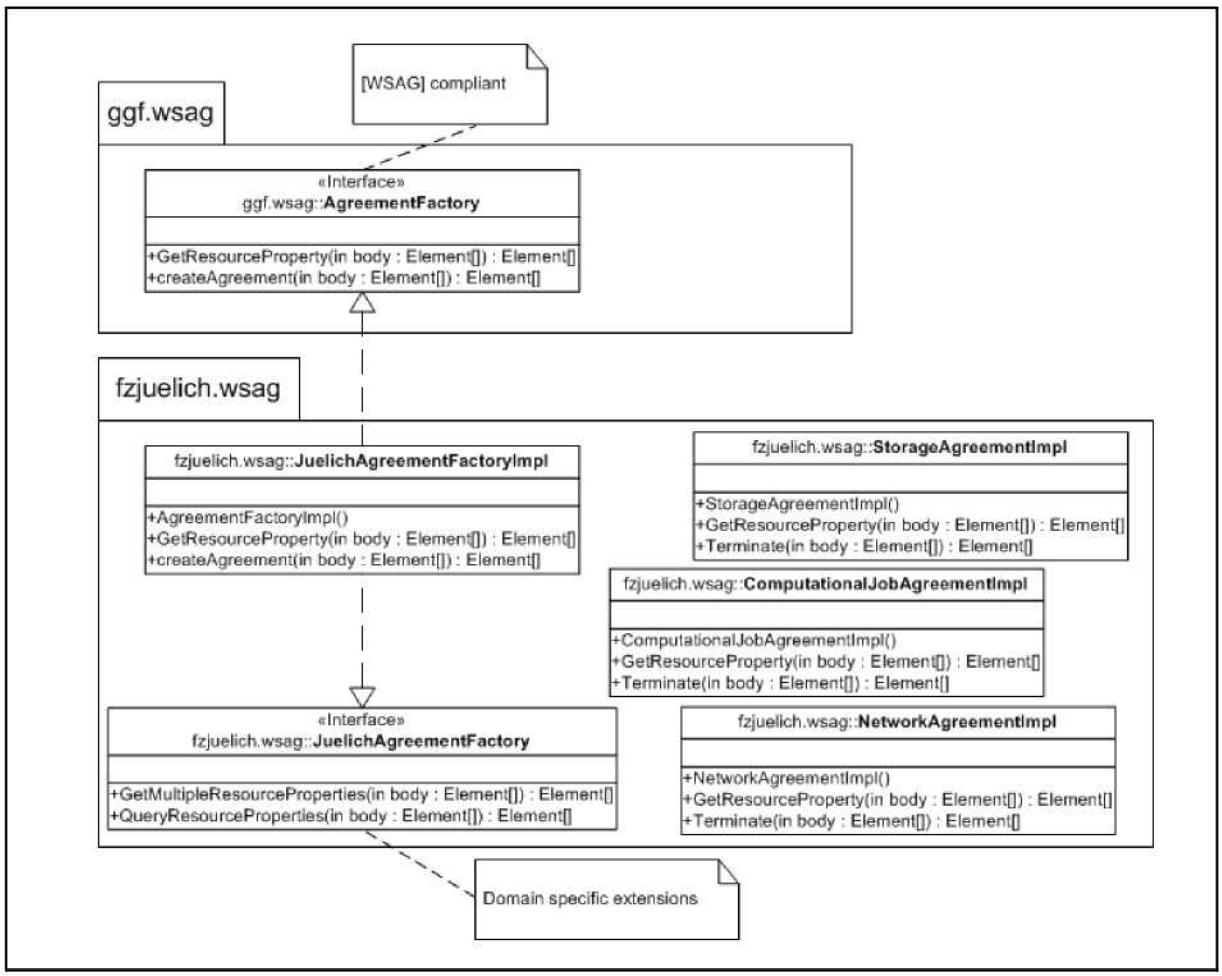

Figure 6.31: Specification of the `JuelichAgreementFactory` service implementation.

Figure 6.31 shows an interface `AgreementFactory` that contains the two methods that are defined for an **[WS-Agreement]** compliant agreement, namely:

- `GetResourceProperty`
- `createAgreement`

Hence, this interface is part of the **[WS-Agreement]** compliant `ggf.wsag` package. Furthermore, the interface `JuelichAgreementFactory` is specified that should contain any kind of domain-specific methods related to the implementation of the `JuelichAgreementFactory` service. Therefore, the interface includes both methods that were introduced in the WSDL service description:

- `GetMultipleResourceProperties`
- `QueryResourceProperties`

Note that this interface can also include local domain-specific methods that are not exposed as Web Service operations in the WSDL file. For instance, database access methods that can be used to check some kind of policy during an invocation of the exposed Web Service operations. To conclude, the design approach also allows the definition of local operations that cannot be invoked remotely. In addition the new methods can be used to provide access to

more than one usual template inside the resource properties. Even if the definition inside the **[WS-Agreement]** specification describes only one template per factory. However, to extend the factory to an autonomic factory, it should be possible to get templates for other agreement services such as `ComputationalJobAgreement` or `StorageAgreement` service.

The benefit of the differentiated interfaces approach is that implementations have both domain-specific and **[WS-Agreement]** compliant identity. Hence, every class that implements the interface `AgreementFactory` provides **[WS-Agreement]** compliant functionality. To provide well-defined domain-specific functionality related to the creation of services, the same class can also implement an domain-specific interface, for instance the `JuelichAgreementFactory` interface. The `JuelichAgreementFactoryImpl` class represents the implementation of the `JuelichAgreementFactory` service and implements both the `AgreementFactory` and the `JuelichAgreementFactory` interface. As a consequence, every operation invocation of the service invokes one method of the `JuelichAgreementFactoryImpl` class.

6.3.4 Implementation of the service

This section provides an overview of the important interfaces and important source elements of the service. Any agreement class that implement the `ggf.wsag.AgreementFactory` interface is a **[WS-Agreement]** compliant factory. This interface is shown in Figure 6.32.

```
package ggf.wsag;

import org.w3c.dom.Element;

public interface AgreementFactory
{
  public Element[] GetResourceProperty(Element[] body)
  throws Exception;

  public Element[] createAgreement(Element[] body)
  throws Exception;
}
```

Figure 6.32: Java interface `ggf.wsag.AgreementFactory`.

To support domain specific extensions another interface is used, called `JuelichAgreementFactory`. Note that this interface is not a part of the official `ggf.wsag` package, because this interface can include methods related to the creation of agreements in a domain-specific matter. Therefore, the domain-specific package `fzjuelich.wsag` contains this interface. The interface is shown in Figure 6.33.

```
package fzjuelich.wsag;

import org.w3c.dom.Element;

public interface JuelichAgreementFactory
{
  public Element[] GetMultipleResourceProperties(Element[] body)
  throws Exception;

  public Element[] QueryResourceProperties(Element[] body)
  throws Exception;
}
```

Figure 6.33: Java interface `fzjuelich.wsag.JuelichAgreementFactory`.

The implementation of the service is called the `JuelichAgreementFactoryImpl` class. This class implements the **[WS-Agreement]** compliant interface `AgreementFactory` and also the domain-specific `JuelichAgreementFactory` interface. Figure 6.34 shows the implemented methods of both interfaces.

```
package fzjuelich.wsag;

public class JuelichAgreementFactoryImpl
implements AgreementFactory, JuelichAgreementFactory
{
  ...
  public Element[] GetResourceProperty(Element[] body)
  throws Exception {...}

  public Element[] createAgreement(Element[] body)
  throws Exception {...}

  public Element[] GetMultipleResourceProperties(Element[] body)
  throws Exception {...}

  public Element[] QueryResourceProperties(Element[] body)
  throws Exception {...}

  ...
}
```

Figure 6.34: `JuelichAgreementFactoryImpl` class.

Using plain Web Services for [WS-ResourceProperties]

Remember, the `NetworkAgreementImpl` methods use the `request` object funcionality to implement resource property requests as defined in **[WS-ResourceProperties]**. This can be done, because the implementation of the service consists of a WS-Resource that can be used with this mechanism. A factory is responsible to create such WS-Resources for services. Such a factory is the `JuelichAgreementFactory` which contains no WS-Resource for any kind of resource propery requests. Hence, the `request` object functionality cannot be used in the

methods inside the `JuelichAgreementFactoryImpl` class that are related to resource property requests. That leads to a plain Web Service approach. This is no problem, because the `GetResourceProperty` method provides only one static template in this case. Note that this method must be changed when the factory is responsible for the creation of different services. In particular, the method must provide access to different templates related to the different services that can be created by an autonomic factory. For instance, different templates for the `ComputationalJobAgreement` or `StorageAgreement` service. However, to keep it simple this method provides only access to one template that is stored in a `network_template.xml` file. The method use DOM elements to send an template back to the service requestor, including the XML template file. More information about the DOM can be found in [**DOM**].

```
public class JuelichAgreementFactoryImpl
implements AgreementFactory, JuelichAgreementFactory
{
  ...
  public Element[] GetResourceProperty(Element[] body)
  throws Exception {
    ...

    Document respDoc =
      Utils.createDocumentFromURL("http://localhost:4400/wstk/wsag/
         network_template.xml");

    Element respElement =
      respDoc.createElementNS("http://www.ibm.com/xmlns/
        stdwip/Web-services/
          WS-ResourceProperties","GetResourcePropertyResponse");

    respElement.appendChild( respDoc.importNode(
      respDoc.getDocumentElement(), true));

    SOAPBodyElement sbe = new SOAPBodyElement(respElement);

    return new Element[] sbe.getAsDOM();
  }
  ...
}
```

Figure 6.35: Implementation parts of the `GetResourceProperty` method.

Because the generation of the response is not created automatically, the namespace of the service method must be added to the response at first as shown in Figure 6.35.

createAgreement

Another method of the `JuelichAgreementFactory` class is the `createAgreement` method as defined in the [**WS-Agreement**] specification. Surely this method is the most important method of the factory. At first the method checks if the request contains an agreement-offer that the service can accept. For instance, the creation constraints of the correspondent template can be checked if they are met within the agreement-offer. When the offer is accepted the `createResourcePropertiesTree` method is used to transform the agreement-offer into resource properties as defined in [**WS-Agreement**]. In particular, this method uses the `body` pa-

rameter to retrieve the agreement-offer and translate it into the agreement structure by using resource properties. The result of this method is an XML tree with resource properties. This properties are used as a parameter for the creation of an `ResourcePropertiesManagerDefaultImpl` instance. Remember this class is used to provide a mechanism to access resource properties. As introduced in the `NetworkAgreement` service specification, the WS-Resource consists of three parts. One of this three parts is the central manager of the WS-Resource, namely: `NetworkAgreementResource`. The instance of the `ResourcePropertiesManagerDefaultImpl` is used as a parameter for the creation of a `NetworkAgreementResource` instance. Figure 6.36 shows a portion of correspondent source in the `createAgreement` method.

```
public class JuelichAgreementFactoryImpl
implements AgreementFactory, JuelichAgreementFactory
{
  ...
  public Element[] createAgreement(Element[] body)
  throws Exception {
    ...
    String checkResult = checkCreateAgreementRequest( body );

    if (checkResult.equals("")) {

      Element wsrpTree = createResourcePropertiesTree( body );

      ResourcePropertiesManagerDefaultImpl wsrpResource =
        new ResourcePropertiesManagerDefaultImpl(wsrpTree);

      Resource agreementResource =
        new NetworkAgreementResource(wsrpResource);
      ...}
  }
}
```

Figure 6.36: Implementation parts of the `createAgreement` method.

The aim of a factory method in WSRF is to provide an EPR that can be used to address the created WS-Resource. Note that the *'autonomic factory approach'* means that the factory itself is a standalone service. Hence, the `NetworkAgreement` service is another service. So the creation of an EPR must consist of an address that provides access for the WS-Resource through the `NetworkAgreement` service. As a side-remark, this is one of the most important differences between the usually implementations of the *'WSRF factory pattern'*, because the service URL stays normaly the same. This is accomplished by defining only one `create` method in the same service that provides access for the WS-Resources. However, the service design of the `JuelichAgreementFactory` is using the *'autonomic factory approach'* that leads to an EPR containing an URL pointing on the `NetworkAgreement` service. The creation of a real resource that can be addressed via an EPR is accomplished by the `createResource` method of the `LifetimeManager`. In addition, some DOM functionality and elements are used to define the namespace and the structure of the response as defined in **[WS-Agreement]**. Figure 6.37 shows the creation of the EPR and the construction of the response.

```
public class JuelichAgreementFactoryImpl
implements AgreementFactory, JuelichAgreementFactory
{
  ...
  public Element[] createAgreement(Element[] body)
  throws Exception {

    ...
    EndpointReference epr =
      wsResourceLifetimeManager.createResource(agreementResource,
           null, new URL("http://localhost:4400/wstk/
                services/NetworkAgreement"));

    Document respDoc = Utils.createEmptyDocument();

    Element respElement =
      respDoc.createElementNS("http://www.ggf.org/namespaces/
          ws-agreement", "createAgreementResponse");

    Element createdAgreementEPR =
      epr.toDOM("wsag:CreateAgreementOutputType/
         createdAgreementEPR","createdAgreementEPR");
    ...
    }
  ...
}
```

Figure 6.37: Implementation parts of the `createAgreement` method.

6.3.5 Deployment

To work properly with the embedded hosting environment of the ETTK remember the following environment variables, namely:

- `$ETTK_HOME`
  used to refer to the home of the ETTK installation
- `$DEPLOY_HOME`
  used to refer to the root directory for deploying services

Deployment of the service

The hosting environment of the ETTK uses the AXIS SOAP engine. The deployment of the service is inside the `$DEPLOY_HOME/Web-INF/server-config.wsdd` file. Figure 6.38 shows the deployment description of the `JuelichAgreementFactory` service.

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
    ...
    <service name="JuelichAgreementFactory" provider="java:MSG">
        <parameter
            name="className"
            value="fzjuelich.wsag.JuelichAgreementFactoryImpl"/>
        <parameter name="allowedMethods" value="*"/>
    </service>
    ...
</deployment>
```

Figure 6.38: Deployment of the `JuelichAgreementFactory` service.

The `<service>` element is used to deploy the `JuelichAgreementFactory` service that is provided as a message service, using the `provider="java:MSG"` attribute. As shown in Figure 6.38, the first parameter defines the class that implements the Web Service. The implementation of the service interface is the `fzjuelich.wsag.JuelichAgreementFactoryImpl.class`. The value `*` indicates that all methods are exposed as Web Service methods. Note that the WS-Addressing handler must also be deployed, because the `JuelichAgreementFactory` service uses also the WS-Addressing functionality.

Beside the deployment description with the use of the WSDD language, there must be also a deployment location for the description of the service that is written in WSDL. The `$DEPLOY_HOME/wsag` location is used to deploy all WSDL or related XML schema files. Hence, this directory contains all the files that are related to the description of the `JuelichAgreementFactory` service. This files are shown in the following enumeration:

1. JuelichAgreementFactory.wsdl
2. JuelichAgreementFactory-impl.wsdl
3. agreement_factory_port_type.wsdl
4. agreement_types.xsd

Deployment of used libraries

Inside the `Web-INF` directory is the `lib` directory that contains the libraries used by the service:

- `wsrf.jar`
  API for the WSRF functionality
- `wsa.jar`
  API for the WS-Addressing functionality

Testing the deployment

Remember, the test if all steps are correctly done is also a part of the deployment procedure. By using the shell script `$ETTK_HOME/bin/startserver.sh` the embedded application server is started. For testing the server and also the `JuelichAgreementFactory` service the `http:`

`//localhost:4400/wstk/services/JuelichAgreementFactory` can be visited by a normal Web-browser. `AXIS error - No service is available at this URL` means that there is no service at this URL. This implies that configurations in the `server-config.wsdd` file are wrong or maybe some typo's are inside the WSDL files. On the other hand the message `Service JuelichAgreementFactory is deployed.` tells the developer that his configurations and deployment procedure was correctly done. Remember that the description of the Grid Service can be found using the `http://localhost:4400/wstk/wsag/JuelichAgreementFactory.wsdl`. Note that working with a remote server demands the exchange of `localhost` with the obversely Server IP address.

6.4 Prospects

The emerging new technologies based on WSRF and other related Web Service technologies allow many prospects for further implementations. Some of them are discussed within this section.

6.4.1 Integration of WS-Security

Security for Web Services is an important issue and necessary in many scenarios. One security issue is a situation in that a service provider wants to make sure that the service requestor cannot deny that the service request comes from him. Similarly, a service requestor wants to make sure that for the response message. This security requirement is called non-repudiation ([**ETTK-Doc**]). A typical example for this security requirement is the purchase order scenario. In particular, the requestor sends a purchase order to the service provider and receives an acknowledgment of the order. Hence, the service provider wants to make sure that the requestor cannot deny his sent of the purchase offer while the requestor wants to make sure that the provider cannot deny his sending of an acknowledgement. To satisfy the non-reputation requirement digital signatures can be used.

The integration of a non-repudiation security requirement is very simple, because the ETTK for instance provides AXIS handlers that implement XML-Signatures. Note that normal Secure Sockets Layer (SSL) cannot be used for non-repudiation. Therefore a special [**WS-Security**] specification was defined. The AXIS handler chains provide a rich mechanism for implementation of SOAP security ([**ETTK-Doc**]) that allows complex security functionality. The signature and verification handlers are deployed so that each outgoing message from the client is signed. When the message is received at the server, the verification handler validates the signature of it. In particular, the verification handler proofs that the message has not been altered. Note that the message response could also be signed.

To conclude, the ETTK and possible other hosting environments provide many handlers that can be used in combination to create even more secure communication paths ([**ETTK-Doc**]). For instance, an encryption handler used in combination with a signature handler. This supports a secure communication path that provides non-repudiation as well as confidentiality. Another prospect related to complex security issues is the dealing with key management as defined in [**XML-KMS**]. Hence, there can be many prospects concerning WS-Security issues that are also important in a Grid environment.

6.4.2 Persistent agreements

Persistent agreements need a mechanism to store agreement resources and their properties inside a database. Hence, all databases are different, therefore there must be a framework that provides uniform data access in Grid environments. This leads to an database-independent design and persistent resources like an agreement. The Open Grid Services Architecture - Data Access and Integration (OGSA-DAI) is such a framework that aims to provide Middleware for exposing data resources such as XML and relational databases in a Grid environment. The benefit of OGSA-DAI is that its services wrap heterogeneous data resources to provide a uniform data access interface. Interesting is that the underlying data model is not hidden ([**OGSA-DAI**]). As a consequence, a relational data resource will not masquerade as an XML data resource. In addition, there is also the possibility to access pure XML data resources. OGSA-DAI services can be used to create sophisticated higher-level services. Higher-level services can offer data federation and distributed query processing within a Grid environment. More information about distributed query processing can be found in [**OGSA-DQP**]. OGSA-DAI is participating in the standardization of data access interfaces through the GGF Database Access and Integration Services Working Group (DAIS-WG).

OGSA-DAI provides software that can be considered logically as a number of cooperating Grid Services ([**OGSA-DAI**]). Each Grid Service acts as a proxy for the systems that hold the data. The data can be stored in a relational database, for instance MySQL. Furthermore, the data can be stored in a pure XML database such as Xindice. Therefore, the OGSA-DAI Grid Services provide access to the data held within such databases.

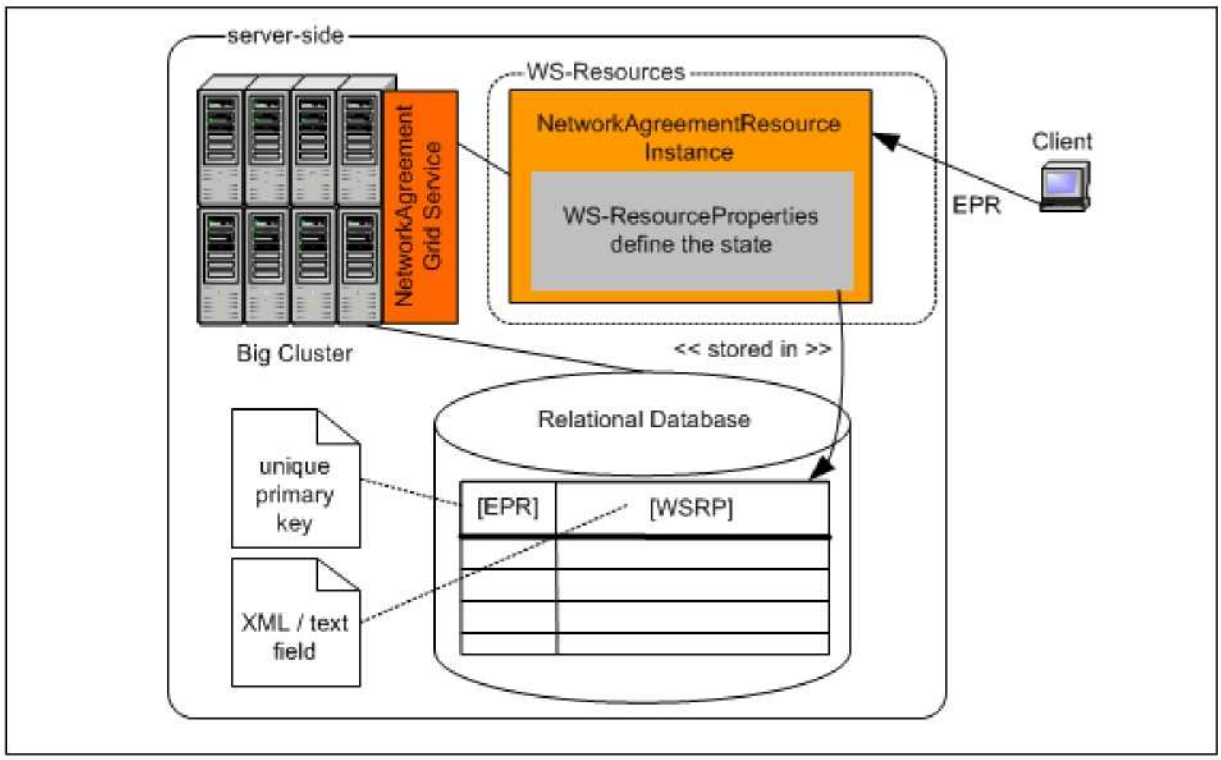

Figure 6.39: Using the EPR as a primary key in relational databases.

Figure 6.39 shows how to store resource properties in a relational database using the EPR of a resource as primary key. This primary key is unique, because every EPR consists of the `<ReferenceProperties>` element that uniquely defines a resource at the server-side. In addition, a long text database field contains the resource properties, usually an XML tree. To conclude, the OGSA-DAI framework can be easily adapted to the realized prototypes. Using this framework to create an persistent `NetworkAgreement` Grid Service is a good prospect for further work.

6.4.3 Client implementations and interoperability

One prospect of this thesis is the implementation of a client, including interoperability issues occuring in heterogenous Grid environments. To understand the relationship between the client and the server processes Figure 6.40 provides an overview of the simple agreement negotiation process. Note that the process is the simple agreement negotiations approach of **[WS-Agreement]** that consists of templates and the acceptance or rejection of agreement-offers.

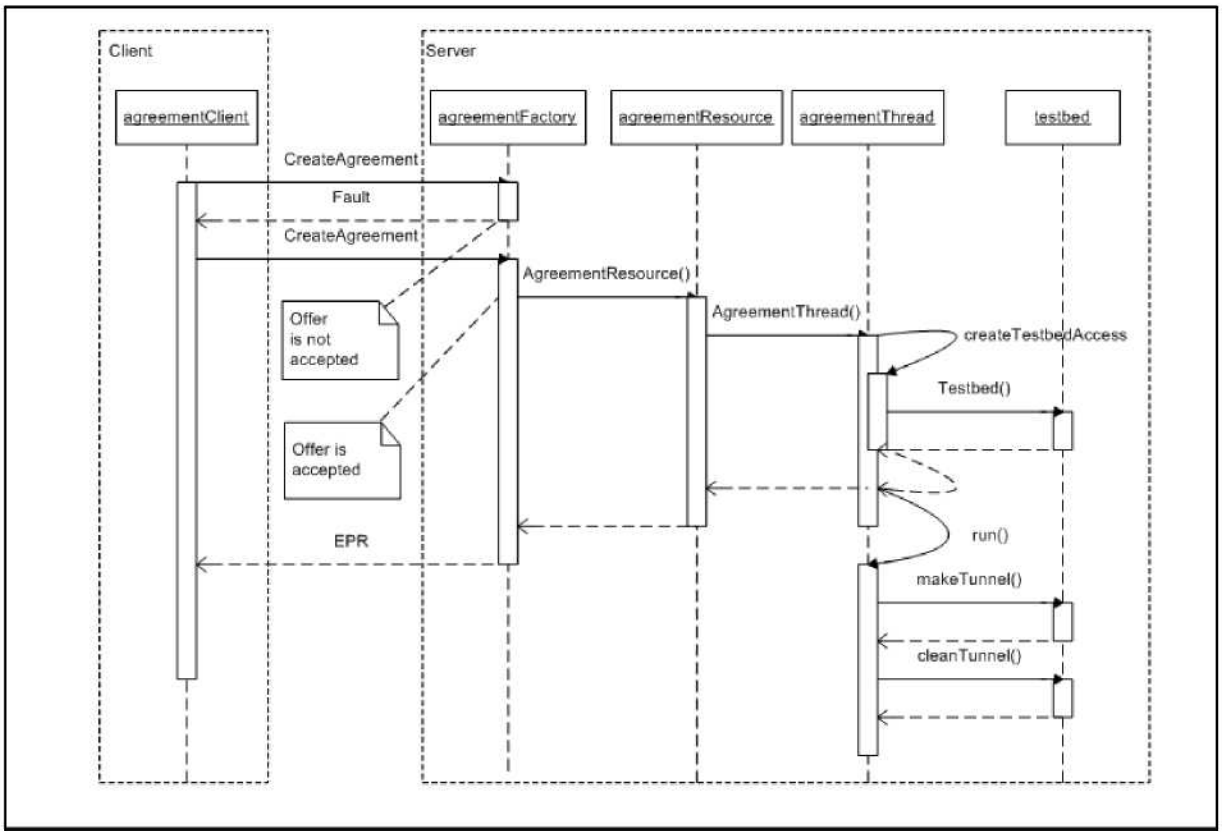

Figure 6.40: Simple negotiation.

Furthermore, Figure 6.40 shows the message exchange between an `AgreementThread` instance and the access to the `Testbed`. The whole figure demonstrates the creation of a network tunnel and its removal.

The prototypes of the services are implemented within the WSRF implementation inside the ETTK. The client implementation can use any implementation that can generate WSRF compliant message exchanges. For instance, the server implementation is running on Linux and is implemented with the `wsrf.jar` library of the ETTK. There can be a client in Windows that uses the WSRF.NET implementation to create WSRF compliant requests for WS-Resources that are provided by the server on the Linux machine. On the other hand there could be a small WSRF::Lite implementation for the client on a Linux machine that also creates WSRF compliant requests for WS-Resources on the same server. Hence, the combinations are endless. Important is that every client uses message exchanges as defined in [**WS-Agreement**] and [**WS-Adressing**]. More information about the interoperability of WSRF compliant services can be found in [**WSRF-Interop**].

6.4.4 Improving performance

One prospect can be the improvement of performance using HTTP compression. Suppose, a resource properties XML tree that contains addresses of all customers. If you start a `GetMultipleResourceProperty` request ([**WS-ResourceProperties**]) to get all addresses at once, it might be a good idea to compress the XML before sending it to the service requestor. Suppose, every address has elements such as `<name>`, `<surname>` or `<street>`. Because every subelement of the tree is an address that contains such elements, the whole resource properties tree contains a massive amount of the same elements that can be compressed very well using the Gzip compression or HTTP compression ([**HTTP-COMP**]). Another example suitable for compression uses an agreement template as defined in [**WS-Agreement**] that contain many `<ServiceDescriptionTerm>` elements.

6.4.5 Using more resource properties functionality

Another prospect for implementation can be the integration of the `SetResourceProperty` method as defined in [**WS-ResourceProperties**]. Suppose, there is an agreement switching network connections through a network. The bandwidth and other needed QoS related agreement terms for a network tunnel are already defined by the agreement. The endpoints of the network tunnel are the only missing pieces of information. Hence, the agreement can be created before the real endpoints of the network tunnel are defined. But important is that when the start-time of the service that represents this agreement is reached all information have to be present, including the missing pieces of information. For that purpose, the `SetResourceProperty` method can be used to add the missing agreement terms before the service starts.

The implementation of this method also includes negotiation aspects, because not every agreement term will be accepted by the method. In particular, the `SetResourceProperty` method can use the creation constraints of the template to make sure if the updated or newly created resource properties are allowed. There are many problems to solve when working with the `SetResourceProperty` method such as locking of the resource properties tree, if the EPR of the WS-Resource is published to more than one service requestor. Furthermore, the new agreement terms are not published through the WSDL file, because it defines only abstract resource properties that are corresponding to the structure of an agreement as specified in [**WS-Agreement**]. Hence, another client that uses the same EPR for the same WS-Resource cannot know what new resource properties are available.

One perspective can be the definition of the resource properties dynamically inside an XML schema file which is imported into the service description each time the service is invoked. However, the implementation of such functionality provides a good perspective for further implementations.

6.4.6 Integration of the WS-Notification family

The WS-Notification family provides a powerful framework that can be used to notify service requestors if some agreement guarantee terms cannot be reached anymore. Suppose, the defined bandwidth of an network agreement cannot be reached anymore due to disturbances in the network that lead to a service which cannot achieve his defined QoS. Hence, the service can send a `notify` message to the service requestor that the defined QoS cannot be reached anymore. After it, the service requestor may change the service provider to reach the demanded QoS by a similar service.

The WS-Notification family can be used in several service designs, for instance a multiple or shared agreement service design. The realization of such a design, in combination with the prototypes, provides a further prospect for future work. The ETTK has also an implementation of the WS-Notification functionality as defined in [**WS-PubSub**]. This thesis cannot give an introduction to all details of this functionality, but the essential point is that the *'Notification Model'* follows a topic-based publish/subscribe pattern. The interesting part is that the message is not initiated by the service requestor, like the known *'poll mechanism'*, rather the service provider initiates messages based on a subscription of a topic from a service requestor. Inside the ETTK are two architecture models available, but it is important to know that both models require a hosting environment, in this case the ETTK. That means that the notification producer and the notification consumer are deployed services of the embedded application server of the ETTK. In fact, there will be no way to get a `notify` message without a predeployed service. But it is possible to send a notification message directly to another service, without being inside of a hosting environment. However the whole functionality of WS-Notification is inside the library `wsrf.jar` of the ETTK.

The first architecture model, also called basic notification, describes a message exchange between two roles, the notification producer and notification consumer. In this model the notification consumer subscribes on a special topic to the notification producer. In this thesis the notification producer could be the agreement itself with some kind of internal state and an implemented publish/subscribe mechanism. When an internal state of the agreement is changed an event could be fired and used to publish a message on a topic. All on this topic subscribed notification consumers are being notified, after that they can start actions like renegotiation of service terms or some internal processing.

The second architecture model extends this basic scenario with one role, called notification broker which makes this model more interesting und useful. In this model the notification consumer subscribes on a topic on this notification broker. All published messages that are sent by notification producers are sent to the notification broker. Note that this notification broker is also a deployed service of the embedded application server of the ETTK. After the messages arrive at the notification broker, the broker, by the matching of the topics, forwards the messages to the notification consumers. The second architecture model has some advantages, for example the publisher, in this case, the agreement does not need the logic to keep track of subscriptions. Furthermore there could be multiple agreements, so a subscriber need

not to subscribe to each agreement, but only to the broker. So all the agreements publish the notifiy informations to the broker. That means that the location of the agreement service could change without affecting subscribers. This could be an important feature when a service is maybe delegating some parts of his own agreement terms to another service. So there are some kind of shared agreements, but all are sending notifications to one central notification broker and the consumer does not need to know that the agreement is shared. In fact it is possible that a notification consumer subscribes on a topic at the notification broker, even when the service publisher itself is not available at this time.

This was a brief overview of possibilities that can be realized using the power of the WS-Notification family. Note that also other WSRF hosting environments provide a WS-Notification family implementation.

Chapter 7

Conclusions and Future Work

This thesis evolves a *'WSRF compliant'* architecture that addresses flexible service offering in a Grid environment. The proposed architecture takes advantage of the refactoring of OGSI and the WS-Resource approach of WSRF. In the presented architecture, the advantageous technology framework WSRF is used to realize **[WS-Agreement]** compliant agreement negotiations by stateful Web Services, also known as Grid Services. Such Grid Services are well suitable to address the demand of flexible service offering in a Grid environment.

All negotiation aspects of the **[WS-Agreement]** specification were successfully demonstrated by the prototypes in a Grid environment. These prototypes were implemented with the use of a sophisticated WSRF hosting environment. The **[WS-Agreement]** proposals such as the service description terms and guarantee terms of an well-defined agreement structure, allow an advantageous formal way of defining the structure of an agreement. Agreement negotiations realized via templates, faults and offers are leading to an accepted agreement that can be modeled as an WS-Resource that uses this well-defined agreement structure. Therefore, the WSRF Grid Services infrastructure was used to interact in a well-defined, **[WS-Agreement]** compliant way with these WS-Resources. To conclude, it is possible to create and to use *'WSRF compliant Grid Services'* that are also following the proposals of the **[WS-Agreement]** specification.

The *'autonomic factory approach'* was presented as an enlargement of the standard WSRF *'factory pattern'*. The design approach integrates more functionality and more responsibility into the factory than the factory pattern. As a consequence, the role of the factory was extended in the realization of the prototypes. The analysis of how much responsibility an autonomic factory should and must have and the resulting design lead to advantages such as using one central policy or database for decisions. On the other hand, the analysis emphasizes some restrictions of an autonomic factory such as the deployment of the factory in the same hosting environment or the restriction of having only read-only properties. However, the design approach supports the creation of one central autonomic factory for multiple different services. To conclude, the *'autonomic factory approach'* can be very useful for huge server clusters that provide a massive number or different services. In addition, even a small server cluster with a little number of different services can provide extensibility with the use of an autonomic factory.

The restrictions of the *'autonomic factory approach* are still an open problem. A workaround would be to implement the normal WSRF *'factory pattern'* inside the factory. This allows the creation of WS-Resources that can solve some restrictions of an autonomic factory, but a sophisticated solution to solve all restrictions is still missing. Furthermore, the workaround leads to an unsuitable design wherein a factory needs another factory.

Another open problem related to agreement negotiations is the demand for re-negotiation of agreement terms during the lifetime of an agreement. Hence, the agreement service must provide its functionality while some terms of the agreement are actually re-negotiated.

This thesis provides an overview of prospects that are related to the combination of WS-Agreement and WSRF technologies. Hence, the realization as well as problems of the realization of these prospects can be analyzed in the future. Furthermore, recent efforts on standardizations show that Web Service specifications are designed to be composable. For instance, WS-Security can be easily adapted to current Web Service realizations. Hence, new specifications related to Web Services or Grid Services can maybe easily composed or adapted to the current implementation of the prototypes. In addition, many specifications are not standards yet, for instance **[WS-Agreement]** or **[WSRF]**. The standardization process of such specifications will evolve new issues related to the technology presented inside this specifications. For instance, when the WSRF framework will be accepted by OASIS, there may be also changes to the WSRF framework that have to be integrated into the prototypes. To conclude, all efforts related to the maintenance of recently accepted or still not accepted specifications such as **[WS-Agreement]** or **[WSRF]** can be also an interesting motivation for future realizations.

An interesting aspect of future work will be also the integration of more functionality into the `NetworkAgreement` service prototype. This includes new specifications of WSRF or Web Service standards and also new proposals of the GRAAP working group. For instance, the realization of the proposed GRAAP layered architecture in the current working draft concerning WS-Negotiations that will probably include aspects such as re-negotiation of agreement terms. Finally, interesting future work will be the integration of all new Web technologies that can be usefull in Grid environments.

List of Acronyms

AC Autonomic Computing

AI Artificial Intelligence

API Application Programming Interface

ASP Active Server Pages

AXIS Apache eXtensible Interaction System

B2B Business-to-Business

BPEL Business Process Execution Language

CERN Centre Européen de Recherche Nucléaire

CLR Common Language Runtime

CORBA Common Object Request Broker Architecture

COM Component Object Model

CPU Central Processing Unit

CTS Common Type System

DAIS-WG Database Access and Integration Services Working Group

DCOM Distributed Component Object Model

DIS Distributed Interactive Simulation

DOM Document Object Model

DOS Disk Operating System

EDI Electronic Data Interchange

EJB Enterprise Java Beans

EPR Endpoint Reference

ETTK Emerging Technologies Toolkit

FTP File Transfer Protocol

GDSS Grid Data Service Specification

GGF Global Grid Forum

GRAAP Grid Resource Allocation Agreement Protocol

GT3 Globus Toolkit 3

GT4 Globus Toolkit 4

LHC Large Hadron Collider

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IDL Interface Description Language

IIOP Internet Inter-ORB Protocol

IIS Internet Information Services

IP Internet Protocol

JAR Java Archive

J2EE Java 2 Platform Enterprise Edition

JDK Java Developer Kit

JMS Java Messaging Service

JSDL Job Submission Description Language

JSP JavaServer Pages

LAN Local Area Network

MIDL Microsoft Interface Description Language

MIH Message Information Header

MIME Multipurpose Internet Mail Extensions

MRI Magnetic Resonance Imaging

MSDE Microsoft SQL Server Desktop Engine

NEES Network for Earthquake Engineering and Simulation

NICE Narrative Immersive Constructionist/Collaborative Environments

OASIS Organization for the Advancement of Structured Information Standards

ODBC Open Database Connectivity

OGSA Open Grid Services Architecture

OGSA-DAI Open Grid Services Architecture - Data Access and Integration

OGSI Open Grid Services Infrastructure

OMA Object Management Architecture

OMG Object Management Group

QoS Quality of Service

ORB Object Request Broker

OS Operating System

OWL Web Ontology Language

PID Process ID

QName Qualified Name

RDF Resource Description Framework

RMI Remote Method Invocation

RPC Remote Procedure Call

SDK Software Development Kit

SDT Service Description Terms

SLA Service Level Agreement

SMTP Simple Mail Transfer Protocol

SOA Service Oriented Architecture

SOAP Simple Object Access Protocol

SRM Scheduling and Resource Management

SSL Secure Sockets Layer

STM Scanning Tunneling Microscope

TCP Transmission Control Protocol

UI User Interface

UDDI Universal Description, Discovery and Integration

UUID Universal Unique IDentifier

UNIX Uniplexed Information and Computing System

URI Uniform Resource Identifier

URL Uniform Resource Locator

VO Virtual Organizations

W3C World Wide Web Consortium

WAR Web Application Archive

WAN Wide Area Network

WS Web Services

WS-Addressing Web Services Addressing

WSDD Web Service Deployment Descriptor

WSDL Web Services Description Language

WSE Web Services Enhancements

WSFL Web Services Flow Language

WSIL Web Services Inspection Language

WSLA Web Service Level Agreement

WSRF Web Services Resource Framework

WSTK Web Services Toolkit

WWW World Wide Web

XHTML Extensible HyperText Markup Language

XML Extensible Markup Language

XPATH XML Path Language

References

[ADO.NET-Home]
Microsoft Corporation (2004)
.NET Framework Developer's Guide - Overview of ADO.NET
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconoverviewofadonet.asp

[AdminTasks]
L.R. Perlov, A. Rivera, E.W. Kyu (01/22/2003)
Useful Commands for WebSphere Application Server - Part 2: System Administration Tasks
http://www-106.ibm.com/developerworks/Websphere/library/techarticles/0301_rivera/rivera.html

[Anatomy]
I. Foster, C. Kesselman, S. Tuecke (2001)
The Anatomy of the Grid - Enabling Scalable Virtual Organizations
http://www.globus.org/research/papers/anatomy.pdf

[Ant]
The Apache Software Foundation (2000-2004)
Apache Ant 1.6.1 Manual
http://ant.apache.org/manual/index.html

[Apache-Addressing]
Apache Web Services (2004)
Apache Addressing
http://ws.apache.org/ws-fx/addressing/

[Autonomic]
A. G. Ganek, T. A. Corbi (2003)
The dawning of the autonomic computing era
http://www.research.ibm.com/journal/sj/421/ganek.pdf

[AXIS-Arch]
Apache Web Services Project, The Apache Software Foundation (2000-2003)
Axis Architecture Guide, Version 1.1
http://ws.apache.org/axis/java/architecture-guide.pdf

[AXIS-Home]
Apache Web Services Project, The Apache Software Foundation (2000-2004)
WebServices - Axis Home page
http://ws.apache.org/axis/index.html

[AXIS-Ref]
Apache Web Services Project, The Apache Software Foundation (2000-2003)
Axis Reference Guide, Version 1.1
http://ws.apache.org/axis/java/reference.pdf

[AXIS-User]
Apache Web Services Project, The Apache Software Foundation (2000-2003)
Axis User's Guide, Version 1.1
http://ws.apache.org/axis/java/user-guide.pdf

[BizTalk]
Microsoft (2004)
Microsoft BizTalk Server Homepage
http://www.microsoft.com/biztalk/

[Condor]
M. Litzkow, M. Livny, M. Mutka (06/1988)
Condor - A Hunter of Idle Workstations, Proceedings of the 8th International Conference of Distributed Computing Systems
http://www.cs.wisc.edu/condor/doc/icdcs1988.pdf

[Cookie-Spec]
Netscape (1999)
Persistent Client State - HTTP Cookies, Preliminary Specification
http://wp.netscape.com/newsref/std/cookie_spec.html

[Corba]
Object Management Group Inc. (03/27/2004)
CORBA/IIOP Specification, Object Management Group, Inc.
http://www.omg.org/technology/documents/formal/corba_iiop.htm

[Distributed]
A.S. Tanenbaum, M. van Steen (11/11/2002)
Distributed Systems - Principles and Paradigms, Prentice-Hall Inc.
ISBN:0-13-088893-1

[DL-Handbook]
F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider (2003)
The Description Logic Handbook, Cambridge University Press
ISBN:0521781760

[DotNET-Home]
Microsoft Corporation (2004)
Microsoft .NET Homepage
http://www.microsoft.com/net/

[DOM]
W3C (2004)
Document Object Model (DOM)
http://www.w3.org/DOM/

[ETTK-Home]
IBM alphaWorks (08/03/2004)
Emerging Technologies Toolkit Overview
http://www.alphaworks.ibm.com/tech/ettk

[ETTK-Doc]
IBM Corporation (2004)
Emerging Technologies Toolkit Documentation
http://awWebx04.alphaworks.ibm.com:80/wstk/common/wstkdoc/README.htm

[ETTK-WS]
IBM (2004)
Implementing ETTK Web Services with WebSphere Studio Application Developer 5.1.1
http://dwdemos.dfw.ibm.com/wstk/common/wstkdoc/tutorial/pages/tutorial.html

[Grid-Fundamentals]
Viktors Berstis (11/11/2002)
Fundamentals of Grid Computing, IBM Redbooks Paper
http://www.redbooks.ibm.com/redpapers/pdfs/redp3613.pdf

[Globus-Home]
Globus Alliance (07/23/2004)
The Globus Alliance Homepage, University of Chicago
http://www.globus.org/

[GRID-Blueprint]
I. Foster, C. Kesselman (1999)
The GRID - Blueprint fo a New Computing Infrastructure, Morgan Kaufmann Publishers, Inc.
ISBN:1-55860-475-8

[GRID-Computing]
F. Berman, G.C.Fox, A.J.G. Hey (2003)
Grid Computing - Making the Global Infrastructure a Reality, Wiley
ISBN:0-470-85319-0

[GT3-Core]
T. Sandholm, J. Gawor (07/02/03)
Globus Toolkit 3 Core - A Grid Service Container
http://www-unix.globus.org/toolkit/3.0/ogsa/docs/gt3_core.pdf

[GT3-Tutorial]
B. Sotomayor (05/11/04)
The Globus Toolkit 3 Programmer's Tutorial
http://www.casa-sotomayor.net/gt3-tutorial/

[GT4-Design]
J. Gawor, S. Meder (05/24/04)
GT4 WS Java Core Design
http://www-unix.globus.org/toolkit/docs/development/wsrf/3.9.1/WSRFDesign.pdf

[HTTP-COMP]
S. Radhakrishnan (07/22/03)
Speed Web delivery with HTTP compression
http://www-106.ibm.com/developerworks/web/library/wa-httpcomp/

[IIS-Home]
Microsoft Corporation (2004)
Internet Information Services Home page
http://www.microsoft.com/windowsserver2003/iis/default.mspx

[Java-Servlet]
J. Hunter, W. Crawford (04/2001)
Java - Servlet Programming, O'Reilly 2nd Edition
ISBN:0-596-00040-5

[Java-XML]
B. McLaughlin (2002)
Java & XML, O'Reilly
ISBN:3-89721-296-X

[JAX-RPC]
Sun Microsystems, Inc. (1994-2004)
Java API for XML-Based RPC (JAX-RPC) Homepage,
http://java.sun.com/xml/jaxrpc/index.jsp

[JINI-Spec]
Sun Microsystems, Inc. (1997-2003)
Jini Architecture Specification, Sun Microsystems, Inc
http://www.jini.org/nonav/standards/davis/doc/specs/html/jini-spec.html

[JSDL-Wg-Home]
JSDL-WG (07/13/2004)
Job Submission Description Language Working Group Homepage, GGF JSDL Working Group
http://www.epcc.ed.ac.uk/~ali/WORK/GGF/JSDL-WG

[JSDL-Spec]
F. Brisard, A. Ly(07/05/2004)
Job Submission Description Language (JSDL) Specification, Version 0.4.4, GGF JSDL Working Group
http://forge.gridforum.org/projects/jsdl-wg/document/draft-ggf-jsdl-spec/en/8

[Middleware]
Arno Puder, Kay Römer (2001)
Middleware, dpunkt publishing company
ISBN:3-932588-03-7

[MONO]
Novell, Inc (2004)
The Mono Project Home Page
http://www.mono-project.com/about/index.html

[MSDN]
Microsoft Corporation (2004)
The Microsoft Developer Network (MSDN) Home Page
http://msdn.microsoft.com/

[NICE-Home]
University of Illinois at Chicago
The Narrative Immersive Constructionist/Collaborative Environments project
http://www.evl.uic.edu/tile/NICE/NICE/intro.html

[OASIS-Home]
OASIS (2004)
OASIS Home page, OASIS
http://www.oasis-open.org/

[OASIS-WSRF]
OASIS (2004)
OASIS Web Services Resource Framework Technical Committee, OASIS
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

[OpenSource]
W. Andreas Klimke (05/2001)
Web Application Development Using Open Source and Java Technologies
http://www.andreasklimke.de/Webdev/index.html

[WS-Refactor]
K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling, S. Tuecke (03/05/2004)
From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring & Evolution, Version 1.1
http://www-106.ibm.com/developerworks/library/ws-resource/ogsi_to_wsrf_1.0.pdf

[OGSA-DAI]
(2004)
Open Grid Services Architecture Data Access and Integration OGSA-DAI Home Page
http://www.ogsadai.org.uk/

[OGSA-DQP]
(2004)
Service Based Distributed Query Processor
http://www.ogsadai.org.uk/dqp/

[OGSI-Spec]
S. Tuecke, K. Czajkowski, I. Foster, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling, P. Vanderbilt (04/05/2003)
Open Grid Services Infrastructure (OGSI) Version 1.0, Draft
www.gridforum.org/ogsi-wg/drafts/draft-ggf-ogsi-gridservice-29_2003-04-05.pdf

[OGSI.NET-Home]
University of Virginia Grid Computing Group (2004)
Open Grid Services Infrastructure .NET Homepage
http://www.cs.virginia.edu/~gsw2c/ogsi.net.html

[OWL-XML-Syntax]
M. Hori, J. Euzenat, P.F. Patel-Schneider (06/11/2003)
OWL Web Ontology Language XML Presentation Syntax, W3C Note
http://www.w3.org/TR/2003/NOTE-owl-xmlsyntax-20030611/owl-xmlsyntax.html

[OWL-Overview]
D.L. McGuiness, F. van Harmelen (2004)
OWL Web Ontology Language Overview, W3C Working Group Note
http://www.w3.org/TR/2004/REC-owl-features-20040210/

[OWL-Ref]
S. Bechhofer, F. van Harmelen, J. Hender, I. Horrocks, D.L. McGuiness, P.F. Patel-Schneider, L. A. Stein (2004)
OWL Web Ontology Language Reference, W3C Recommendation
http://www.w3.org/TR/owl-ref/

[Pab-Report]
M. Fidler, W. Klimala, V. Sander(07/12/2004)
Path Allocation in Backbone Networks: Project Report
http://www.pab.rwth-aachen.de/images/PABFinalReport.pdf

[Physiology]
I. Foster, C. Kesselman, J.M. Nick, S. Tuecke (06/22/2002)
The Physiology of the Grid - An Open Grid Services Architecture for Distributed Systems Integration
http://www.globus.org/research/papers/ogsa.pdf

[RDF]
G. Klyne, J.J. Caroll (2002)
Resource Description Framework: Concepts and Abstract Syntax, W3C Working Draft
http://www.w3.org/TR/2002/WD-rdf-concepts-20021108/

[RFC1738]
T. Berners-Lee, L. Masinter, M. McCahill (12/1994)
RFC 1738 - Uniform Resource Locators (URL), Request for Comments
http://www.ietf.org/rfc/rfc1738.txt

[RFC2396]
T. Berners-Lee, R. Fielding, L. Masinter (08/1998)
RFC 2396 - Uniform Resource Identifiers (URI), Request for Comments
http://www.ietf.org/rfc/rfc2396.txt

[RFC2616]
R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee (06/1999)
Hypertext Transfer Protocol HTTP/1.1, Request for Comments
http://www.ietf.org/rfc/rfc2616.txt

[Servlet-SPEC]
D. Coward, Y. Yoshida (11/24/2003)
Java Servlet Specification Version 2.4, Java Community Press
http://www.jcp.org/aboutJava/communityprocess/final/jsr154/

[SOA-Definition]
Barry & Associates, Inc. (2000-2004)
Service-oriented architecture (SOA) definition, Barry & Associates, Inc.
http://www.service-architecture.com/Web-services/articles/service-oriented_architecture_soa_definition.html

[SOAP]
M. Gudgin, M.Hadley, N. Mendelsohn, J. Moreau, H.Nielsen (06/24/2003)
SOAP Version 1.2 Part 1: Messaging Framework, W3C Recommendation
http://www.w3.org/TR/soap12-part1/

[SunWAR]
Sun Microsystems (08/07/2002)
Web Application Archives, Tutorial
http://java.sun.com/Webservices/docs/1.0/tutorial/doc/WebApp3.html

[Tao]
S. Burbeck (10/2000)
The Tao of e-business, IBM Software Group
ftp://www6.software.ibm.com/software/developer/library/ws-tao.pdf

[Websters]
P. B. Gove, Merriam-Webster Editorial staff (1993)
Webster's Third New International Dictionary, Merriam-Webster, Inc.
ISBN:3-8290-5292-8

[WebSphere-AppExpress]
IBM Corporation (2004)
IBM WebSphere Application Server Express - Version 5.1 - Spec Sheet, IBM
http://www-306.ibm.com/software/Webservers/appserv/express/WASexpress51-spec.pdf

[WebSphere-Home]
IBM Corporation (2004)
IBM WebSphere software platform, IBM
www-306.ibm.com/software/info/Websphere

[Wikipedia]
Wikimedia Foundation (2004)
Wikipedia - The Free Encyclopedia, Wikimedia Foundation
http://en.wikipedia.org/wiki/Main_Page

[WS-Adressing]
A. Bosworth, D. Box, E. Christensen, F. Curbera, D. Ferguson, J. Frey, C. Kaler, D. Langworthy, F. Leymann, B. Lovering, S. Lucco, S. Millet, N. Mukhi, M. Nottingham, D. Orchard, J. Shewchuk, T. Storey, S. Weerawarana (03/2004)
Web Services Adressing
http://www-106.ibm.com/developerworks/library/ws-resource/ws-resourceProperties.pdf

[WS-Agreement]
A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, M. Xu (05/19/2004)
Web Services Agreement Specification, Version 1.1, Draft 20
`https://forge.gridforum.org/projects/graap-wg/document/WS-AgreementSpecification/en/4`

[WS-AgreementNegotiation]
A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, M. Xu (08/01/2004)
Web Services Agreement Negotiation Specification, Version 1.0, Working draft
`https://forge.gridforum.org/projects/graap-wg/document/WS-AgreementNegotiationSpecificationDraft.doc/en/1`

[WS-Arch]
D. Booth, H. Haas, F. McCabe, E. Newcomer, I. M. Champion, C. Ferris, D. Orchard (2004)
Web Services Architecture, W3C Working Group Note
`http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/`

[WS-Corba]
Irmen de Jong (and others) (04/27/2002)
Web Services/SOAP and CORBA, Object Management Group, Inc. Whitepaper
`http://www.omg.org/news/whitepapers/CORBA_vs_SOAP1.pdf`

[WSDL 1.1]
E. Christensen, F. Curbera, G. Meredith, S. Weerawarana (03/15/2001)
Web Services Description Language (WSDL) 1.1, W3C Note
`http://www.w3.org/TR/wsdl`

[WSDL 2.0]
R. Chinnici, M. Gudgin, J.J. Moreau, J. Schlimmer, S. Weerawarana (03/26/2004)
Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, W3C Working Draft
`http://www.w3.org/TR/wsdl20/`

[WSFL]
F. Leymann (05/2001)
Web Services Flow Language (WSFL), Version 1.0, IBM
`http://www-306.ibm.com/software/solutions/Webservices/pdf/WSFL.pdf`

[WS-Inspection]
K. Ballinger, P. Brittenham, A. Malhotra, W. A. Nagy, S. Pharies (11/2001)
Web Services Inspection Language (WS-Inspection), Version 1.0, International Business Machines Corporation, Microsoft
`http://www-106.ibm.com/developerworks/Webservices/library/ws-wsilspec.html`

[WSLA-Home]
IBM Corporation (2001-2003)
Web Service Level Agreements Project, IBM Corporation
`http://www.research.ibm.com/wsla/`

[WSLA-Report]
A. Keller, H. Ludwig (05/22/2002)
The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services, IBM Research Report
http://domino.watson.ibm.com/library/cyberdig.nsf/papers/CDEDB79080F59EE285256C5900654839/$File/RC22456.pdf

[WSLA-Spec]
H. Ludwig, A. Keller, A. Dan, R. P. King, R. Franck (01/28/2003)
Web Service Level Agreement Language Specification, Version 1.0, IBM Corporation
http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf

[WS-BaseFaults]
S. Tuecke, K. Czajkowski, J. Frey, I. Foster, S. Graham, T. Maguire, I. Sedukhin, D. Snelling, W. Vambenepe (03/31/2004)
Web Services Base Faults, Version 1.0
http://www-106.ibm.com/developerworks/library/ws-resource/ws-basefaults.pdf

[WS-Base Notification]
S. Graham, P. Niblett, D. Chappell, A. Lewis, N. Nagaratnam, J. Parikh, S. Patil, S. Samdarshi, I. Sedukhin, D. Snelling, S. Tuecke, W. Vambenepe, B. Weihl (03/05/2004)
Web Services Base Notification (WS-Base Notification), Version 1.0
ftp://www6.software.ibm.com/software/developer/library/ws-notification/WS-BaseN.pdf

[WS-Brokered Notification]
S. Graham, P. Niblett, D. Chappell, A. Lewis, N. Nagaratnam, J. Parikh, S. Patil, S. Samdarshi, I. Sedukhin, D. Snelling, S. Tuecke, W. Vambenepe, B. Weihl (03/05/2004)
Web Services Brokered Notification (WS-Brokered Notification), Version 1.0
ftp://www6.software.ibm.com/software/developer/library/ws-notification/WS-BrokeredN.pdf

[WS-DEVETTK]
D. Davis (05/02/2003)
Developing using the ETTK, Part 1, IBM Web service basics
http://www-106.ibm.com/developerworks/Webservices/library/ws-devettk/

[WS-DEVETTK2]
D. Davis (05/13/2003)
Developing using the ETTK, Part 2, IBM Web service basics
http://www-106.ibm.com/developerworks/Webservices/library/ws-devettk2/

[WSE-Home]
Microsoft Corporation (2004)
Web Services Enhancements (WSE) Homepage
http://msdn.microsoft.com/Webservices/building/wse/default.aspx

[WS-MetaDataExchange]
K. Ballinger, D. Box, F. Curbera, S. Graham, C. K. Liu, B. Lovering, A. Nadalin, M. Nottingham, D. Orchard, C. v. Riegen, J. Schlimmer, J. Shewchuk, G. Truty, S. Weerawarana (02/2004)
Web Services Metadata Exchange (WSMetadataExchange), Version 1.0, IBM Corporation
http://xml.coverpages.org/WS-MetadataExchange.pdf

[WS-ModelingResources]
I. Foster, J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, I. Sedukhin, D. Snelling, T. Storey, W. Vambenepe, S. Weerawarana (03/05/2004)
Modeling Stateful Resources with Web Services
http://www-106.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf

[WS-PubSub]
S. Graham, P. Niblett, D. Chappell, A. Lewis, N. Nagaratnam, J. Parikh, S. Patil, S. Samdarshi, I. Sedukhin (03/05/2004)
Publish-Subsribe Notification for Web Services, Version 1.0
http://www-106.ibm.com/developerworks/library/ws-pubsub/WS-PubSub.pdf

[WS-QOS]
H. Ludwig (12/13/2003)
Web Services QoS: External SLAs and Internal Policies, IBM Keynote
http://www.research.ibm.com/people/h/hludwig/publications/WQWKeynoteDec2003.pdf

[WSRF]
K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, I.Sedukhin, D. Snelling, S. Tuecke, W. Vambenepe (03/05/2004)
The Web Services Resource Framework, Version 1.0
http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrfpaper.html

[WSRF.Lite-Home]
The University of Manchester - Manchester Computing (07/20/2004)
WSRF::Lite Homepage
http://www.sve.man.ac.uk/Research/AtoZ/ILCT

[WSRF.Lite-Introduction]
The University of Manchester - Manchester Computing (07/20/2004)
WSRF - Building WS-Resources with WSRF::Lite
http://vermont.mvc.mcc.ac.uk/WSRF-Lite.pdf

[WSRF.Lite-Workshop]
The University of Manchester - Manchester Computing (07/15/2004)
OGSI::Lite and WSRF::Lite - Creating Grid Services in Perl
http://vermont.mvc.mcc.ac.uk/RealityGridWorkshop2.pdf

[WSRF.NET]
G. Wasson, N. Beekwilder, M. Morgan, M. Humphrey (2004)
Web Services Resource Framework on .NET
http://www.cs.virginia.edu/~gsw2c/WSRFdotNet/wsrfon.net.pdf

[WSRF.NET-Home]
University of Virginia Grid Computing Group (2004)
Web Services Resource Framework .NET Homepage
http://www.cs.virginia.edu/~gsw2c/wsrf.net.html

[WSRF.NET-Reference]
G. Wasson (06/23/2004)
WSRF.NET Programmer's Reference
http://www.cs.virginia.edu/~gsw2c/WSRFdotNet/WSRFdotNet_programmers_reference.pdf

[WSRF-Interop]
M. Humphrey (06/08/2004)
WSRF Interop Demo, University of Virginia
http://www.gridforum.org/Meetings/GGF11/presentations/WSRF_Interop.ppt

[WS-ResourceLifetime]
J. Frey, S. Graham, K. Czajkowski, D. Ferguson, I. Foster, F. Leymann, T. Maguire, N. Nagaratnam, M. Nally, T. Storey, S. Tuecke, W. Vambenepe, S. Weerawarana (01/20/2004)
Web Services Resource Lifetime
http://www-106.ibm.com/developerworks/library/ws-resource/ws-resourceLifetime.pdf

[WS-ResourceProperties]
S. Graham, K. Czajkowski, D. Ferguson, I. Foster, J. Frey, F. Leymann, T. Maguire, N. Nagaratnam, M. Nally, T. Storey, S. Tuecke, W. Vambenepe, S. Weerawarana (01/20/2004)
Web Services Resource Properties
http://www-106.ibm.com/developerworks/library/ws-resource/ws-resourceProperties.pdf

[WS-Security]
B. Atkinson, G.Della-Libera, S. Hada, M. Hondo, P. Hallam-Baker, J. Klein, B. LaMacchia, P. Leach, J. Manferdelli, H. Maruyama, A. Nadalin, N. Nagaratnam, H. Prafullchandra, J. Shewchuk, D. Simon (04/05/2002)
Web Services Security, Version 1.0
ftp://www6.software.ibm.com/software/developer/library/ws-secure.pdf

[WS-ServiceGroup]
S. Graham, T. Maguire, J. Frey, N. Nagaratnam, I. Sedukhin, D. Snelling, K. Czajkowski, S. Tuecke, W. Vambenepe (03/31/2004)
Web Services Service Group, Version 1.0
http://www-106.ibm.com/developerworks/library/ws-resource/ws-servicegroup.pdf

[WS-SpeedStart]
IBM developerWorks Live (05/06/2004)
Speed-start Web Services - Technical Briefing Documentation
http://ibm.com/developerworks/offers/techbriefings/presentations/Webservices.html

[WS-Topics]
S. Graham, P. Niblett, D. Chappell, A. Lewis, N. Nagaratnam, J. Parikh, S. Patil, S. Samdarshi, I. Sedukhin, D. Snelling, S. Tuecke, W. Vambenepe, B. Weihl (03/05/2004)
Web Services Topics (WS-Topics), Version 1.0
ftp://www6.software.ibm.com/software/developer/library/ws-notification/WS-Topics.pdf

[WSTK]
J. Feller (2000)
IBM Web Services Toolkit - A showcase for emerging Web Services technologies, IBM
http://www-306.ibm.com/software/solutions/Webservices/wstk-info.html

[XML-KMS]
W. Ford, P. Hallam-Baker, B. Fox, B. Dillaway, B. LaMacchia, J. Epstein, J.Lapp (03/30/2001)
XML Key Management Specification, W3C Note
http://www.w3.org/TR/xkms/

[XML-Definitive]
P. Walmsley(2002)
Definitive XML Schema, Prentice Hall
ISBN:0-13-065567-8

[XML-NS]
W3C (01/14/99)
Namespaces in XML, W3C
http://www.w3.org/TR/REC-xml-names/

[XML-Prof]
V. Chopra, Z. Zoran, G.Damschen, C. Dix, P. Cauldwell, R. Chawla, K. Saunders, G. Olander, F. Norton, T. Hong, U. Ogbuji, M. Richman (9/2001)
Professional XML Web Services
ISBN:1861005091

[XML-Schema]
W3C (2000-2003)
XML Schema Definition Language, W3C
http://www.w3.org/XML/Schema

[XPath]
J. Clark, S. DeRose (11/16/99)
XML Path Language (XPath), Version 1.0 W3C Recommendation
http://www.w3.org/TR/xpath

Appendix A - Network Reservation Scenario

Agreement-Offer

The following agreement offer includes service description terms and guarantee terms.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsag:AgreementOffer
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wsag="http://www.ggf.org/namespaces/ws-agreement"
    xmlns:job="http://www.gridforum.org/namespaces/job"
    xsi:schemaLocation="http://www.ggf.org/namespaces/ws-agreement
    agreement_types.xsd http://www.gridforum.org/
        namespaces/job job_terms.xsd">
  <wsag:Name>TunnelOffer1</wsag:Name>
  <wsag:Context>
    <wsag:AgreementInitiator>EPR Unknown
    </wsag:AgreementInititator>
    <wsag:AgreementProvider>FactoryEPR</wsag:AgreementProvicer>
    <wsag:AgreementInitiatorIsServiceConsumer>true
    </wsag:AgreementInitiatorIsServiceConsumer>
    <wsag:TerminationTime>2004-07-31T20:00:00</wsag:TerminationTime>
    <wsag:TemplateName>AgreementTemplate1</wsag:TemplateName>
    <wsag:RelatedAgreements/>
  </wsag:Context>
  <wsag:Terms>
    <wsag:All>
      <wsag:ServiceDescriptionTerm wsag:Name="StartTime"
          wsag:ServiceName="NetworkAgreement">
        <job:StartTime>2004-07-01T08:00:00</job:StartTime>
      </wsag:ServiceDescriptionTerm>
      ...
```

```
...
<wsag:ServiceDescriptionTerm wsag:Name="EndTime"
    wsag:ServiceName="NetworkAgreement">
  <job:EndTime>2004-07-31T20:00:00</job:EndTime>
 </wsag:ServiceDescriptionTerm>
 <wsag:ServiceDescriptionTerm wsag:Name="IPADDRESSA"
     wsag:ServiceName="NetworkAgreement">
   <job:IPAddr>130.68.90.9</job:IPAddr>
 </wsag:ServiceDescriptionTerm>
 <wsag:ServiceDescriptionTerm wsag:Name="IPADDRESSB"
     wsag:ServiceName="NetworkAgreement">
   <job:IPAddr>130.68.90.9</job:IPAddr>
 </wsag:ServiceDescriptionTerm>
 <wsag:ServiceDescriptionTerm wsag:Name="PORTA"
     wsag:ServiceName="NetworkAgreement">
   <job:Port>5000</job:Port>
 </wsag:ServiceDescriptionTerm>
 <wsag:ServiceDescriptionTerm wsag:Name="PORTB"
     wsag:ServiceName="NetworkAgreement">
   <job:Port>3000</job:Port>
 </wsag:ServiceDescriptionTerm>
 <wsag:ServiceDescriptionTerm wsag:Name="BANDWIDTH"
     wsag:ServiceName="NetworkAgreement">
   <job:Bandwidth>50MBit/s</job:Bandwidth>
 </wsag:ServiceDescriptionTerm>
 <wsag:ServiceDescriptionTerm wsag:Name="DELAY"
     wsag:ServiceName="NetworkAgreement">
   <job:Delay>10ms - 50ms</job:Delay>
  </wsag:ServiceDescriptionTerm>
  <wsag:ServiceReference/>
  <wsag:ServiceProperties wsag:ServiceName="ComputeJob1">
    <wsag:VariableSet>
      <wsag:Variable wsag:Name="StartTimeProperty">
        <wsag:Location>
          /wsag:AgreementOffer/wsag:Terms/wsag:All/
               wsag:ServiceDescriptionTerm[@wsag:Name='StartTime']
        <wsag:Location/>
      </wsag:Variable>
      <wsag:Variable wsag:Name="EndTimeProperty">
        <wsag:Location>
           /wsag:AgreementOffer/wsag:Terms/wsag:All/
               wsag:ServiceDescriptionTerm[@wsag:Name='EndTime']
        <wsag:Location/>
      </wsag:Variable>
    </wsag:VariableSet>
  </wsag:ServiceProperties>
  ...
```

```
          ...
          <wsag:GuaranteeTerm wsag:name="TunnelLifetime">
             <wsag:ServiceScope/>
                <wsag:ServiceName>Agreement<wsag:ServiceName/>
             </wsag:ServiceScope>
             <wsag:Variables>
                <wsag:Variable name="start">
                   <wsag:Location>
                      /wsag:AgreementOffer//wsag:Terms/wsag:All/
                       wsag:ServiceDescriptionTerm[@wsag:Name='StartTime']
                   </wsag:Location>
                 </wsag:Variable/>
                 <wsag:Variable name="end">
                    <wsag:Location>/wsag:AgreementOffer//
                       wsag:Terms/wsag:All/
                       wsag:ServiceDescriptionTerm[@wsag:Name='EndTime']
                       </wsag:Location>
                    </wsag:Variable/>
                  <wsag:Variables/>
             <wsag:QualifyingCondition/>
             <wsag:ServiceLevelObjective>NO_INTERUPTION_OF_SERVICE
             <wsag:ServiceLevelObjective/>
             <wsag:BusinessValueList>
                <wsag:Importance/>
                <wsag:Penalty>
                   <wsag:AssessmentInterval>
                      <wsag:Count>1</wsag:Count>
                   </wsag:AssessmentInterval>
                   <wsag:ValueExpression>2</wsag:ValueExpression>
                </wsag:Penalty>
                <wsag:Reward/>
             <wsag:BusinessValueList/>
          </wsag:GuaranteeTerm>
       </wsag:All>
    </wsag:Terms>
</wsag:AgreementOffer>
```

Appendix B - NetworkAgreement Service Description

agreement_port_type.wsdl

The listing of this WSDL file can be found inside [**WS-Agreement**].

NetworkAgreement.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:mrpxsd="http://www.fz-juelich.de/
        namespaces/wsag/Agreement-xsd"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://www.fz-juelich.de/namespaces/wsag/
       NetworkAgreement"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/
        2003/03/addressing"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsrl="http://www.ibm.com/xmlns/stdwip/
        Web-services/WS-ResourceLifetime"
    xmlns:wsrp="http://www.ibm.com/xmlns/stdwip/
        Web-services/WS-ResourceProperties"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:wsag="http://www.ggf.org/namespaces/ws-agreement"
    name="Agreement"
    targetNamespace="http://www.fz-juelich.de/
      namespaces/wsag/NetworkAgreement">

   ...
```

```
...

<import location="http://localhost:4400/wstk/wsag/
      agreement_port_type.wsdl"
    namespace="http://www.ggf.org/
    namespaces/ws-agreement"/>

<binding name="NetworkAgreementBinding"
    type="wsag:Agreement">

 <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>

 <wsdl:operation name="GetResourceProperty">
    <soap:operation soapAction="http://www.ibm.com/
          xmlns/stdwip/Web-services/WS-ResourceProperties"/>
    <wsdl:input>
       <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
       <soap:body use="literal"/>
    </wsdl:output>
 </wsdl:operation>

 <wsdl:operation name="Terminate">
    <soap:operation soapAction="http://www.ggf.org/
         namespaces/ws-agreement/Terminate"/>
    <wsdl:input>
       <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
       <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>

  <!-- Domain Specific extensions -->
  <wsdl:operation name="GetMultipleResourceProperties">
    <soap:operation soapAction="http://www.ibm.com/xmlns/
        stdwip/Web-services/WS-ResourceProperties"/>
    <wsdl:input>
       <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
       <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  ...
```

```
    ...

    <wsdl:operation name="QueryResourceProperties">
       <soap:operation soapAction="http://www.ibm.com/xmlns/
             stdwip/Web-services/WS-ResourceProperties"/>
       <wsdl:input>
          <soap:body use="literal"/>
       </wsdl:input>
       <wsdl:output>
          <soap:body use="literal"/>
       </wsdl:output>
     </wsdl:operation>
  </binding>
</definitions>
```

NetworkAgreement-Impl.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
   name="NetworkAgreement-impl"
   targetNamespace="http://www.fz-juelich.de/
       namespaces/wsag/NetworkAgreement"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:interface="http://www.fz-juelich.de/
       namespaces/wsag/NetworkAgreement"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <import location="http://localhost:4400/wstk/wsag/
         NetworkAgreement.wsdl"
       namespace="http://www.fz-juelich.de/namespaces/
         wsag/NetworkAgreement"/>

   <service name="NetworkAgreement">
      <documentation>Simple Agreement Service to illustrate
      GGF GRAAP agreement negotiations.</documentation>
      <port binding="interface:NetworkAgreementBinding"
        name="AgreementPort">
        <soap:address location="http://localhost:4400/wstk/
           services/NetworkAgreement"/>
      </port>
  </service>
</definitions>
```

Appendix C - JuelichAgreementFactory Service Description

agreement_factory_port_type.wsdl

The listing of this WSDL file can be found inside [**WS-Agreement**].

JuelichAgreementFactory.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
   name="JuelichAgreementFactory"
   targetNamespace="http://www.fz-juelich.de/
     namespaces/wsag/JuelichAgreementFactory"
  xmlns:tns="http://www.fz-juelich.de/namespaces/
     wsag/JuelichAgreementFactory"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsag="http://www.ggf.org/namespaces/ws-agreement">

  <import location="http://localhost:4400/wstk/wsag/
       agreement_factory_port_type.wsdl"
     namespace="http://www.ggf.org/namespaces/ws-agreement"/>

  <binding name="JuelichAgreementFactoryBinding"
     type="wsag:AgreementFactory">

   <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>

   <wsdl:operation name="createAgreement">
     <soap:operation soapAction="http://www.ggf.org/
        namespaces/ws-agreement/createAgreement"/>
     <wsdl:input>
       <soap:body use="literal"/>
     </wsdl:input>
     <wsdl:output>
       <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    ...
```

```
    ...

    <wsdl:operation name="GetResourceProperty">
      <soap:operation soapAction="http://www.ibm.com/xmlns/stdwip/
         Web-services/WS-ResourceProperties/GetResourceProperty"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>

    <wsdl:operation name="GetMultipleResourceProperties">
      <soap:operation soapAction="http://www.ibm.com/xmlns/
          stdwip/Web-services/WS-ResourceProperties"/>
      <wsdl:input>
         <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
         <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>

    <wsdl:operation name="QueryResourceProperties">
       <soap:operation soapAction="http://www.ibm.com/xmlns/
             stdwip/Web-services/WS-ResourceProperties"/>
       <wsdl:input>
          <soap:body use="literal"/>
       </wsdl:input>
       <wsdl:output>
          <soap:body use="literal"/>
       </wsdl:output>
     </wsdl:operation>

  </binding>

</definitions>
```

JuelichAgreementFactory-Impl.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
   name="JuelichAgreementFactory-impl"
   targetNamespace="http://www.fz-juelich.de/namespaces/
      wsag/JuelichAgreementFactory"
  xmlns:tns="http://www.fz-juelich.de/namespaces/wsag/
      JuelichAgreementFactory"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">

  <import
    location="http://localhost:4400/wstk/wsag/
       JuelichAgreementFactory.wsdl"
    namespace="http://www.fz-juelich.de/namespaces/wsag/
       JuelichAgreementFactory"/>

  <service name="JuelichAgreementFactory">
    <documentation>
       Simple AgreementFactory Service to illustrate
       GGF agreement negotiations.
    </documentation>

    <port binding="tns:JuelichAgreementFactoryBinding"
          name="AgreementFactoryPort">
      <soap:address location="http://localhost:4400/wstk/
          services/JuelichAgreementFactory"/>
    </port>

  </service>
</definitions>
```

Forschungszentrum Jülich
in der Helmholtz-Gemeinschaft

Jül-4154
Dezember 2004
ISSN 0944-2952