universität wien

DISSERTATION

Titel der Dissertation

A service-oriented Grid environment with on-demand QoS support

Verfasser

Mag. Gerhard Engelbrecht

angestrebter akademischer Grad

Doktor der technischen Wissenschaften (Dr.tech.)

Wien, 2009

Studienkennzahl lt. Studienblatt:	A 786 175
Dissertationsgebiet lt. Studienblatt:	Wirtschaftsinformatik
Betreuer:	ao. Univ.Prof. Dipl.-Ing. Dr. Siegfried Benkner

A service-oriented Grid environment with on-demand QoS support

Gerhard Engelbrecht

Abstract

Grid computing emerged as a vision for a new computing infrastructure that aims to make computing resources available as easily as electric power through the power grid. Enabling seamless access to globally distributed IT resources allows dispersed users to tackle large-scale problems in science and engineering in unprecedented ways.

The rapid development of Grid computing also encouraged standardization, which led to the adoption of a service-oriented paradigm and an increasing use of commercial Web services technologies. Along these lines, service-level agreements and Quality of Service are essential characteristics of the Grid and specifically mandatory for Grid-enabling complex applications from certain domains such as the health sector.

This PhD thesis aims to contribute to the development of Grid technologies by proposing a Grid environment with support for Quality of Service. The proposed environment comprises a secure service-oriented Grid infrastructure based on standard Web services technologies which enables the on-demand provision of native HPC applications as Grid services in an automated way and subject to user-defined QoS constraints.

The Grid environment adopts a business-oriented approach and supports a client-driven dynamic negotiation of service-level agreements on a case-by-case basis. Although the design of the QoS support is generic, the implementation emphasizes the specific requirements of compute-intensive and time-critical parallel applications, which necessitate on-demand QoS guarantees such as execution time limits and price constraints. Therefore, the QoS infrastructure relies on advance resource reservation, application-specific resource capacity estimation, and resource pricing. An experimental evaluation demonstrates the capabilities and rational behavior of the QoS infrastructure.

The presented Grid infrastructure and in particular the QoS support has been successfully applied and demonstrated in EU projects for various applications from the medical and bio-medical domains. The EU projects GEMSS and Aneurist are concerned with advanced e-health applications and globally distributed data sources, which are virtualized by Grid services. Using Grid technology as enabling technology in the health domain allows medical practitioners and researchers to utilize Grid services in their clinical environment which ultimately results in improved healthcare.

Eine service-orientierte Grid Umgebung mit abrufbarer QoS Unterstüztung

Gerhard Engelbrecht

Kurzfassung

Grid Computing entstand aus der Vision für eine neuartige Recheninfrastruktur, welche darauf abzielt, Rechenkapazität so einfach wie Elektrizität im Stromnetz (power grid) verfügbar zu machen. Der entsprechende Zugriff auf global verteilte Rechenressourcen versetzt Forscher rund um den Globus in die Lage, neuartige Herausforderungen aus Wissenschaft und Technik in beispiellosem Ausmaß in Angriff zu nehmen.

Die rasanten Entwicklungen im Grid Computing begünstigten auch Standardisierungsprozesse in Richtung Harmonisierung durch Service-orientierte Architekturen und die Anwendung kommerzieller Web Services Technologien. In diesem Kontext ist auch die Sicherung von Qualität bzw. entsprechende Vereinbarungen über die Qualität eines Services (QoS) wichtig, da diese vor allem für komplexe Anwendungen aus sensitiven Bereichen, wie der Medizin, unumgänglich sind.

Diese Dissertation versucht zur Entwicklung im Grid Computing beizutragen, indem eine Grid Umgebung mit Unterstützung für QoS vorgestellt wird. Die vorgeschlagene Grid Umgebung beinhaltet eine sichere Service-orientierte Infrastruktur, welche auf Web Services Technologien basiert, sowie bedarfsorientiert und automatisiert HPC Anwendungen als Grid Services bereitstellen kann.

Die Grid Umgebung zielt auf eine kommerzielle Nutzung ab und unterstützt ein durch den Benutzer initiiertes, fallweises und dynamisches Verhandeln von Serviceverträgen (SLAs). Das Design der QoS Unterstützung ist generisch, jedoch berücksichtigt die Implementierung besonders die Anforderungen von rechenintensiven und zeitkritischen parallelen Anwendungen, bzw. Garantien für deren Ausführungszeit und Preis. Daher ist die QoS Unterstützung auf Reservierung, anwendungsspezifische Abschätzung und Preisfestsetzung von Ressourcen angewiesen. Eine entsprechende Evaluation demonstriert die Möglichkeiten und das rationale Verhalten der QoS Infrastruktur.

Die Grid Infrastruktur und insbesondere die QoS Unterstützung wurde in Forschungs- und Entwicklungsprojekten der EU eingesetzt, welche verschiedene Anwendungen aus dem medizinischen und bio-medizinischen Bereich als Services zur Verfügung stellen. Die EU Projekte GEMSS und Aneurist befassen sich mit fortschrittlichen HPC Anwendungen und global verteilten Daten aus dem Gesundheitsbereich, welche durch Virtualisierungstechniken als Services angeboten werden. Die Benutzung von Gridtechnologie als Basistechnologie im Gesundheitswesen ermöglicht Forschern und Ärzten die Nutzung von Grid Services in deren Arbeitsumfeld, welche letzten Endes zu einer Verbesserung der medizinischen Versorgung führt.

This PhD thesis is dedicated to my mother the late Christine Engelbrecht.

Contents

List of Figures

List of Tables

Listings

Acknowledgements

First of all, I would like to express my appreciation to Professor Siegfried Benkner for being my supervisor over the past years. He constantly stretched my mind by pushing me forward to learn about diverse domains and view similar topics from different perspectives. Furthermore, I am very thankful for the opportunity to bring in my experience in multiple international research projects to collaborate with scientists on the leading edge of the e-science community. Above all that he helped me to stay focused on my overall research direction and finally kept me motivated to accomplish my thesis.

I am also very grateful to my second adviser Dr. Erwin Laure. He provided me new insights to one of the major research efforts of the Grid community, the EGEE project, and extended my view of Grid computing with a number of controversial discussions.

I would also like to thank my current and former colleagues at the Institute of Scientific Computing, University of Vienna, especially Rainer Schmidt and Ivona Brandic. I enjoyed being part of the team and all the fruitful discussions we had.

I would like to express many thanks to Tina Csaicsich, who provided her superior English skills as prospective interpreter to proofread this thesis.

Finally, I would like to express deeply my gratefulness for my brother Christian. We both had a hard time upon the sudden death of our mother, but he helped me to keep on track with my scientific work, while on the other hand he also motivated me to have a go at new activities on leisure time like climbing or diving.

Chapter 1

Introduction

This chapter presents a high-level survey of this thesis. It comprises a brief introduction to Grid computing and Quality of Service (QoS), derives therefrom the motivation of this thesis and subsequently proposes an approach to address the situation. Finally, the organization of this thesis is being outlined.

1.1 Grid computing

Science and engineering are facing an increasing complexity of their research questions at hand. Grid computing is intended to support scientists to encounter these challenges. The increasing scale of scientific applications, the growing number of dispersed high-end computing facilities and the enormous quantities of information available in globally distributed data sources constitute the motivating frame of Grid computing. The Grid is projected to fill the existing gap and bring all these resources together and provide scientists and engineers capabilities for seamless and transparent access to this visionary infrastructure.

Grid Computing is considered as a major contribution to the ongoing evolution of the Internet towards the vision of Internet computing. Distributed IT resources of all kinds are being operated in a collaborative fashion en route to an integrated new IT cyber-infrastructure, which supplies IT power and/or resources as easily and just as required as electricity from the power grid.

Grid technologies are utilized in a wide range of domains, covering computational, data and information as well as collaborative Grids. Computational Grid, such as the US TeraGrid comprises a large number of interconnected super-computing centers to offer superior computational capabilities. The aggregated computing power is far beyond the limits of a single computing facility at one site and enables the execution of advanced high-performance and/or high-throughput applications. Data Grids, like the EU EGEE Grid, rather focus on the management and sharing of enormous amounts of data for a globally distributed scientific community. Finally collaborative Grids, as for example the EU Virolab project, aim to create virtual laboratories to accomplish a research and collaboration environment for scattered scientists, e.g. to control remote equipment, sensors, and instruments.

1.2 Motivation

The research challenges linked to Grid computing can not be addressed by a single comprehensive and generic approach, which can even be seen by the differentiation of various kinds of Grids. The state of the art in Grid computing was and is rather diverse with a shift towards the adoption of a service-oriented paradigm and an increasing utilization of commercial Web services technologies. However, Grid infrastructures, even when following the latest trends in using Web services, usually address a certain scientific domain such as life sciences, economics, or natural sciences, which can also be concluded from the various Grid-powered projects in respective fields. But most projects create domain-specific, rather proprietary solutions, which are hard to migrate and reuse in other fields.

Besides the proprietary Grid middleware developments, a second even more essential motivation for this thesis is the diverse or non-existent support for Quality of Service. Most Grid projects have their origin in the academic domain, which share their resources on a voluntary basis. This academic model is not well suited for QoS and business purposes. If QoS can not be guaranteed in an economic context, the industry is usually not interested in picking up a certain solution for later business exploitation.

Given these circumstances, this thesis aims to contribute to the evolution of Grid computing by comprehensively investigating the state of the art as well as proposing a novel Web services-based Grid infrastructure supporting Quality of Service to address the mentioned issues. QoS-aware Grid computing envisaged in this work comprises on-demand provision of HPC applications as Grid services and negotiable QoS guarantees for clients on a case-by-case basis.

The proposed approach rests upon the basic assumption, that large resource-intensive applications from scientific and engineering domains essentially require Quality of Service comprising real guarantees with respect to time and price in order to be utilized in e-sciences and eventually improve the e-challenges at hand. The current technology available does not address sufficiently this situation and hence this thesis proposes a customized approach to guarantee QoS for Grid services.

1.3 Contribution

The main work in the context of this PhD thesis envisages an integrated Web services-based Grid computing environment with support for dynamic and negotiable Quality of Service. More precisely, this works comprises the following major contributions:

- Design and implementation of a service-oriented Grid environment
- Quality of Service models for capabilities, requests and negotiation
- Secure Quality of Service support infrastructure

The first major contribution comprises the design and development of a prototype service-oriented Grid middleware based on Web services. The realized software framework consists of a client and service infrastructure to conveniently set up and maintain Grid services, which expose native applications or data as well as seamlessly and transparently provide access to these services.

Towards the Quality of Service support generic models are being developed. The models main purpose is to identify potential challenges and underpin the subsequently presented practical solution theoretically.

Finally, the secure Quality of Service support represents the third major contribution of this thesis. The QoS infrastructure that has been designed and implemented, constitutes as a key distinguishing feature in contrast to existing environments. The realized QoS support emphasizes parallel HPC applications and aims to guarantee the scheduled execution time and the price. The implementation relies on a reservation-based approach and the basic assumption that performance requirements of a certain parallel application are predicable by corresponding QoS models.

In addition to the negotiable QoS support a security infrastructure with state of the art security solutions has been assembled in order to provide best practice security for sensible data.

1.4 Thesis organization

The basic organization of this thesis is depicted in Figure 1.1 following a building-style approach. The foundation is represented by the basic technologies which introduce a number of further on required topics. Resting upon the basics, a number of pillars constitute two major domains this thesis is concerned with, namely the basic

Grid environment and the QoS support. On top of these developments a practical and experimental evaluation has been performed by the software appliance in projects as well as by a number of experiments.

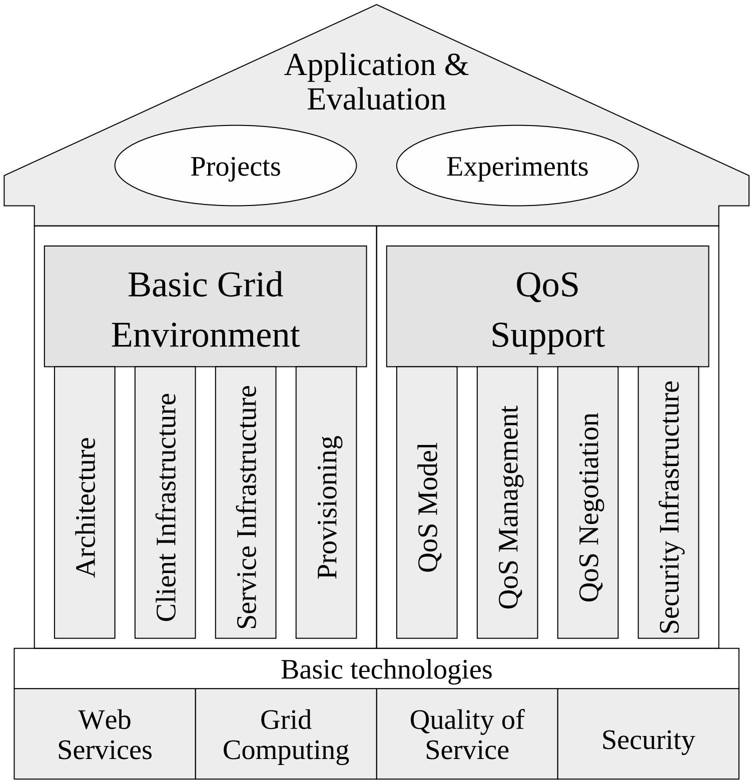


Figure 1.1. Thesis organization

The basic technologies are discussed in Chapter 2, which consists of individual sections for Web services, Grid computing, Quality of Service and Security. The basic Grid environment is presented in Chapter 3, that builds upon its architecture, client and service infrastructure and the provisioning environment. The QoS support rests upon the QoS model introduced in Chapter 4, and the QoS implementation examined in Chapter 5 comprising QoS management and negotiation as well as security as its main pillars.

Under the roof of the thesis organization structure, projects and experiments are presented in individual Chapters 6 and 7, respectively, to evaluate the proposed solutions in a practical and experimental way.

Finally, related work is being discussed in Chapter 8. To round up this thesis an according conclusion and future directions are briefly outlined in Chapter 9.

Chapter 2

Basic Technologies

This chapter constitutes the technical foundation of this thesis and the later on presented work. Therefore, a wide range of diverse topics and their relations are covered and each individual topic contributes pieces to the overall motivation of this work towards a comprehensive approach for QoS-aware Grid computing.

The organization of this chapter is based on four separate sections, which are concerned with the following topics:

- Web service
- Grid computing
- Quality of Service
- Security

Each section introduces its domain and attempts to survey potential contributions to be achieved later on in this work. In particular, Web services are introduced as an essential realization of a service-orientated architecture (SOA), which is most commonly used in todays Internet applications. Grid computing is being outlined with a general emphasis on its evolution and a particular focus to Grid architectures. Finally, Quality of Service (QoS) and security round up this chapter. Both are closely related with security being commonly referred to as subarea of QoS. However, Quality of Service in this chapter is being introduced in a general way, while in the context of security an overview of relevant issues and techniques is presented.

2.1 Web Services

This section introduces Web services as an essential realization of a service-orientated architecture (SOA). Most modern information systems, that provide services via the Internet adopt a SOA approach and utilize Web services technologies for its implementation. The involved basic technologies and standards are outlined following an initial definition and survey of Web services. More comprehensive information beyond the scope of this section can be obtained from the corresponding book "Web Services" by [*Alonso et al.*, 2004].

The description of the technologies and standards comprise SOAP, WSDL and UDDI. SOAP is also formerly known as the Simple Object Access Protocol, which represents the communication protocol used with Web services. WSDL stands for Web Services Description Language and serves to describe a Web service interface and finally, UDDI abbreviates Universal Description, Discovery and Integration, which provides publishing and searching capabilities for Web services in a registry.

Definitions

A generic and short definition of Web services is given in the book by [*Alonso et al.*, 2004] as follows:

> *"Web services are applications accessible to other applications over the Web."*

More precisely the W3C defines a Web service in their term-glossary [*Haas and Brown*, 2004] in this way:

> *"A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."*

These definitions encompass versatile systems, but the common understanding is that clients and servers, which are actually hosting services, communicate over HTTP. Two categories of Web services can be distinguished: Big or traditional Web services and RESTful Web services.

Big or traditional Web services act more in pursuance of the initial W3C definition by utilized XML messages following the SOAP standard as well as WSDL in order to describe their interfaces in a machine-processable way. WSDL has not been defined as a requirement to use Web services with SOAP, but it has been implemented in many Java and .NET SOAP frameworks in order to automatically generate client-side code. Even some industry organizations such as the Web Services interoperability organization (WS-I)[1] mandate WSDL as the defined language to describe Web service interfaces.

RESTful Web services are Web services that follow representational state transfer (REST) principles, which have been introduced by [*Fielding*, 2000]. REST defines a set of principles how resources are defined and addressed using in particular http and all its request methods (get, post, put and delete) as interface. RESTful Web services regained popularity more recently due to their broader definition and lower complexity. These Web services also fulfill the W3C definition, but do not necessarily require complex XML messages or WSDL-based service-API-definitions.

The advantages and disadvantages of traditional and RESTful Web services are also subject to many almost philosophical discussions as presented by [*Weerawarana*, 2007]. This PhD thesis focuses, if not stated otherwise, exclusively on traditional Web services and thus, the term Web services refers only to traditional Web services.

Components and relations

In the context of Web services a number of ordinary terms are used, and in the following these terms and their relations are outlined. Figure 2.1 serves as a starting point by depicting the client- and service-side with all related components of a Web service.

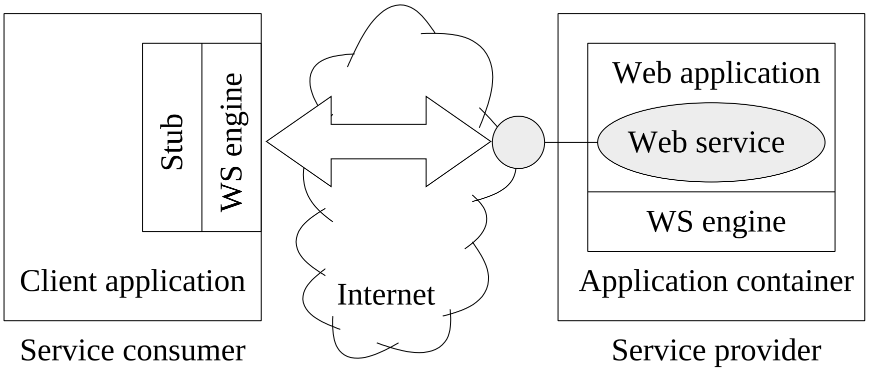


Figure 2.1. Web service overview

Web services (WS) are offered by service providers (SP), mostly organizations, which run an appropriate infrastructure to expose Web services. This infrastructure typically includes a Web server with Web applications hosting capabilities, which is then also referred to as (Web) application container. Web services are usually realized as a special kind of Web applications (web-apps), which are extended using a

[1] http://www.ws-i.org/

Web service engine (usually available as part of a Web service framework) to support provisioning and exposing of the Web service within a Web application. In summary, to offer a Web service, an application container (special Web server) and a Web service engine are required.

The access to a Web service is performed by Web service users, who are also known as service consumers, service requesters or just clients. The actual connection to a Web service is typically initiated by humans or other services, which utilize an according infrastructure similar to the service-side. This typically comprises a Web service client application, which internally utilizes a local representation of the remote service, which is also referred to as a Stub. The client application may be incorporated in all kinds of devices or programs and platforms. The only real requirement is network access to establish a connection to the Web service based on HTTP as learned from the initial Web services definitions.

An example for an application container software is Apache Tomcat[2], which is also commonly assembled in even more comprehensive development environments such as the Java Enterprise Edition (J2EE). Examples for Web services frameworks (also known as SOAP frameworks) are Apache Axis[3] or Microsoft .NET[4].

Publish-find-bind principle

The typical service provision and consumption follow the publish-find-bind principle. It comprises the following actors: a service provider, a service consumer and a service registry. The interaction among them is depicted in Figure 2.2.

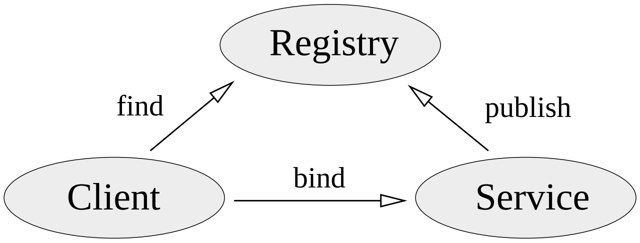


Figure 2.2. Publish-find-bind principle

First the service provider publishes the Web services (i.e. the service description) with the service registry. The service registry follows a certain specification to organize the published services. The service consumer contacts the service registry to find descriptions of services, which fit certain search criteria (e.g. a business category or name). Then the service consumer can select (if necessary) and bind a specific service, as well as execute operations of the service as specified in its service description.

[2] http://tomcat.apache.org/

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

[4] http://www.microsoft.com/NET/

This publish-find-bind principle also implements the concept of loose coupling by dynamically binding clients to services at runtime. This comes with the benefit that changes of the services can be made without notice of the client; only the description has to be updated in the service registry.

2.1.1 SOAP

SOAP specifies an XML-based protocol for exchanging structured information with Web services. It relies on XML for the message formatting and on other application layer protocols such as the mostly used HTTP for message transmission. Originally, SOAP stood for simple object access protocol, but this acronym was dropped with version 1.2 of the standard [*Gudgin et al.*, 2007] which became a W3C recommendation.

Generally, SOAP constitutes the foundation layer of the Web services protocol stack, also referred to as WS-*, because it provides the underlying messaging framework upon which all other layers are built.

The SOAP specification encompasses the following:

- Definition and syntax of SOAP messages
- Model for exchanging SOAP messages
- Framework for data representation in SOAP messages
- Guidelines for SOAP messages that uses HTTP as transport protocol
- Definition of SOAP messages used for remote procedure calls (RPCs)

From the basic topics of the SOAP specification the central element of SOAP is being educed: a SOAP message. Subsequently SOAP messages are being detailed.

SOAP messages

A SOAP message is a structured XML-based document consisting of an envelope as root element. The envelope consists of an optional header- and a mandatory body-element, which may contain a fault-element. The SOAP message structure is shown in Figure 2.3 and the individual elements are outlined in the following.

Envelope: The SOAP envelope specifies an XML document as a SOAP message and encloses the entire SOAP message. Furthermore, the envelope is associated with the XML schema for SOAP messages[5] by an according namespace attribute, as well as with other namespace declarations, as required (e.g. additional data types).

[5] http://www.w3.org/2003/05/soap-envelope

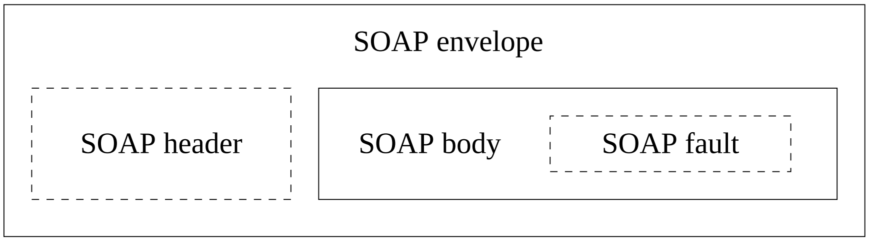


Figure 2.3. SOAP message structure

Header: The optional header-element is a child element of the SOAP envelope. It comprises information such as security information (authentication signatures, keys for encryption, etc.), session information (e.g. session identifiers) or routing information, which typically follows an appropriate standard (e.g. security assertion markup language - SAML).

Body: The mandatory body-element is also a child element of the SOAP envelope and it contains the SOAP message itself. The SOAP body contains information about a remote procedure call (RPC), in particular, which operation to invoke with the according parameters (SOAP request) or the return of the RPC (SOAP response).

Fault: The optional fault element is a child element of the SOAP body and it appears in case of an error comprising information about the error.

In general SOAP messages are created and processed by an according SOAP framework (also known as Web services framework), which internally utilizes a corresponding XML parser to process the XML messages.

The SOAP specification consists of different message *styles* and message *uses*, which both may be defined by an according attribute (i.e. *style* and *use*) in the WSDL description. Both attributes affect the structure and the content of the SOAP body element, as well as the representation of data therein. The *style* attribute is either set to *document* for pure message-driven (also asynchronous) communication or to *rpc* for the invocation of remote operations. The *use* attribute specifies the encoding with either *encoded* which sets the encoding based on the SOAP specification or *literal* which defines the encoding with XML schema types.

In theory there are four different attribute combinations of *style* and *use*, but only three are commonly used according to [*Cohen*, 2003]:

- **SOAP remote procedure call encoding** (RPC-encoded) is also known as Section 5 encoding as defined by the SOAP 1.1 specification. This was the original *style-use* combination which is still applied as default by Apache Axis. SOAP encoding relies on a set of rules which are based on XML Schema datatype definitions in order to encode the data. The SOAP body itself does not need to be conform to a particular XML schema.

- **SOAP remote procedure call literal encoding** (RPC-literal) uses RPC methods to make calls but uses a literal XML encoding, i.e. the SOAP body content has to be conform to a specific XML Schema. This mostly implies individual marshaling of the data and thus, is rather rarely used.
- **SOAP document-literal encoding** is also known as message-style or document-style encoding. This *style-use* combination is applied as default by Microsoft .NET. Furthermore, this combination of style and use is recommended by the Web services interoperability organization (WS-I)[6].

All these mentioned *style-use* combinations are supported by most Web service frameworks even though some have to be used on a low level API.

SOAP with attachments

SOAP messages are based on XML and thus, these are plain text messages. In order to transfer binary data (e.g. graphics data) with SOAP, the data has to be transformed into plain text. The data is then either integrated in the SOAP message directly (inline) or attached to the SOAP message. The latter is referred to as SOAP with attachments (SwA). It is more commonly used with an increasing data size as well as due to the structuring of the SOAP message with an extra attachment vs. a large SOAP message with included data.

The most commonly applied method to incorporate binary data in plain text is the Base64 encoding, which is specified with respect to the use in XML messages in the ITEF RFC 4648 [*Josefsson*, 2006]. It defines the transformation of 8-bit binary data into 64 different characters (6-bit) of the ASCII character set, which eventually results in an increase of the data volume of one third comparing the original binary data with the transformed text data.

The inline-method of transferring binary data with SOAP by including the encoded binary data directly in SOAP messages is only feasible with fairly small data sizes. An increasing size of the binary data will also expand the SOAP message and accordingly the SOAP processing time, because it usually requires the XML parser to load the entire SOAP message into the memory, before the actual content can be processed. Moreover, the content of the binary data does not have an impact on the actual SOAP processing (e.g. which operation to invoke). Due to this chain of effects the use of SOAP attachments should be preferred with increasing data sizes.

SOAP with attachments follows one of the subsequently outlined standards. The actual attachment is linked in the SOAP body using an according reference, such as *href* with HTML-hyperlinks.

Multipurpose Internet mail extension (MIME): This standard origins from Internet emailing and attaching information to emails. It specifies the format of mails

[6] http://www.ws-i.org/

with more than one part (multipart mails) and the separation of these parts using a distinct sequence of characters, which is defined in the header of the MIME message. Furthermore, the header comprises additional information about the content of each part (e.g. content-type or content-encoding). This standard can be applied in a similar way to SOAP messages as to email messages.

Direct Internet message encapsulation (DIME): This standard has been developed by IBM and Microsoft and is fairly similar to MIME. DIME also specifies a header with information on every attachment(-part), but it is encoded binary and includes the length of the attachment part. Consequently, no separation characters are required. With DIME the length of the attachment has to be determined in advance by the sender, but is useful for the receiver, who knows knowing the length of the attachment. Contrarily, the advantage of MIME is vice versa as the sender benefits with no need to determine the length of the attachment. Additionally DIME also supports chunking in order to separate the data into multiple blocks (chunks).

Message transmission optimization mechanism (MTOM): This standard has been specified most recently and utilizes the W3C XML-binary optimized packaging (XOP) mechanism [*Gudgin et al.*, 2005]. MTOM associates and logically includes attachments as components in the SOAP message, while XOP aims to serialize XML more efficiently. Moreover, XOP supports direct streaming of binary data, which improves the performance significantly compared to MIME or DIME as shown by [*Cha et al.*, 2007].

2.1.2 WSDL

The Web services description language (WSDL) is an XML-based language to describe Web service interfaces in order to enable communication with them using SOAP. SOAP provides a generic XML-based communication standard (i.e. protocol), but it does not specify the concrete format of the inputs and outputs a Web service is able to receive, process and return. Therefore, WSDL has been developed to provide a language for documents that describe the input and output of Web services.

WSDL is currently used in two different versions. The most recent version is 2.0, which is endorsed by the W3C as recommendation [*Chinnici et al.*, 2007]. This version had been originally published as version 1.2, but was renamed to WSDL 2.0 because of substantial changes to its predecessor version 1.1 [*Christensen et al.*, 2001]. WSDL 2.0 supports binding to all HTTP request methods, as opposed to version 1.1 which only defines binding to GET and POST, and thus it has improved the support for RESTful Web services. Web services frameworks generally provide support rather for WSDL 1.1 only, such as Apache Axis or Microsoft .NET, than for WSDL 2.0.

WSDL documents

A WSDL document is a concrete description of a Web service interface. Dependent on the concretely used Web service framework the WSDL document of a certain Web service can be retrieved via an URL and it is either generated dynamically upon request or statically in advance to its setup.

The structure of a WSDL document is depicted in Figure 2.4 and it comprises the following elements:

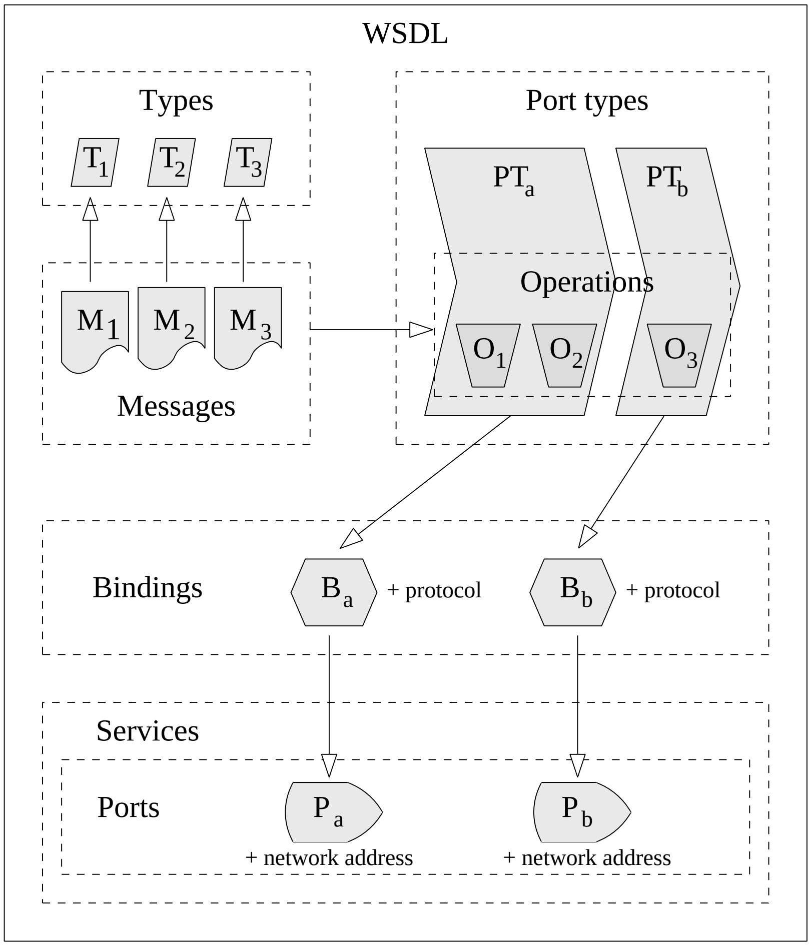


Figure 2.4. WSDL document structure (adapted from [*Merdy*, 2008])

Types: If additional complex data types are required, these can be defined in the types-element in accordance to XML schema definitions.

Messages: This element defines messages and their content, which are exchanged (e.g. upon a certain operation invocation). Typically, in case of an RPC-style Web service a request and a response message are defined for each operation. Moreover, optional parameters require a parts-element to define the name and type of the argument.

PortTypes: The actual interface of the Web service and in particular its operations are defined in usually one or more unique portType-elements. For each operation, the name and all required messages (input, output or fault) are enclosed in separate operation-elements.

Bindings: This element defines the linking of the portTypes to protocols, such as SOAP. In case of a SOAP binding the binding, furthermore, specifies the used transport protocol such as http and the style of the SOAP message (rpc or document).

Services and ports: These elements define the endpoints of the Web service. One service-element may contain multiple port-elements, each containing a name, a URL-endpoint and a reference to a certain binding.

Documentation and imports: These are optional elements, which comprise additional, perhaps human-readable documentation or comments, as well as additional other WSDL documents or XML schema definitions to be imported for a more modular structuring of the WSDL definition.

In summary, WSDL enables the description of the interface to a Web service, while SOAP specifies the actual access independently from the platform of programming language.

2.1.3 UDDI

Universal description, discovery and integration (UDDI) is an XML-based registry service that supports mechanisms to register and query businesses and their Web services using SOAP. With respect to service oriented architectures UDDI realizes the publishing and discovering part of the publish-find-bind principle as shown in Figure 2.2.

UDDI has been developed by major industry players such as Microsoft, IBM and SAP. The first specification was published in 2000. Now, UDDI is endorsed by OASIS in its most recent version 3 specification [*Clement et al.*, 2004]. UDDI has also been integrated in the Web services interoperability (WS-I) standard.

Components

A UDDI registry comprises a set of three components similar to a telephone book:

- **White Pages** comprise the basic information about a service provider such as a contact and an address.
- **Yellow Pages** consist of business-categories (branches) listing related service providers and following a standard taxonomy.

- **Green Pages** contain technical information about the interface of each service (i.e. the WSDL document) and its association with a service provider.

The purpose of these components is to improve service discovery by supporting searches by basic information (white pages), via categories or with technical details from the WSDL descriptions. On the other hand, the service providers have to publish their businesses and services accordingly. The UDDI specification and corresponding UDDI frameworks such as Sourceforge UDDI4J[7] or Microsoft UDDI.Net[8] provide SOAP APIs to perform queries and to publish specific information in the white, yellow and green pages. Additionally an appropriate XML representation of the data model in the registry as well as a WSDL interface definition are specified.

UDDI servers can be distinguished according to their access-type (private, protected and public), which is also often related to the actual end-users' requirements [*Garofalakis et al.*, 2004]:

- **Public** UDDI servers allow querying for all Web service consumers and thus the UDDI registry is a Web service itself. Publishing is usually restricted to a secure channel such as https. Such a system is also referred to as universal business registry (UBR).
- **Protected** UDDI registries require trust between its collaborators (publishers and consumers). Usually such registries are run in a closed environment and accessed either by the trusted collaborators or by selected and monitored external users.
- **Private** UDDI servers are secure isolated systems in a fully closed environment. Typically, such registries are domain specific and run in an internal network only for internal usage.

In general, private and protected usage of UDDI servers prevails, because the information gained in public services cannot be considered as up-to-date and because major software vendors such as Microsoft or IBM have closed their publicly available UDDI nodes. Private or protected UDDI registries usually provide only information about the services hosted within a certain domain (e.g. a company) and hence the maintenance to keep the UDDI registry consistent with the available services is better manageable.

The problem of outdated information in a UDDI registry comes with the specification of a passive collection of service information, i.e. a UDDI registry serves the requests for service publishing, updating or discovery, but there is no defined verification or continuous monitoring (e.g. about the availability of a published service). Thus, discovering a certain service in a UDDI registry, does not necessarily mean, that it is still up and running. By using a Web service crawler engine (WSCE) and

[7] http://uddi4j.sourceforge.net/

[8] http://www.microsoft.com/net/

publicly accessible UBRs [*Al-Masri and Mahmoud*, 2008] determined that only 63 percent of the available Web services on the Web can be considered as active.

Extensions

In order to address the issues related to outdated information in the registry extensions of UDDI have been proposed. In particular [*ShaikhAli et al.*, 2003] recommends UDDIe and [*Du et al.*, 2006] presents the Ad-UDDI system to meet these challenges.

UDDIe stands for UDDI-extensions and it is based on the UDDI specification version 2. A prototype system has been developed in Java at Cardiff University. It introduces extended features to UDDI by introducing additional blue pages as well as a service-lease mechanism. The information provided in the blue pages consists of additional service properties, such as QoS attributes. UDDIe allows service discovery based upon these blue pages information. Furthermore, UDDIe supports a Java Jini-like lease mechanisms, which requires the services to successively renew their lease in the UDDI registry in order to be listed as an online service.

Ad-UDDI stands for active and distributed UDDI and it is based on the UDDI specification version 3. A prototype has been developed at Beihang University. The system performs an active monitoring of services by checking the state of the services continuously. This active monitoring enables clients discovering services with Ad-UDDI to receive up-to-date information about their preferred services.

Both systems provide interesting approaches, but so far, these have not been picked up by the industry.

Summary

This section introduced Web services as an essential realization of a service-orientated architecture (SOA) as well as mechanisms to implement the publish-find-bind principle. SOAP, WSDL and UDDI have been described from the technical point of view. SOAP serves as the communication protocol used with Web services. WSDL enables the definition of a Web service interface and finally UDDI specifies mechanisms for the publishing and discovering of Web services in a registry.

2.2 Grid computing

This section introduces Grid Computing as a vision of a new distributed computing infrastructure. Grid Computing and in particular the Grid was considered as a next step on the evolutionary ladder of the Internet. Actually, even the term Grid as a metaphor for making access to and use of computing power as easy as using electricity via the power grid was one of the buzz-words in information technology from the late 1990s to the late 2000s. Also a huge amount of research and development has been devoted to Grid Computing keeping the researchers busy for more then a decade.

Most recently the interest in Grid computing seems to shift towards Cloud computing, which is promoted by industry due to its pay-as-you-go approach. Contrarily, Grids have their origin in academia and thus more likely provide free access to shared resources. However, Cloud computing adopts Grid and Web technology and furthermore, a huge number of Grid systems which facilitate Cloud-like capabilities such as virtualization, have been deployed and are still evolving. Perhaps further developments in both fields will feed into the next generations of cyber-infrastructures.

This section provides an overview of Grid computing, corresponding definitions with respect to their historical background, technical characterizations and the vision of an according service-oriented Grid architecture.

2.2.1 Definitions

Starting with a generic and short definition of Grid computing, which has been taken from one of the largest users of Grid technology, CERN, and their Grid-Cafe[9]:

> *"Grid computing is a service for sharing computer power and data storage capacity over the Internet."*

This definition immediately raises the issue of differentiating the Grid from the Web: Grid computing uses the Internet to share computing resources, while the Web uses the Internet for sharing information. A comparison of the Grid and the Internet stated by Tom Hawk, the general manager of Grid computing at IBM, sounds similar:

> *"The Internet is about getting computers to talk together; Grid computing is about getting computers to work together."*

The basic idea of Grid computing has roughly been outlined in these initial definitions. The real innovative vision has been introduced by Ian Foster, who is also referred to as the "father of the Grid" [*Braverman*, 2007] and his colleague Carl

[9] http://www.gridcafe.org/

Kesselman. They authored the book: "The Grid: A Blueprint for a New Computing Infrastructure" [*Kesselman and Foster*, 1998] and defined the Grid with a focus on the computing aspect as follows:

> *"A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities."*

Subsequently, they contributed significantly to the evolvement of the Grid. Most notable are the publications concerned with the anatomy [*Foster et al.*, 2001] and the physiology of the Grid [*Foster et al.*, 2002]. The latter eventually resulted in the OGSA vision [*Foster et al.*, 2006], which will be discussed in further detail later in this section. In the concourse of the Grid's evolution its definition has been refined. Foster and Kesselman identified resource sharing and related issues, in particular resource coordination and negotiation of arrangements therein, as a key concept and noted in the second edition of the book "The Grid 2: A Blueprint for a New Computing Infrastructure" [*Foster and Kesselman*, 2003]:

> *"The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a virtual organization."*

In the light of his own definitions Ian Foster wrote a column about the Grid for the On-demand enterprise publications[10], at that time known as GRIDtoday, that captured the essential characteristics of a Grid in a three point checklist [*Foster*, 2002]. A Grid with respect to this checklist is a system with the following primary attributes:

1. The resources of a Grid are not subject to centralized administration.
2. A Grid uses standard, open, general-purpose protocols and interfaces.
3. Nontrivial Quality of Service is achieved in a Grid.

Furthermore, the article revisited the overall vision of *a* and *the* Grid in comparison to the Internet and the following essence can be summarized (note the differentiation between *the* and *a* Grid): *The* Grid is actually the Intergrid, which is

[10] http://www.on-demandenterprise.com/

a collection of many Grids (i.e. the Grid is a Grid of Grids), like the Internet is a network of networks.

Besides these mentioned definitions other notable researchers have made contributions and adjustments to the Grid and its vision, which lead to diverse global understanding. An approach to summarize the current view on the Grid has been made by [*Stockinger*, 2007]. This article presents an empirical survey of attempts to define the Grid by aggregating information gained in many interviews with leaders of IT companies and researchers. The main conclusion drawn in this article was that a common understanding about the Grid vision exists, even though many technological changes and diverse advances have been made in the recent years.

2.2.2 Characterization approaches

The definition section showed that characterizing the Grid is a complex undertaking, alike distinguishing types of Grids. The most commonly known categories of Grids are based on their types of applications as well as on their size and distribution. In the following, both categories are briefly introduced.

Application type Grids

This category differentiates Grids by the type of applications that is run within these Grids. The following types of Grids are distinguished:

- **Computational Grids**: This is the most commonly used and classic form of a Grid. Computational resources such as workstations, PCs or even PC clusters are combined to form a virtual supercomputer, that provides far more aggregated computing power then a single cluster or workstation. Typically compute-intensive applications with a rather well scaling behavior prevail in this kind of a Grid.

- **Data Grids**: This type of a Grid integrates different, mostly distributed and heterogeneous data sources to a virtual database. Issues related to data access and integration dominate in this kind of a Grid.

- **Scavenging Grids**: The idle CPU power of a single PC or workstation is utilized by a scavenging Grid to compute tiny portions of complex operations. The most commonly known examples are SETI@home[11] and Folding@home [12].

- **Semantic Grids**: These Grids utilize semantically enriched services (e.g. compute and data services with additional human-readable and machine-processable

[11] http://setiathome.berkeley.edu/

[12] http://folding.stanford.edu/

descriptions) to maximize the ease of use, sharing, automation and reuse. Furthermore, a major focus has been put on the reuse of technologies invented in the context of the semantic Web.

- **Knowledge Grids**: In this type of Grids focuses on the discovery of knowledge from distributed sources (services) based on the approaches of semantic Grids.

This work mainly focuses on traditional computational Grids, while the other types of Grids are mentioned for the sake of completeness. Thus, in the reminder of this work a Grid refers to a computational Grid, if not explicitly stated otherwise.

Distribution and size of Grids

This category differentiates Grids by their size and distribution. This is comparable to the distinction of networks with respect to their size, such as local area network (LAN), metropolitan area network (MAN), and wide area network (WAN). The following types of Grids are distinguished:

- **Cluster Grids**: Clusters of PCs or workstations, most recently also of playstations, can represent small sized Grids if these systems are compliant with the required Grid attributes. Such Grids are typically located within a room or its components are even mounted within a rack. This often comes with the benefit of high speed interconnection among the Grid nodes, which enables highly parallel utilization. On the downside with intensive utilization these systems reach their boundaries quickly. Sometimes such systems are also referred to as departmental Grids if located and used by a single department of an organization or company.

- **Enterprise Grids**: This type of a Grid is usually referred to if a number of cluster or departmental Grids are combined. In this case the nodes of this Grid are distributed physically within the entire company or organization, but to some degree the infrastructure is still homogeneous. However, security aspects, such as authentication, authorization and accounting play an important role in this kind of a Grid.

- **Campus Grids**: Similar to enterprise Grids this type of Grids refers to one or similar organizations, such as universities. Typically, a university comprises a lot of PCs in (student-)laboratories, which are capable of providing a huge amount of idle CPU-time. Utilization of this CPU-time is targeted in this kind of Grids. The nomenclature of this kind of Grids originated from the distribution of laboratories that are spread over an entire campus of e.g. a university.

- **Global Grids**: Global Grids are the largest type of Grids and are comparable to the Internet. Typically, organizations running enterprise or campus Grids combine their resources on a global scale, mostly to address a certain global research problem such as the EGEE Grid[13]. The global dimension of such a Grid comes with huge challenges such as security, e.g. different types of users and their authentication and authorization, heterogeneity or resources, etc. But thereof in terms of research and development this kind of Grids also constitutes the most interesting type of a Grid.

This work mainly focuses on solutions and in particular on software to address the challenges of global-scale Grids.

2.2.3 Layered architecture

The initially introduced view at the architecture of the Grid has been described by [*Foster et al.*, 2001] in terms of layers. Each layer represents a functionality provided based on the layers beneath. The top layers are typically focusing on the users requirements, while the layers towards the bottom increasingly deal with the hardware. The middle-layers are also referred to as middleware(-layers) which provide abstractions to hardware and enable interoperability of heterogeneous hard- and software.

Once again the Internet was an indication for design of an architecture for the Grid. The Internet protocol architecture also commonly known as TCP/IP model [*Braden*, 1989] consists of a set of communication protocols organized in a layered fashion. In relation to TCP/IP model a layered Grid architecture has been proposed by [*Foster et al.*, 2001], which incorporates layers for: applications, collectives, resources, connectivity and fabrics. The layered Grid architecture and its relation to the TCP/IP model is shown in Figure 2.5.

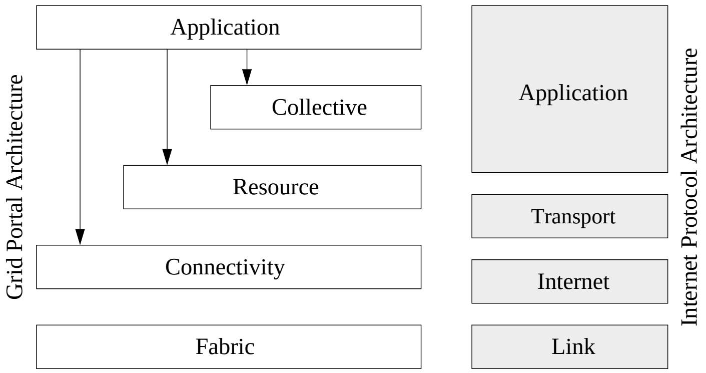


Figure 2.5. Layered Grid architecture [*Foster et al.*, 2001]

Furthermore, the proposed Grid architecture shows that the application layer may access layers beneath at an abstraction level of its choice. The functionality

[13] http://www.eu-egee.org/

of a certain layer is defined by corresponding services and/or protocols and exposed by appropriate application programming interfaces (APIs) and software development kits (SDKs) in order to be accessed by development frameworks and languages. Subsequently the Grid layers are outlined briefly from the bottom to the top.

- **Fabric layer**: This layer comprises the actual compute, storage and network resources which are subject to be shared in the Grid. Furthermore, appropriate uniform mechanisms and protocols are provided to access these resources to enable seamless and collaborative usage in the higher layers.
- **Connectivity layer**: This layer consists of the core communication and authentication protocols. Network communication based on the TCP/IP enables data exchange among Fabric layer resources. Adopting according security mechanisms provides authentication including single sign on, delegation, integration with local security solutions, and user-based trust relations.
- **Resource layer**: This layer provides protocols to obtain information about a specific resource and to manipulate it. The resource layer is entirely concerned with individual resources from the layers beneath.
- **Collective layer**: This layer focuses on the complex coordination of multiple resources. Similar to the resource layer, which deals with a single resource, this layer provides a similar functionality for collections of resources. This is typically far more advanced than with a single resource.
- **Application layer**: The final layer comprises the user applications, which utilize the functionality of the layers beneath at the abstraction layer of their choice.

This layered view of a sample architecture for the Grid provides a high level of abstraction and thereof most Grid designs are compliant with this general architecture. In particular the architecture was developed in accordance with the initial versions (2.x) of the Globus toolkit, which was by that time a collection of almost independent and diverse tools. The Globus software has been redesigned - almost reinvented - in the recent years and it is now one of the most important Grid software infrastructures available, e.g. as used in the EU EGEE Grid[14] or the US caGrid[15]

2.2.4 Service architecture

The growing influence of service-oriented architectures was also also affecting the Grid domain. Initially, Grid infrastructures were run by proprietary developments and toolkits such as Globus, but the general tendency was to raise the level of abstraction

[14] EU Enabling Grids for E-sciencE, http://www.eu-egee.org/

[15] US cancer biomedical informatics Grid (caBIG), http://www.cagrid.org/

by adopting a SOA approach and, in particular, run Grid services based on Web services technology. Some Grid projects like GEMSS[16][*Benkner et al.*, 2005a] and GRIA[17][*Surridge et al.*, 2005] have been early adopters of pure Web services-based Grid solutions.

The Open Grid Forum (OGF)[18] formerly known as Global Grid Forum (GGF) is a major community to drive standardizations in the Grid, also picked up the SOA approach and developed the vision of an Open Grid Services Architecture (OGSA) [*Foster et al.*, 2006]. The OGSA specification outlines a framework with a comprehensive set of services for a Grid derived from a list of requirements. In the following, the Open Grid Services Architecture, the proposed services therein, initial implementation attempts and the transition toward the Web services are introduced.

Open Grid Services Architecture

The Open Grid Service Architecture as proposed in [*Foster et al.*, 2006] and supported by the OGF constitutes a vision of a service-oriented architecture for the Grid. A major objective of OGSA is the specification of services for the seamless access and management of distributed and heterogeneous resources. The utility to perform a certain task with a service and its resource(s) is captured by a set of capabilities. Figure 2.6 depicts some of these capabilities based on three logical and abstract tiers.

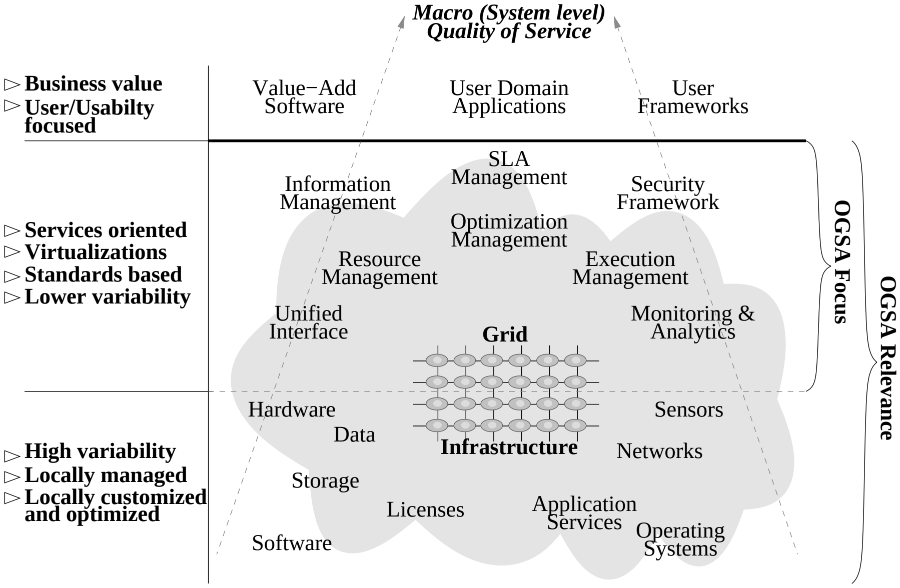


Figure 2.6. OGSA conceptual view [*Foster et al.*, 2006]

[16] Grid-enabled medical simulation services, `http://www.gemss.de/`

[17] Grid resources for industrial application, `http://www.gria.org/`

[18] Open Grid Forum (OGF), `http://www.ogf.org/`

Moreover, the conceptual view illustrated in Figure 2.6 shows the characteristics of each tier, as well as their relevance to and the focus of OGSA. The logical and physical resources can be found in the bottom tier, the capabilities are depicted in the middle tier and the Grid applications are shown in the top tier.

OGSA Services

Logical and physical resources are represented in OGSA with basic services that enable access and management of a single resource. The middle tier comprises the OGSA capabilities that actually provide the Grid functionality based on virtualization of one or more basic services (and their resources). These capabilities are the main focus of OGSA and they are defined in terms of services, interfaces, associated resources as well as their semantic behavior and their interactions. Figure 2.7 sketches categories of OGSA services and outlines concrete services of these categories.

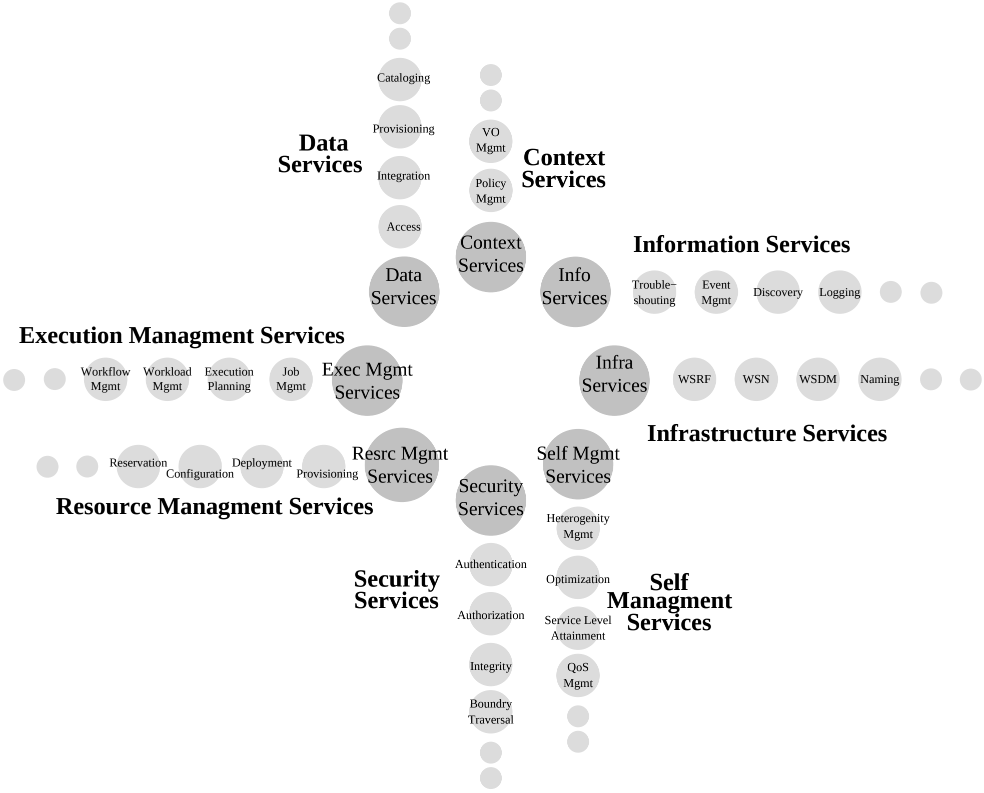


Figure 2.7. OGSA services [*Foster et al.*, 2006]

Concrete OGSA services are generally loosely coupled peers and realize capabilities ideally by different implementations, complex composition, or interactions with other services. The service categories distinguished in OGSA as shown in Figure 2.7 comprise the following:

- Infrastructure services
- Context services
- Data services
- Execution management services
- Resource management services
- Security services
- Self-Management services
- Information services

The OGSA specification [*Foster et al.*, 2006] describes the architecture of a Grid infrastructure with the potentially required Grid services and their capabilities, but it is no technical specification to detail these services formally. In the following a brief description of OGSA realization attempts is presented.

OGSA realizations

The first attempt to provide an infrastructure layer for OGSA was the Open Grid Service Infrastructure (OGSI) [*Tuecke et al.*, 2003], specified again by the OGF and implemented by the Globus toolkit version 3 (GT3). OGSI and GT3 became obsolete soon and have been replaced by the Web Services Resource Framework (WSRF) and its implementations such as the Globus toolkit 4 (GT4). The relation of Web and Grid services, OGSA, OGSI/GT3 and WSRF is shown in Figure 2.8.

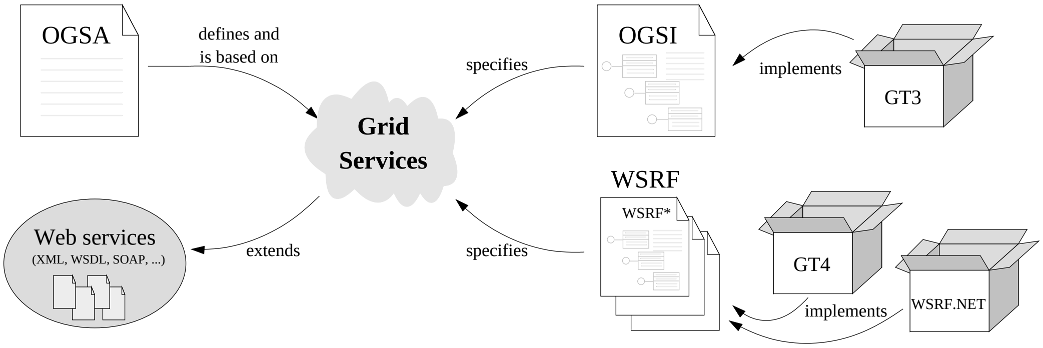


Figure 2.8. OGSA relations (extended from [*Sotomayor*, 2004])

OGSI adopted a stateful Web services approach. OGSI basically extended Web services and creates new services instances upon request to accommodate Grid services that are transient and stateful. Due to maintaining individual states with service instances and the resulting performance problems with many clients, OGSI was replaced by the WSRF soon.

WSRF is a set of OASIS specifications[19] which are summarized in [*Banks*, 2006]. WSRF enables Web services to provide a set of operations to become stateful. In particular, the a WSRF-compliant Web service incorporates its state through its associated stateful resource. The access to and querying of this resource is also specified in detail and performed through endpoint references (EPR) as defined by WS-Addressing [*Gudgin et al.*, 2006].

The history of Grid architectures and their potential realizations show a transition of Web and Grid technologies towards the Web Services Resource Framework as illustrated in Figure 2.9. Developments in both domains, the Grid and the Web started far apart and have been converging with time.

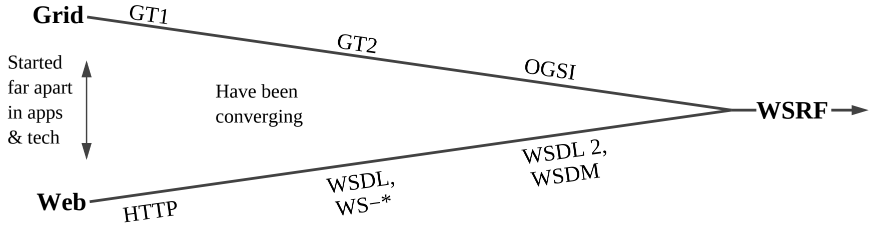


Figure 2.9. Grid and Web technology transition [*Allcock*, 2004]

The incorporation of state in Web and/or Grid services in general and in particular with respect to applying WSRF is being discussed controversial in both the Grid and Web service community. The main criticism about WSRF is that it has limited compatibility with the mainstream Web services interoperability (WS-I) architecture as opposed to industry-driven approaches such as Web Services for Management (WS-Management) standard [WSMgmt]. On the other hand these approaches are not addressing Grid-related issues and thereof are not adopted by the Grid community.

Summary

A comprehensive summary about Grid architectures taking into account the controversial discussion related to WSRF and its alternatives would go far beyond the scope of this thesis. Given the survey presented in this section, the following characteristics of a Grid are considered further on in this thesis: using service oriented architectures, following a model for representation of state (such as defined in WSRF) and adopting mainstream and stable Web services standards and technology. Further related work in this context of Grid computing will be discussed in Chapter 8.

[19] OASIS WSRF version 1.2 comprises specifications of WS-Resource, WSRF-RP, WSRF-RL, WSRF-SG and WSRF-BF available via http://www.oasis-open.org/

2.3 Quality of Service

This section comprises an overview of topics related to Quality of Service. The description deals almost entirely with technical issues and does not question the actual applicability and potential implications of the presented QoS approaches. Quality of Service is firstly defined in a general way, followed by a differentiation using distinct QoS levels. Mostly this section refers to QoS in the context of service oriented architectures (SOA) as introduced in the context of Web and Grid services.

More comprehensive information far beyond this section can be found in the books "Internet QoS" by [*Zheng*, 2001] mostly dealing with technical network-related QoS issues as well as "Technical, Commercial and Regulatory Challenges of QoS" by [*Xiao*, 2008] describing QoS besides the technical issues in the context of commercial and social issues.

A generic description of Quality of Service (QoS) is the ability of a service to guarantee a certain level of quality. This abstract definition of QoS can be applied in different contexts, but also needs an appropriate refinement according to the specific domain such as telecommunication or networking, in which the term QoS originated. Thus the first step in this section is a survey of Quality of Service definitions. Following these initial definitions a formal distinction of QoS levels is applied, which include service-, application-, and networking-level QoS. All these QoS levels address different issues and comprise distinct approaches and solutions, which will be discussed in further detail.

Definitions

Quality of Service in the field of telephony has been defined in 1994 by the Telecommunication Standardization Sector of the International Telecommunication Union (ITU-T) in its recommendation E.800 [E.800]:

> *"The collective effect of service performance which determine the degree of satisfaction of a user of the service."*

Furthermore, the definition has been refined by stating that a combination of aspects actually characterize QoS, including: service support, service operability, serveability, service security as well as other factors specific to each service. This initially shows that QoS is fairly complex and addresses a number of issues.

Quality of Service in the field of data networking has been published 1998 by the ITU-T in its Recommendation X.641 [X.641], which comprises the following definition of QoS:

> *"A set of qualities related to the collective behavior of one or more objects."*

Generally, the purpose of recommendation X.641 is to improve the development and the enhancement of related standards and therefore, it provides appropriate concepts and a terminology with respect to Quality of Service. This rather vague definition also addresses QoS as a set of aspects.

Also more recent work, such as [*Zheng*, 2001], define Quality of Service in the context of the Internet as follows:

> *"The capability to provide resource assurance and service differentiation in a network."*

These definitions are fairly similar by addressing QoS as a collection of assured aspects indicating how a service or resource will behave. Furthermore, these definitions have been made in the context of traditional QoS research focusing on the network-level, but they can also be applied in a broader scope concerning QoS in Web and Grid computing. The range of QoS aspects to consider in order to assure a certain quality just increases and networking may be one aspect while executing Grid applications. A simplified example is assuring a certain network throughput which can be achieved by bandwidth reservation, while guaranteeing the response time of a Grid application may involve booking of networking-, storage-, computing- and potentially other resources.

In summary, Quality of Service represents a collection of aspects indicating the behavior of a service or resource in advance to or while consumption, whereas even a single aspect to guarantee (e.g. response time) may result in a number of aspects to consider. The aspects to guarantee are recorded in an agreement or contract, which is usually subject to negotiation either in advance or while service consumption if the service is adaptable.

In the following networking-, application-, and service-level QoS and approaches therein are discussed. Even if security is usually mentioned in the context of QoS, certain security-related QoS aspects, especially within the networking domain, are not discussed here, but detailed in a separate section of this chapter.

2.3.1 Network-level Quality of Service

The emphasis of traditional QoS research is networking and, in particular, it aims at providing alternatives to the best effort approach of the Internet. The main achievements towards network-level QoS have been standardized in 1990s by integrated and differentiated services. Integrated services address the best-effort-situation by a resource reservation approach, while differentiated services adopt classification of services linked with prioritizing. Subsequently, both approaches are outlined briefly.

Integrated Services

Integrated services, shortly referred to as IntServ, describe a system in which elements exist that are able to guarantee a certain quality e.g. in order to enable continuous video- or sound-transmission. The systems specifies a fine-grained QoS approach in which every router in the system implements integrated services and every application that requires QoS has to book all necessary resources. The underlying mechanism to organize the reservations is the resource reservation protocol (RSVP) as specified by the Internet Engineering Taskforce (IETF) in the request for comments (RFC) 2205 [*Braden et al.*, 1997].

The RSVP protocol is a transport layer protocol to reserve resources across a certain path through a network. The protocol sets up receiver-initiated resource reservations backwards hop by hop to the sender. This traces a path consisting of routers and/or nodes (hops) backwards from the receiver to the sender and each node or router along the path has to confirm the reservation. RSVP specifies exactly how applications have to book and relinquish resources once not needed anymore.

The main obstacle of using integrated services and, in particular, with RSVP comes with the many states that have to be stored in each network element, which results in a poor scaling behavior opposed to the scaling requirements of the Internet. As a consequence, IntServ is not very popular and RSVP is rarely deployed. A new approach appeared from the field of traffic engineering, which provides an extension of RSVP: RSVP-TE. It supports establishing MPLS[20] label switched paths (LSPs), which consider network constraints such as available bandwidth and explicit hops. The protocol was developed in 2001 and recently revised in RFC 5151 [*Farrel et al.*, 2008].

Differentiated Services

In contrast to integrated services and their fine-grained QoS approach, differentiated services or shortly referred to as DiffServ adopted a more coarse-grained solution utilizing a system based on classification of services. DiffServ provide mechanisms for classifying and managing network traffic to distinguish between certain classes of network traffic in order to prioritize one service (e.g. for voice or video transmission), while simply applying best-effort guarantees to non-critical services. The IETF specified an architecture for differentiated services in RFC 2475 [*Blake et al.*, 1998].

The principle of DiffServ is traffic classification by associating each data packet with a traffic class and treat each data packet according to its traffic class. This enables each router to differentiate network traffic based on its class, while each traffic class can be managed and configured independently. Given a certain traffic class, a data packet can be treated with the associated priority to ensure e.g. a

[20] Multi Protocol Label Switching represents a data transport mechanism in packet-switched networks.

high quality. The system does not specify any concrete type of traffic, which should be given priority, it rather provides a framework to allow classification and thereof differentiated treatment. Even though DiffServ recommends a standardized set of traffic classes to improve interoperability.

The main advantage of differentiated services constitutes its full implementation at the core of the Internet, such as routers and nodes, and no additional complexity is associated with reservations, such as collecting states and even payment records. Furthermore, it does not require an advanced setup or time-consuming end-to-end reservation negotiation for each data flow as with RSVP.

DiffServ also come with disadvantages. The basic principle of using prioritization does not provide any real guarantees, which then can be used for critical applications and consequently it can't be sold in such an environment. Another criticism of DiffServ is that it leads to dropping of packets with low priority in case of insufficient bandwidth, which firstly leads to waste of resources that have already been used to carrying the packet so far. Secondly dropping of low-priority packets leads the clients to increase the priority of their applications. The obvious result is, that eventually the differentiation is lost and a pure best-effort network is established (again), due to all packets having a high priority.

Differentiated Services are most commonly used by Internet Service Providers today, to balance their fair use networks by differentiating Internet applications and/or its customers. For example, peer-to-peer (P2P) network applications and protocols may be given a low priority to generally reduce P2P-related network traffic, while money making business costumers may be privileged with a high priority for their entire network traffic.

Other approaches

Besides IntServ and DiffServ other approaches exist to enable network-level QoS on the layers beneath the transport layer of the OSI model [X.200] such as IPv6, the best-known example But these are not discussed in this context, because most applied protocols on these layers are specific to a certain application domain such as ATM [*Joel*, 1993]. Furthermore, to ensure bandwidth availability dedicated network lines or systems are often used, such as bandwidth reservation for user work (BRUW) [*Hwang and Riddle*, 2005] in the context of the Internet2 or advance multi-domain provisioning system (AMDPS) [*Patil et al.*, 2006] within the GEANT2 project[21].

In summary, some approaches exist to ensure network-level Quality of Service and the more promising are based on resource reservation, rather then prioritizing. But in general increasing the network bandwidth overall is probably more practical than developing QoS mechanisms in the existing infrastructure, as concluded by the Internet2 QoS working group[22].

[21] GEANT2, `http://www.geant2.net/`

[22] Internet2 QoS WG, http://qos.internet2.edu/wg/

Apart from the technical dimension the use of QoS mechanisms also has a social dimension. The Internet is considered as neutral or net-neutral, which means simplified that all users and their traffic is treated equally (at least outside of the network of their ISPs). This net-neutrality would be dismissed by allowing Internet service providers to charge a QoS fee and prioritize the ones that pay more as outlined in the big picture of [*Xiao*, 2008].

2.3.2 Application-level Quality of Service

The purpose of application-level QoS is - as the name states - quantifying and assuring the behavior of an application with respect to application-specific characteristics, such as performance, security, dependability, adaptability and many others. Although the list of characteristics is rather long, the focus on performance and, in particular, on execution-/response-time seems to be addressed most of all. According to [*Woodside and Menasce*, 2006] application-level QoS gained increasing importance, but also constitutes the Achilles' heel of services offered over the Internet. Achieving adequate QoS-support at the application-level is challenging.

Basically guaranteeing application runtimes requires comprehensive knowledge about the application itself and the resources required for a specific task in advance, as well as a mechanisms to pre-book the required resources. This is commonly referred to as advance reservation [*Smith et al.*, 2000]. These are critical prerequisites for achieving application-level QoS and thereof are discussed subsequently.

Performance prediction

The prediction of the runtime (and potentially other factors) of an application in advance to its actual execution is referred to as performance prediction. It is usually a complex task, especially if applied for parallel and/or distributed programs as discussed in [*Nudd et al.*, 2000] or [*Pllana and Fahringer*, 2005]. Typically, performance prediction is conducted in the context of performance engineering and performance analysis, which is more commonly known in software engineering as profiling.

Performance prediction can roughly be divided in three categories:

Simulation-based prediction obtains performance data from simulation systems, that simulate the execution of the application.

Profile-based prediction is the classic approach that divides a program into a set of blocks in an execution path and the total execution time is the sum of execution time of each block multiplied by its execution frequency.

Analytical modeling constitutes a mathematical approach to calculate the performance with an analytical model using certain indicators especially about the job's input data.

Hybrid approaches such as proposed by [*Pllana and Fahringer*, 2005] combine the mechanisms above and improve the prediction results for certain applications significantly.

Other mechanisms may also be applied to estimate the performance in advance, like gaining performance information from historical runs of an application as presented in [*Smith et al.*, 1998]. All application executions are stored in a database that expands with time and the number of application runs. Moreover, neural networks may be feasible for this task, but generally performance prediction and the utilized approaches are highly dependent on a concrete application, the input data, the execution infrastructure, etc.

In summary, there are mechanisms to predict the runtime and other factors of an application in advance to an actual execution, which are essential to determine and pre-book the required resources. Inaccurate estimations will either result in resource bookings that waste resources, i.e. if an application runs faster than anticipated, or the booking does not comprise enough resources to complete the application, which is even more problematic, if the application can not continue e.g. due to other resource reservations. Hence a precise prediction of the required resources for a specific application job is a strong prerequisite for accurate advance resource reservations.

Advance resource reservation

Applications or programs require resources to execute a specific job and in a best-effort-based system usually application-jobs are queued until the resources are available. Moreover, these resources are often not available exclusively, which prevents proper prediction of their behavior, because different processes/tasks may interfere. In order to guarantee a program execution within a certain time limit, the required resources have to be reserved in advance (i.e. resource reservation). This reservation will make sure that an application has access to the resources as required, e.g. exclusively for a certain period of time. The required resources have to be pre-planned and determined e.g. by performance prediction and a reservation has to be made with an according advance resource reservation facility. Both constraints have to be fulfilled to guarantee certain qualities of an application like runtime limits.

The actual advance resource reservation process is usually performed by an according facility, e.g. in case of compute resources by a scheduling system, that manages the booking procedure and the associated reservations. Thereof such facilities that support advance reservation capabilities are required as well (c.f. Integrated Services and the RSVP for network-level QoS). Compute resources like CPUs or even entire machines in a cluster can be managed and assigned to tasks/jobs by scheduling and/or queuing systems. Examples for such job scheduling systems, that support advance reservation are the Sun Grid Engine [SGE][23], Maui [*Jackson et al.*, 2001] or Cosy [*Cao and Zimmermann*, 2004] on cluster systems or the scheduling systems

[23] http://gridengine.sunsource.net/

used in the ICENI [*McGough et al.*, 2004] or Gridbus [*Sulistio and Buyya*, 2004] Grid. Job allocation on a Grid- or cluster-scale may also be referred to as coarse-grained scheduling.

Opposed to Grid- or cluster-scale, fine-grained job allocation and scheduling as required to perform multitasking on the level of the operating system deals with assigning threads, processes or data flows to system resources. This assignment procedure is usually performed according to a scheduling algorithm in order to load-balance the system effectively and/or achieve a certain Quality of Service. The most commonly known and simple algorithm used is the round-robin scheduling algorithm, which assigns equally portioned time slices to each process in a cyclic order. This fine-grained job scheduling is mentioned for the sake of completeness, but it is not further discussed in this work.

Resulting requirements

In summary, to enable application-level Quality of Service two requirements have to be fulfilled: a possibility to predict the performance of an application dynamically dependent on different input data sets and varying machine configurations as well as a facility to reserve resources in advance. Both issues and corresponding solutions have been introduced briefly.

2.3.3 Service-level Quality of Service

Service-level Quality of Service is less defined, but in general it aims to address issues arising with application- and network-level QoS as well as with other QoS-related aspects of service-oriented architectures (SOA). Service-oriented systems comprise services that are accessible via a network and perhaps expose native applications. Thus service-related QoS comprise generally a combination of network- and application-level QoS. Furthermore, other aspects such as security (e.g. security services for certificate issuance) are addressed by service-level QoS, which will be discussed in Section 2.4.

Quality of Service discussed in the context of Web services typically addresses either a single service, that expose a rather simple application or comprises non-complex application logic accessing e.g. a single data source, or a collection of such services alike. Hence QoS-related research dealing with a single services, like the often referred stock quote services, concentrates on network-level QoS aspects, such as availability, reliability, response time, etc. If a collection of services is addressed, like in the typical travel agent scenario, aggregation of QoS aspects (e.g. total price) are investigated and appropriate negotiation and aggregation mechanisms are developed. A common aspect of service-level QoS is the establishment of an agreement about the qualities of a service, mostly as a result of a negotiation process. Such an agreement is also referred to as a service level agreement (SLA).

Service level agreements

A service level agreement (SLA) formally defines the level of service. Practically, an SLA constitutes a negotiated agreement between two parties (mostly a provider and a consumer) about the consumption/usage of one or more services. According to the SLA information zone[24] an SLA includes: a definition of the involved services, according performance measurements, a problem management description, duties of all parties, warranties, methods of recovery and the termination of the agreement. A few specifications of standard SLAs for Web services exist, such as the Web Services Level Agreement (WSLA) language specification [*Heiko et al.*, 2003] and the Web Services Agreement (WSA) specification [*Andrieux et al.*, 2007], which are both subject to further discussion on the concourse of this work and also comprise the mentioned information.

The subject of an SLA is the final result of the service as perceived by the consumer. The service provider has to deliver the service while the consumer has to use the service, both as agreed in the SLA. This usually includes certain levels of service such as level of performance, price, availability or other attributes. The level of service may also be specified as a target in a defined range or minimum, all subject to the commonly agreed understanding specified in the SLA. Furthermore, penalties may be agreed if one of the involved parties does not act compliant with the SLA.

A special form of SLAs are mutual service level agreements of the same service, which constitute a responsibility of the involved parties to deliver a service in the same fashion and quality as the others do. Such mutual SLAs are often established between Internet service providers which commit themself to route the traffic of the other provider and vice versa. Often these agreements are also used in the context of differentiated services to prioritize a certain traffic.

Summary and outline

In summary, the identified levels of Quality of Service have a certain focus and come with distinct solutions, but no comprehensive approach, which incorporates application-level QoS on the service-level based on according service level agreements, can be identified so far. This work focuses on negotiable SLA-based Quality of Service with Grid services exposing parallel HPC applications with varying execution times depending on the input data sets and used machine configurations.

[24] http://www.sla-zone.co.uk/

2.4 Security

Security is a broad field of interest in information technology (IT) with very varying objectives, which may range from protection of information to the preservation of service availability. This also comes with different implementations, which again may range from physical limitations such as human access control using dedicated hardware to the application of sophisticated security software and protocols. This section emphasizes software-related security and in particular focuses on network-related security aspects, which also constitute a basic requirement in Grid computing.

Network security attempts to protect a computer network infrastructure by applying certain policies and mechanisms to prevent unauthorized access, to perform traceable authentication, to run consistent and continuous monitoring and logging, as well as to ensure confidential message and data exchange. Besides the secure message and data exchange these issues are addressed by authentication, authorization and accounting, which are commonly known as AAA. Moreover, when adding auditing as well, it is referred to as AAAA. All mentioned issues (including encryption) are referred to as security facilities and will be described in further detail subsequently.

A bunch of mechanisms and techniques is used in these security facilities which ranges from cryptographic methods such as RSA to the use of logging and monitoring systems. A basic foundation of network security is the usage of digital certificates, which are organized and issued by a public key infrastructure (PKI). Following an introduction to encryption and AAAA, this section comprises a detailed description of PKIs and their application on the transport layer.

2.4.1 Security facilities

Subsequently encryption, authentication, authorization, accounting and auditing, as well as related topics are introduced in order to present an overview of the terms and technology used for the later description of PKIs and their application.

Encryption

Cryptography deals with the practice and study of hiding information, and encryption is a fundamental tool to achieve this hiding of information. Encryption has a long history starting with mainly military purposes to facilitate secret communication, but now it is used in many kinds of civilian systems, in particular, all kinds of computer networks and the Internet, but it still serves the same purpose: confidential communication [*Kahn*, 1967].

The main objective of encryption is to hide information. Therefore, a transformation of information into a different form is utilized to make the information unreadable to anyone, except to those utilizing special knowledge. The process of transformation is usually performed by an algorithm, which is also called cipher. The result is encrypted information, also known as ciphertext and the special knowledge is referred to as a key. The reverse process of encryption is decryption, which makes encrypted information readable again, but since encryption implies the possibility of decryption, it is mostly not mentioned separately.

Basically encryption distinguishes two kinds of applied algorithms: symmetric and asymmetric key algorithms [*Goldreich*, 2001]. The symmetric encryption requires both the sender and receiver of a message to share the same key and as it should only be known to both of them, it is also referred to as secret key. Asymmetric key encryption was first introduced in 1976 and relies on a mathematically related keypair, that is also referred to as public and private key [*Diffie and Hellman*, 1976]. A message can be encrypted with one of the keys and only decrypted with the other one. As the name states, the public key can be freely distributed, while the private key remains secret. The calculation of the private key only knowing the public key is computationally infeasible. Both keys are generated as an interrelated pair.

Generally the usage of asymmetric key algorithms is more compute-intensive compared to using symmetric key algorithms, but the latter has a main disadvantage in the key-management. Hence, in practice mostly a hybrid approach is utilized, where a secret key is generated ad hoc and exchanged using asymmetric key encryption (e.g. as detailed in Section 2.4.3 with the transport layer security protocol).

Authentication

Authentication is commonly understood as establishing or confirming that something or someone (i.e. a subject) is authentic. This means that claims expressed by or about a subject are genuine. Usually this involves the confirmation of the identity of the person or the assuring that an IT-system or -service is trusted.

There are a number of procedures and mechanisms to ensure authenticity, which are commonly referred to and grouped as authentication factors. Traditionally these factors depend on:

- Something the user has (e.g. hardware tokens: smart card or mobile phone)
- Something the user is or does (e.g. voice or fingerprint)
- Something the user knows (e.g. password or PIN)

Most recently a fourth factor is also utilized in the IT-systems, which is the social network of the user [*Brainard et al.*, 2006] or in other words:

- Somebody the user knows (i.e. social network)

Additionally other authentication factors are considered with respect to the application domain, such as location-based authentication (e.g. prevent use of credit card in two places) or time-based authentication (e.g. to allow access only during office hours).

In the context of information systems cryptographic methods are usually utilized to establish authenticity. Digital signatures [*Lysyanskaya*, 2002] and challenge-response authentication such as CHAP [*Simpson*, 1996] have been developed and so far are considered as not spoofable as long as the originator's private key has not been compromised.

Authorization

Authorization describes the procedure to decide whether to grant access to a certain IT resource or not. Access is only granted to those resource consumers that have been permitted to use them. This permission is commissioned e.g. by access control lists (ACLs), which are usually subject to be defined by the resource owners or managers. IT resources contain all kinds of data (e.g. files, databases), applications, devices or services, which are guarded by authorization. Typical consumers are either users or other resources.

Authorization is often mixed up with authentication or even thought both handle identical issues. Also many commonly adopted security protocols and standards are based on this assumption, but actually authorization as deciding whether to grant access is a different concept than authentication which verifies the identity. Even though, authorization is usually dependent on authentication. The dependency gets even more clear, when describing both concepts in further detail: authentication is the verification process when a subject claims to be treated as if acting on behalf of a given user or resource, while authorization is the verification process if an authenticated user is privileged with the authority to perform a certain action.

A fairly familiar usage of authentication and authorization is access control in a network of personal computers. The authentication is performed by logging in the system e.g. with username and password or more sophisticated by using smartcards or fingerprints. The authorization takes place on every access to or usage of a network resource by deciding if the authenticated user has the authority to access or execute a certain action on the resource. Usually the *principle of least privilege* is applied in this context, which was first introduced by [*Saltzer and Schroeder*, 1975] and states simplified that a subject should only be given those privileges required to complete its task. Applying this principle to the authorization performed on every resource access, the user should only be granted access (also distinguishing between read and write access) to resources he really requires to do his job.

The authorization in the context of access control follows a certain model. Subsequently different access control models are introduced briefly:

Discretionary access control (DAC) is an object-centric approach, by restricting a subject's access to a certain object. Generally this is applied to limit the access of a user to a certain file. This type of access control has been introduced in the [Orange Book 1985] and applied in UNIX operating systems since.

Mandatory access control (MAC) limits the abilities of a subject to access (i.e. perform or grant some sort of operations on) an object as with DAC and also keeps compliance with a general (e.g. organization-wide) policy, which is mandatory to observe and maintained by a security policy administrator. This also enables organizations to define a central policy according to minimum security restrictions or legal regulations. MAC was also initially introduced in the [Orange Book 1985] and refined in [*Loscocco and Smalley*, 2001] for the security-enhanced Linux (SELinux)

Role-based access control (RBAC) introduces an abstraction layer with the concept of a role, which a user is assigned to and the access is granted based on the role. In order to access a certain resource the user is required to be assigned to, a role and the permission to access the resource has to be given for the role. RBAC is considered to have a great flexibility as user-to-role and permission-to-role relations are many-to-many. This means, that a user can have many roles and a role can be assigned to many users; furthermore, a permission can be assigned to many roles, and a role can have many permissions. RBAC has been introduced in [*D.Ferraiolo and Kuhn*, 1992].

More sophisticated models to manage access control in distributed systems are policy-based or attribute-based access control (PBAC or ABAC). Both appeared most recently to manage complex authentication and authorization processes, which are not subject to a single decision or a single property (such as being member of a certain role). Furthermore, these models are built on and/or extend existing access control models as introduced before. Further explanation of these models in the context of service oriented architectures and identity management can be found in [*Li and Karp*, 2007] but these are beyond the scope of this section.

Accounting

Accounting describes the procedure of tracking the utilization of a certain IT resource globally and also with respect to record each users consumption of a resource individually. Typically, all subjects (users or resources) have accounts and the consumption or usage is charged against these accounts. The gained information may be used solitary or accumulated in management, marketing, planning, billing, or in other purposes. The process of deriving knowledge or support decisions with the data from accounting is also referred to as *business intelligence* [*Luhn*, 1958], which has a long history; but it gained increasing interest with larger companies and more powerful information technology.

Accounting can either be performed real-time or in batch-mode. Real-time accounting is performed concurrently with the consumption of a resource, while batch accounting refers to processing the usage information at a later time. The most important use of accounting is billing and therefore, typically, the following data is gathered: The authenticated user (identity), the authenticated resource or service, as well as the begin- and end-time of the usage.

Accounting is generally a huge field of interest with manifold economic impacts, but in this context the focus is rather on basic IT-requirements of the accounting, which constitute the gathering and storing of the required accounting information. More information can also be found in [*de Laat et al.*, 2000].

Auditing

Auditing is related to accounting and with respect to IT-systems it is also closely linked to logging and monitoring. Generally, auditing enables traceability of an organization, system, process, etc. with respect to ascertain the validity and reliability of information, that is usually provided by accounting and/or logging information. Auditing can also provide a possibility to assess a system's internal control mechanisms. Also very closely related is *provenance* which aims to ascertain and document the origin of computerized data to assess its genuineness at any point in time as targeted in EU Provenance project[25].

In the context of IT-systems accounting and logging subsystems are generally used to supply necessary data to auditing purposes. Accounting is usually performed in a domain-specific manner according to a defined protocol such as RADIUS [*Nelson and DeKok*, 2007] for dial-in Internet service providers, while logging is subject to certain guidelines provided by the used logging framework in order to ascertain important events for later usage (e.g. auditing) and also for debugging. Furthermore, active log-monitoring is performed by intrusion detection systems which aim to detect misuse or anomalies in the system to provide an early warning or at least an after the fact damage analysis. A comprehensive survey of security design guidelines for Web services with respect to auditing and logging can be found in Chapter 3 of [*Meier et al.*, 2008].

In the following the technical realization in a PKI as well as a concrete implementation on the transport layer is being described.

2.4.2 Public Key Infrastructures

Public key infrastructures (PKIs) originate from the cryptographic domain to provide digital certificates, which associate a public key with unique user attributes (identity). Furthermore, a PKI defines policies and operational procedures which

[25] EU Provenance project, `http://www.gridprovenance.org`

constitute the foundation for the management of certificates and keys used by public key-based security systems.

Certificates are issued and signed by a trusted third party named certification authority (CA). These certificates enable users to authenticate each other using the public key information in their certificates as well as to establish confidentiality and message integrity without having to exchange a secret token in advance.

Components of a PKI

According to [Wiki-PKI] and [*Xenitellis*, 2000] a public key infrastructure comprises the following components:

Digital certificates: Digitally signed data to certify the authentication of a certain object/user/service.

Certification authority (CA): A CA is an organization that acts as the trusted third party to issue certificates by signing requests for certificates.

Registration authority (RA): A RA is an organization or a service that guides or supervises the process of requesting a new certificate by an applicant with the CA. Usually it verifies and signs the certificate request that is sent to the CA. This is either done automatically by a service or supervised by a human registration authority officer.

Validation authority (VA): A VA is a service or a local software component that checks if a given certificate is still valid. The validation process relies on up to date information given by the CA, such as according lists of certificates which are rejected (certificate revocation lists). This is either done by a separate service or software component according to a validation protocol.

Certificate revocation list (CRL): A CRL is a list of certificates, which have lost their validity before they expire. Possible reasons are wrong certificate data (attributes) or a compromised key. CRLs do also expire and hence these lists have to be updated on a regular basis. Alternatively, an online verification can be performed using a verification authority, which is usually used with online financial transfers.

Documentation: A PKI keeps a number of documents, that describe the principles of the PKI such as the certificate issuance process, guidelines for the key management (handling and creation), optional legal documentation as well as operational security procedures. Usually a CP (certificate policy), a CPS (certificate practice statement) and a PDS (policy disclosure statement) are kept.

The principle of a public key infrastructure is shown in Figure 2.10. All major roles as defined previously are illustrated as well as the important interactions and procedures are depicted as follows:

First a user applies for a certificate. For this purpose the user generates a key-pair and requests a certificate with a public key and according attributes at a registra-

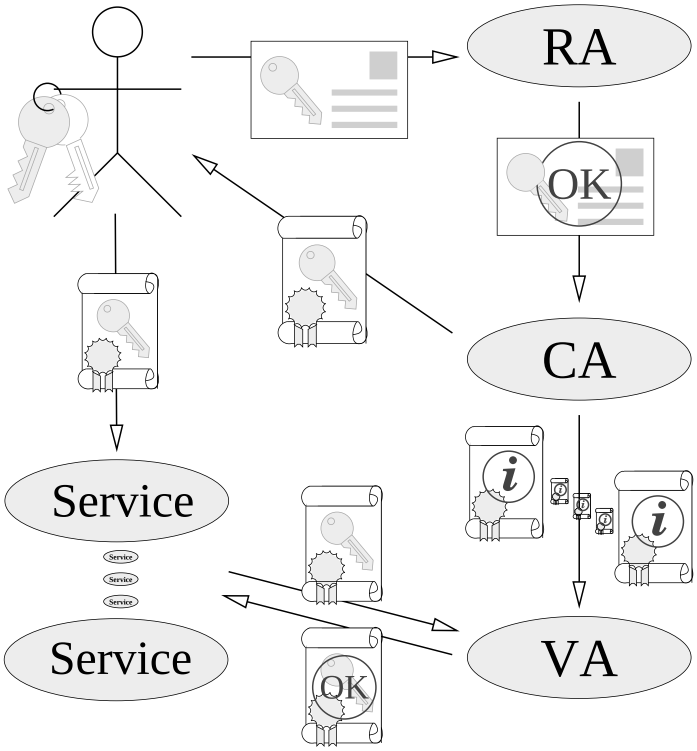


Figure 2.10. Principle of a public key infrastructure

tion authority (RA). The latter confirms the user's identity by signing the certificate request (c.f. OK-sign) for the certification authority (CA). Now the certificate authority is able to issue a proper certificate. The user which is then privileged with his own certificate, can then digitally sign a contract with a service using his certificate. The identity of the user can then be checked by the contracting service with a validation authority (VA) which retrieves the information about all issued certificates from the certification authority. The validation process may also be organized using certification lists, such as certificate revocation lists.

X.509 Certificates

Digital certificates associate a public key with unique user attributes (identity) in order to enable authentication, confidentiality and message integrity with other users that are privileged with a digital certificate. Generally user attributes (e.g. name + email + date of birth, etc.) constitute an individual identity, which are unique at least within a single CA. In order to standardize certificates and PKIs in general the Telecommunication Standardization Sector (ITU-T) developed the X.509 standard for public key infrastructures. It specifies formats for public key certificates, attribute certificates, revocation lists and the certification validation.

An X.509-based public key infrastructure comprises one or more hierarchically organized CAs, which issue certificates binding a public key to a distinguished name (DN) or to an alternative name (e.g. e-mail address or DNS-entry, c.f. attributes). This naming follows the X.500 series, which constitute a number of standards for computer networking developed by the ITU-T and supported by the international organization for standardization (ISO). The most recent upgrade of the X.509 standard is version 3 and it is specified in [*Cooper et al.*, 2008].

The structure of an X.509 certificate in version 3 comprises the following, which is usually encoded in one of the subsequent formats: distinguished encoding rules (DER) [*Dubuisson*, 2008], Privacy Enhanced Mail (PEM) [*Balenson et al.*, 1993], Public Key Cryptography Standards (PKCS) #7 [*Kaliski*, 1998] or PKCS #12:

- Certificate
  - Version
  - Serial Number
  - Algorithm ID
  - Issuer
  - Validity (Not Before and Not After)
  - Subject
  - Subject Public Key Info (Public Key Algorithm and Subject Public Key)
  - Issuer Unique Identifier (Optional)
  - Subject Unique Identifier (Optional)
  - Extensions (Optional)
- Certificate Signature Algorithm
- Certificate Signature

The issuer information and the unique subject identifiers have been introduced in version 2 and the optional extensions in version 3 of the X.509 standard.

Listing 2.1 captures a sample certificate containing all necessary information as defined by the X.509 standard version 3. For the sake of simplicity the content of the certificate has been simplified (e.g. validity) and the information about the subject public key and certificate signature has been shortened.

The validation of a user certificate requires the issue certificate, which is usually a CA certificate. In the example of Listing 2.1 the user certificate belongs to a fictional character and the issuer is the certification authority of the department of Scientific Computing at the University of Vienna. The actual validation procedure comprises the use of the RSA public key of the CA certificate to decode the signature of the user certificate to retrieve a hash value (using e.g. MD5/MD6 or SHA1), which is

```
Certificate:
  Data:
    Version: 3 (0x3)
    Serial Number: 0001 (0x0001)
    Signature Algorithm: md5WithRSAEncryption
    Issuer: C=AT, ST=Vienna, L=Vienna, O=University of Vienna,
            OU=Scientific Computing Certification Authority,
            CN=SC-UNIWIEN/emailAddress=ca@par.univie.ac.at
    Validity
      Not Before: Jan  1 12:00:00 2008 CET
      Not After : Jan  1 12:00:00 2009 CET
    Subject: C=AT, ST=Vienna, L=Vienna, O=University of Vienna,
             OU=Department of Scientific Computing,
             CN=Donald Duck/emailAddress=dd@par.univie.ac.at
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public Key: (1024 bit)
        Modulus (1024 bit):
          00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:
          [...]
          00:00:00:00:00:00:00:00:00:00
        Exponent: 000000 (0x00000)
  Signature Algorithm: md5WithRSAEncryption
    00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:
    [...]
    00:00
```

Listing 2.1. X.509 sample certificate

then compared to the actual hash computed over the rest of the user certificate. If both obtained hash values match, it can be assured that a public key belongs to the stated user (subject).

A CA certificate is a self-signed certificate, as issuer and subject are the same. Furthermore, the CA certificate can be inspected by checking if a CA attribute is present in the X.509v3 extensions section. Generally, there is no possibility to verify a CA certificate, except by checking it against itself. Thereof CA certificates such as of Thawte[26] are recognized as trusted by all conventional Web browsers by default. The private key belonging to such a long-lived, globally trusted certificate, which can be used to sign other certificates has to be well-guarded and stored along the lines of a corresponding policy.

Requesting and Issuing a certificate

The request- and issuance-process has been briefly outlined along the principle of a PKI shown in Figure 2.10, which superficially comprises the user generating a key-pair and requesting a certificate, which is signed by the RA and then used by

[26] Thawte Inc. `https://www.thawte.com/`

the CA to issue a certificate. In the following the certification issuance procedure is described in further detail:

1. **Key-pair generation**: The first action to apply for a certificate is to generate a RSA key-pair using a certain software such as OpenSSL [27]. An RSA-keypair comprises a public and private key and their generation follows the RSA-algorithm [*Rivest et al.*, 1978]. The public key can be distributed to everyone and if used for encryption the encrypted message can only be decrypted using the corresponding private key. If the private key is used for encryption, the message can only be decrypted using the public key, whereas this simplified process is referred to as signing a message.

2. **Creation of a CSR**: After the creation of the RSA-keypair a certificate signing request (CSR) has to be created. The CSR links subject information attributes to the public key of the RSA-keypair (c.f. second half of the data-section in the sample certificate in Listing 2.1). The CSR is then signed with the public key and transfered to the registration authority (RA). The details of this transfer are subject to the applied operational procedure which is defined in the PKI documentation. The transfer is either organized online using a secure transfer protocol such as https, or an offline transfer is performed by exchanging a physical data media such as a memory stick.

3. **RA validation and signing**: Assuming that the registration authority (RA) got the certificate signing request in the defined fashion, the first action of the RA is to verify the CSR. This includes the verification of the signature with the public key of the requester as well as a check if the subject information attributes are correct. Again, this could either be an online process just checking if the subject information data is complete, or if a human acts as RA officer, the subject information attributes are compared against the date of an official ID card. Usually the RA officer knows the requesters as they are located in the same organization, so the verification of the subject information data is self-evident. Finally the RA confirms the correctness of the CSR by signing it with the RA private key and transfers the RA-signed CSR to the certificate authority (CA).

4. **CA certificate issuance**: The transfer of the RA-signed CSR from the registration authority to the certificate authority is usually performed online, which is also detailed in the PKI documentation. The certificate authority performs a CSR verification on each new CSR it receives, in particular, a verification of the signatures, in order to ensure its authenticity. If the authenticity can be proofed the CA creates a new certificate by signing the CSR with its private key. The newly created certificate is then transfered to the requester, which should then again verify the certificate before actively using it.

[27] OpenSSL is an open source software that implements SSL. http://www.openssl.org/

The operational procedure which determines all details of the certificate issuance process is defined in the PKI documentation, which is usually distributed and maintained by the CA and comprises a certificate policy (CP), a certificate practice statement (CPS) and a policy disclosure statement (PDS).

Besides the components and issuance of certificates in a PKI, the establishment of trust among the participants of a PKI plays an important role, which is discussed subsequently.

Trust models

A main foundation of utilizing digital certificates within a public key infrastructure is trust. The exact establishment and level of trust is defined by a trust model. The trust model defines how trust between an issuer of a certificate and the one that verifies a certificate is established and maintained. Usually, the used trust model of a PKI is subject to its application domain, but according to [*Linn*, 2000] the following hierarchical and hybrid trust models as well as decentralized approaches [*Zimmermann*, 1995] exist:

- Hierarchical PKI
- Cross certification and bridging
- Web of trust

Hierarchical PKI: A hierarchical public key infrastructure requires a root certification authority, which is trusted by all participants. In the real world, i.e. on the global scale of the Internet, such a central root certification authority does not exist. Thus countries, international companies or projects run hierarchical PKIs with their own root certification authority. The background is mostly that these PKI operators want to entirely control the rules in their PKI (e.g. issuance of a certificate) rather then trust another certification authority.

Cross certification and bridging: One option to use certificates via multiple hierarchical PKIs is cross-certification. Two certification authorities (mostly root CAs) issue each other a (cross-)certificate in order to establish a trust relation between all participants of both PKIs. The general problem of cross-certification is its scaling behavior, because the number of cross-certificates required increases by square with the number of root certificates. A potential solution is to introduce a neutral bridge-CA, which exchanges cross-certificates with all participating CAs. Given this bridge-CA all certificates of one PKI can establish a trust relation to certificates of another PKI via their CA's cross-certificates. The bridge-CA represents the center of trust in this model and hence it is also known as hub-and-spoke trust model.

Web of trust: The web of trust represents a flat non-hierarchical concept to ensure authenticity of the binding between a public key and the user. Alternatively to hierarchical trust models, which rely on one or more certification authorities, this trust model is decentralized by each user having its own network of other trusted users, which also trusts others, a.s.o. The result is a web of trust with a user and an associated certificate being a part of, or a link between multiple webs. This was firstly formulated by Phil Zimmermann, the creator of the pretty good privacy (PGP) system [*Zimmermann*, 1995].

2.4.3 Transport Layer Security

Transport Layer Security (TLS) is the successor of Secure Socket Layer (SSL) both representing cryptographic protocols to enable secure communication in TCP/IP-based networks. TLS and SSL are utilized in a number of applications such as for web browsing, remote shell access, e-mailing or instant messaging. The protocols encrypt the datagrams of the transport layer in order to establish an end-to-end secure connection across the network. The most recent specification has been published by the IETF in the RFC 5246 [TLS].

Generally the TLS protocol has been designed to prevent tampering, eavesdropping, and message forgery as well as to enable endpoint authentication and communication confidentiality using cryptographic methods. The authentication is usually only provided for the server (i.e. only the identity of the server is ensured) and the client remains unauthenticated, just to make sure with whom the client is communicating. If both communication parties should be authenticated TLS provides a mechanism for mutual authentication, which requires a public key infrastructure (PKI) in place unless transport layer security with pre-shared keys (TLS-PSK) [*Eronen and Tschofenig*, 2005] or secure remote password (SRP) [*Taylor et al.*, 2007] protocols are used.

TLS handshake

The TLS handshake describes the process that two communication parties are performing to establish a SSL/TLS-connection. Basically the TLS handshake comprises three phases, starting with a negotiation for the algorithm support, continuing with the key exchange and the authentication respectively; finally the symmetric cipher encryption as well as the message authentication are preformed.

1. The TLS handshake starts with a **negotiation phase** which comprises the selection of the used algorithms including the cipher suits, the key exchange- and authentication algorithms as well as the message authentication codes (MACs).

2. The second phase is typically covered by public key algorithms unless pre-shared keys are used (i.e. TLS-PSK protocol) to **exchange a symmetric or shared key** and to ensure the authenticity of at least the server.

3. The final phase of the TLS handshake includes a message exchange between the communication parties, whereas the message is authenticated and encrypted using the public keys of the certificates as well as symmetric cipher encryption.

A more detailed explanation of the TLS handshake can be found in the TLS specification [TLS], which exactly defines which messages (esp. their content) are exchanged and which algorithms and standards are applied.

Mutual authentication

Usually authentication is only provided for the server while using TLS to communicate (e.g. if a client browses via https-based URLs). In this case the client remains unauthenticated and the server is authenticated just to let the client be sure with whom the communication is established. Authenticating both communication parties requires a public key infrastructure (PKI) in place to issue certificates for both parties. Assuming both certificates are issued by the same certification authority (CA) the corresponding client- and server-software has to be setup in a way, that it accepts the certificates from the common CA.

If a client is not privileged with an appropriate certificate, the SSL-connection attempt (handshake) will fail. TLS with mutual authentication is often used in closed systems, where the number of clients is limited and a high degree of protection of the servers is required. This principle is also known as applying best practice security on the transport layer.

In summary, this section presented an overview to security technologies and mechanisms related to Web and Grid computing. The security facilities as introduced in this section are further used and applied in the context of the developed Grid environment and its security infrastructure, which constitutes a major contribution towards establishing best practice security in Grid environments.

2.5 Summary

This chapter outlined the basic technologies required for the further on presented work in this thesis. It presented a wide range of diverse topics including Web services, Grid computing, Quality of Service and security. Web services are introduced as an essential realization of a service-orientated architecture (SOA). Grid computing has been outlined with a focus on its evolution and specifically with respect to architectures. Finally, Quality of Service levels have been discussed and basic security aspects have been presented.

Chapter 3

Grid Environment

This chapter encompasses a comprehensive description of the basic Grid environment which is also known and published as *Vienna Grid Environment* (VGE). It constitutes an integral part of the work performed in the context of this thesis and forms the basis of the QoS support presented in the subsequent chapters. The Vienna Grid Environment is a software framework to expose native high performance computing (HPC) applications and distributed heterogeneous data sources as Grid services.

The description mainly focuses on technical aspects such as the architecture, infrastructure and implementation, which is also reflected in the structure of this chapter. Initially a brief historical outline as well as a short overview are presented in order to catch the scope of the Vienna Grid Environment in general and particularly for this thesis.

The main content of this chapter has been initially published in [*Benkner et al.*, 2004b], followed by more specific emphasized work towards Quality of Service for Grid computing in [*Benkner et al.*, 2005c; *Benkner and Engelbrecht*, 2006], applying Grid component frameworks in [*Schmidt et al.*, 2005a, b, 2007] and Grid workflows in [*Brandic et al.*, 2005a, b]. Most recently VGE has been extended to provision also distributed heterogeneous data sources as services, which has been presented in [*Benkner et al.*, 2008].

Brief history

The Vienna Grid Environment is being developed at the Institute of Scientific Computing, University of Vienna. The developments started 2003 in the context of Grid-enabling medical HPC applications of the EU GEMSS project [*Jones et al.*, 2004] as well as compute-intensive parallel applications of the AURORA project[1] [*Koch et al.*, 2003], [*Benkner et al.*, 2005d].

The major initial design decision made for VGE was to develop the system in a way to enable native HPC applications from science and engineering to be exposed as services, i.e. realize a service oriented architecture. The second, maybe even more important decision was to base all developments entirely on standard Web services. Both have proven to be forward-looking, especially due to the proprietary Grid software existing at that time, such as the collection of tools in the initial version of the Globus toolkit or the Unicore system.

These basic decisions for the design of the VGE enabled the system to be applied in Grid projects, such as GEMSS and Aneurist as described later in this thesis as well as in other cooperations and research activities like CPAMSS[2] [*Ruckenbauer et al.*, 2007] or AMADEUS[3] [*Brandic et al.*, 2008]. Moreover, VGE has been extended in order to virtualize beside native HPC applications also scientific data sources as Grid services in order to address data access and integration in the Grid via according data services.

It should be noted that access to distributed heterogeneous data sources in the Grid and especially the capabilities of VGE in this context are beyond the scope of this thesis. The work performed with respect to data access, integration and semantic mediation can be found in [*Kumpf et al.*, 2007].

Brief overview

The Vienna Grid Environment is a service-oriented Grid infrastructure based on standard Web services technologies. Concerning a service provider, who wants to offer an application or data source, VGE enables the provision of native HPC applications or data sources as Grid services in an automated way. For the client application developer VGE offers a high-level client API to hide the complexity of the Grid and to simplify the construction of Grid client applications based on Grid services.

A key distinguishing feature of VGE is the support for Quality of Service. VGE provides a flexible QoS negotiation model to enable clients to dynamically negotiate certain QoS guarantees such as execution time and price with potential service providers on a case-by-case basis. The QoS approach including security will be discussed in depth separately as a further major contribution of this work in the next chapters.

This chapter is organized as follows: The VGE architecture is outlined in the beginning, followed by the VGE access model. Subsequently, introductions to the client and service infrastructure are presented, which include details about the underlying

[1] Aurora, `http://www.vcpc.univie.ac.at/aurora/`

[2] CPAMMS, `http://cpamms.univie.ac.at/`

[3] Amadeus, `http://www.infosys.tuwien.ac.at/Staff/ivona/amadeus/`

service component model. Finally, the VGE provisioning is highlighted again in the context of the client- and service-side.

3.1 Architecture

The Vienna Grid Environment (VGE) employs a service oriented architecture to provide native HPC applications and distributed heterogeneous data sources as Grid services. The high-level architecture comprises multiple Grid services, clients and registries, following the publish-find-bind principle introduced in Section 2.1 and generally applied in service oriented systems. In order to address security one or more certificate authorities (CAs) are typically adduced in the context of the architecture as well, even if a CA is usually not considered as a component of a service oriented architecture. The mentioned components of the VGE high-level architecture are depicted in Figure 3.1.

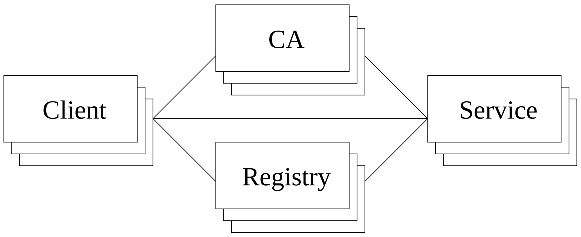


Figure 3.1. VGE high level architecture

Services: VGE services encapsulate applications or data resources as services distinguishing application and data services, respectively. Native HPC applications available on PC clusters or other high performance computing hardware are incorporated and accessed via application services, while distributed heterogeneous data sources are exposed by data services. The functionality of all VGE services is realized by a set of generic service components, including implementations for the management of application-jobs and data-queries, as well as for data transfer and staging, error recovery, Quality of Service, and others. Each service component provides an individual service interface with an according WSDL document that is also included in the composite WSDL of the entire VGE service. Details about the service infrastructure and the service component model will be discussed in Section 3.3.

Clients: VGE clients are usually applications that run on PCs or workstations as well as on mobile devices. These client applications utilize high-level client APIs provided for different platforms by the VGE middleware to access VGE services via the Internet. In order to give the client application developer more freedom but still hide the complexity of the Grid, the client APIs support different local abstractions of remote VGE services. Furthermore, client applications are responsible for the input data generation or preparation, the QoS negotiation if necessary, as well as the post-processing of the output data of a service. Details about the client infrastructure will be discussed in Section 3.4.

Registries: VGE registries are comparable to basic information services. Registries in a VGE Grid are used to keep information about service providers and the actual services they provide as well as to supply this information to clients upon request. The underlying data consists of a set of arbitrary attributes (name/value pairs) associated with a specific service. Such a facility enables service providers to publish their services as well as clients to perform queries for potential services. Clients typically use registries in the context of QoS negotiation to retrieve a list of candidate services to negotiate with.

Certificate authorities: The foundation of security are digital certificates, which are issued by an according certificate authority (CA) in an operational public key infrastructure (PKI). At least one such CA is required to be trusted and privileged to issue certificates in a VGE Grid. The certificates follow the X.509 standard as introduced in Section 2.4 and enable authentication and authorization for clients and services. Furthermore, certificates are used in the context of transport and message layer security.

A VGE Grid

As already mentioned VGE distinguishes between application and data services. Application services expose typically compute intensive native applications, while data services provide access to data sources. Usually a VGE Grid, as shown in Figure 3.2, comprises a number of both types of services, multiple clients, as well as one or more service registries.

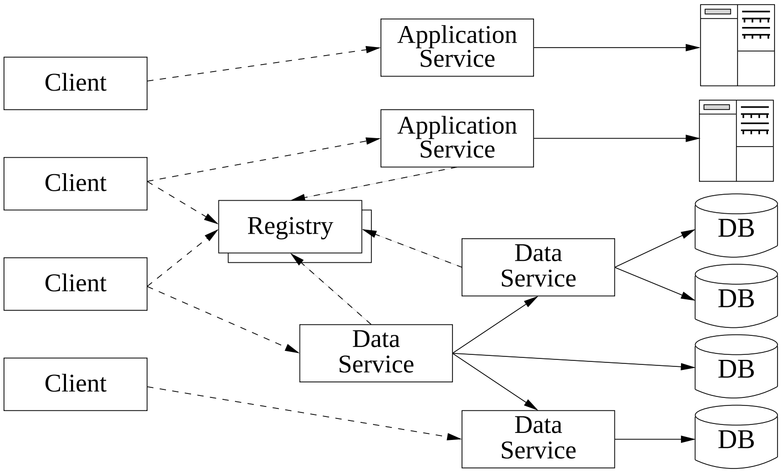


Figure 3.2. VGE Grid

Furthermore, Figure 3.2 illustrates that application services are tightly coupled to a specific machine and application. Data services are bound to one or more data sources, whereas a data source is either a relational database, an XML DB, a flat file or another data service. The tight coupling of applications to services and data sources to services is depicted by a continuous line as opposed to the loose coupling of clients with services and/or registries by a dashed line. Finally, another detail is revealed by Figure 3.2: Service discovery with registries is optional and if used services have to publish themselves in order to be found by clients.

Layered VGE architecture

The architecture of the Vienna Grid Environment is presented in a layered fashion in Figure 3.3. Similar to other Grid architectures resources located at the bottom are transparently accessed through middleware layers by applications at the top.

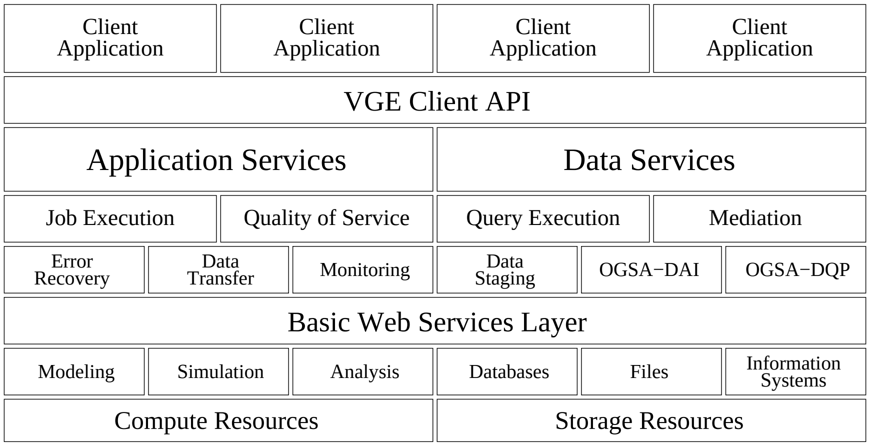


Figure 3.3. Layered VGE architecture

In a top down approach the layered abstraction of the VGE architecture shown in Figure 3.3 can by summarized as follows: Client applications are on top and use the VGE client API to access application and data services. Both types of services are composed out of a number of components realizing certain functionality but all are based on common Web services technology. Application services are incorporating typically native HPC codes dealing with simulation, analysis and modeling that run on high-end computing resources. Data services are exposing data sources such as databases, files or other information systems which all rely on storage resources.

It should be again noted that the emphasis of this work is rather depicted by the left side of Figure 3.3 and, in particular, by the capabilities of the application services to enable access to remote HPC applications on-demand and their support for Quality of Service. Data access and integration (DAI), mediation and distributed query processing (DQP) are beyond the scope of this thesis. More information, especially with respect to the entire architecture, can be found in [*Benkner et al.*, 2008].

3.2 Service access model

VGE services may be accessed following a flexible, multi-phase access model comprising an initial selection phase, an administrative phase, a Quality of Service phase and final job execution phase. The service access model also comprising further details of each phase is depicted in Figure 3.4.

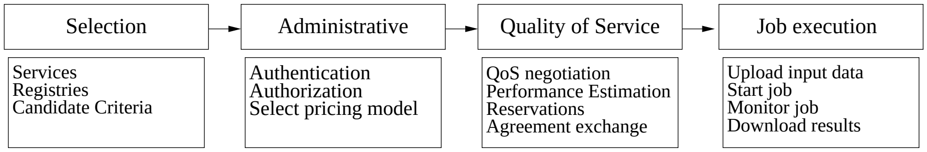


Figure 3.4. Multi-phase VGE service access model

Subsequently each phase in the VGE service access model is outlined briefly.

1. The **selection phase** constitutes the actual pre-selection of one or more candidate services. The client opts out a specific service (category) and therefore, a list of services is required or one or more registries with service attributes has to be supplied to retrieve a list of service candidates. In any case the resulting candidate services are used in the next phase.

2. The **administrative phase** comprises security and pricing issues. The security includes authentication and authorization, i.e. the client has to be authenticated and authorized to access a certain service including authorization on the client- and service-side. Furthermore, an agreement on the used pricing model has to be established, which just means that a specific pricing is selected and will be applied later in the QoS phase. The support for flexible pricing models is explicitly being discussed in Section 5.2.

3. The **QoS phase** consists of all QoS-related tasks. This basically includes a QoS negotiation between the client and the candidate services. At each service estimations and reservations are performed and possibly, in case of a successful negotiation an exchange of an agreement. From the client's perspective the negotiation is typically performed with the candidate services retrieved from the selection phase, while the exchange of an agreement is only conducted with a single service. The details of this phase are comprehensively described in Chapter 5.

4. The **job execution phase** includes all tasks required to perform a remote execution of an application job with a single service the client has eventually reached an agreement with. The job execution comprises uploading of input data, starting the execution of the applications job, and finally downloading the results.

VGE relies on a purely client driven service access, i.e. all interactions of a client with a service are set up by the client via SOAP. This also implies that e.g. the status of a remote job is queried iteratively (i.e. polling). Consequently, neither call-backs nor notifications are required, which usually result in security compromises to tunnel holes through site firewalls. Only a single port for SOAP communication via HTTP (usually port 80) has to be open, which is mostly the case with service providers running web servers. Thus VGE can be considered as firewall-friendly.

3.3 Service infrastructure

The VGE service infrastructure comprises a generic service provision framework, which enables service providers to set up data and application services in a semi-automatic way. A VGE service exposes either a compute-intensive application usually available on an HPC system or distributed data sources. Services are hosted in an according hosting environment and are securely accessed on-demand by clients over the Internet.

3.3.1 Component model

VGE services adhere to a generic and open service component model for the service composition. Consequently, a VGE service is composed out of generic service components which can be selected and configured at the time of the service setup. Furthermore, each component realizes a certain functionality and thus, services can be set up with the desired or required functionality only. Generally, the service components are provided for data transfers between clients and services, data staging between services, job or query execution, monitoring, QoS management, and error recovery.

WSRF influence

Conceptually the VGE service component model follows the Web Services Resources Framework model, which states the following [*Banks*, 2006]:

> *"A generic and open framework for modeling and accessing stateful resources using Web services including mechanisms to describe views on the 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."*

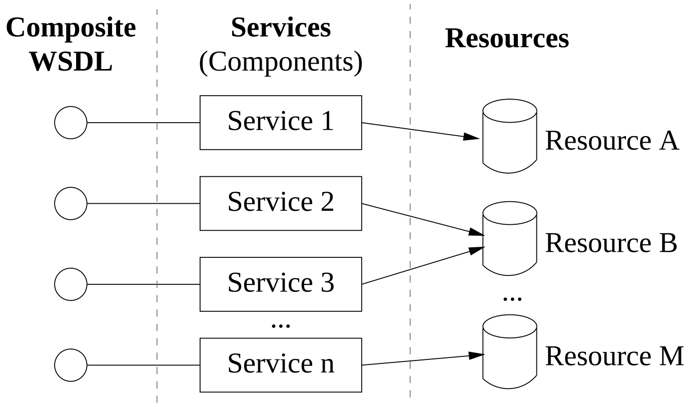


Figure 3.5. VGE service component model

The VGE service component model is illustrated in Figure 3.5, which indicates that each service component represents an abstraction of a resource. A single resource may be virtualized by more than one service component in different ways, but one service component is only associated with a single resource. This implies that the state of a service component is determined by a single resource only. Furthermore, Figure 3.5 shows that the entire service is composed out of service components each providing a separate WSDL document (interface specification). Thus, the overall capabilities and operations of the VGE services are defined by the composite WSDL, which is actually the union of all component WSDLs.

Component framework

In order to organize the communication between service components, the VGE service component framework is a mandatory part of each VGE service. It provides mechanisms to directly invoke operations of other service components utilizing local method invocations instead of processing the entire SOAP stack. Furthermore, the component framework supports listening and redirecting of client SOAP calls using according request and response handlers. The listening capability allows to keep service components informed of operation invocations on other service components. Redirections enable SOAP calls to be intercepted and to synchronously perform additional tasks. A typical consumer of the listening support is the monitoring facility, which records all invoked operations. An example for utilizing redirections is the support for Quality of Service, which checks if certain operations such as up- and downloads are performed according to the exchanged agreement.

Another main functionality of the VGE service component framework constitutes uniform session management for all service components. In oder to support multiple clients accessing the same service, the component framework provides session management capabilities to distinguish among different clients utilizing technologies such as WS-addressing. New sessions are automatically created for new clients or upon request and old sessions are being released if not required anymore. Note that session management is usually required by all service components and consequently is an essential part of a service.

3.3.2 Service components

The integral part of the service component model are, literally, service components. Each individual service component provides separate operations via its own WSDL document and implements a certain capability of the overall VGE service. Service components are provided by VGE for the following capabilities:

- **Application execution** to manage the execution of application jobs.
- **Data transfer** to transfer files between a client and a service.
- **Data staging** to facilitate direct data transfers between services.
- **Data streaming** to allow streaming of output data using just http (no SOAP).
- **QoS management** to dynamically negotiate certain qualities of a service.
- **Error recovery** to enable check-pointing and restarting of an application.
- **Query execution** to perform queries on virtualized data sources.

In accordance to the WSRF model, each service component is linked to a resource and furthermore provides a certain abstraction of its associated resource. These resources include disk space (directories), native applications, scheduling systems, data bases and others, similar to the resources presented in layered VGE architecture in Figure 3.3 at the bottom layers.

The overall VGE service is a composition of service components, which are selected and configured at the time of deployment. The minimal required service components for a VGE service comprise the application execution and the data transfer (or staging) service component, in case of an application service, or, the query execution and data transfer (or staging) service component in case of a data service. Subsequently all mentioned service components, their relations and provided operations are outlined briefly.

Application execution

The application execution service component provides the facility for the management of remote application jobs and, in particular, their execution. The associated resource of this service component is a pre-installed native application, which is capable of being executed in the working directory via a queuing or scheduling system. This implies that the application can be started via according scripts and all file access is performed in a relative manner (i.e. no fully qualified pathnames).

The application execution is being made secure by its design, because opposed to other Grid environments like Globus no applications or scripts are submitted to a service. The native applications, being resources in this context, are pre-installed by the service providers, which have full control over the scripts and applications that are executed on their systems.

The operations provided by the application execution service component are defined as follows and include: starting the execution of an application job, monitoring the status of a running job and killing a job if it does not terminate as expected.

```
start() : void
getStatus() : data
kill() : void
```

These operations are accessible via the service component's WSDL. An invocation of the *start* operation executes the configured start script in the defined working directory. The operation *getStatus* similarly invokes a script, but this time it is a status script and the operation returns arbitrary status information to the client, which is again subject to be configured e.g. as a status file. Finally, the operation *kill* initiates the execution of a pre-configured kill-script which terminates an application job.

All configuration information of a VGE service is stored in its service description, which comprises the XML application descriptor containing a reference to the mandatory start script, as well as optionally to a status-script, a status-file and a kill-script. More details about the service description will be discussed in Section 3.5.

Data transfer

The data transfer service component enables the transmission of input data to the service (upload) and output data back to the client (download). The actual input and output data is stored or retrieved from the so-called *working directory*, which actually constitutes the resource of this service component. The access to the working directory occurs transparently to the user and is under full control of the service provider. Usually, for each client a separate sub-directory is generated to distinguish among clients. Furthermore, clients do not have access to other parts of the service provider's file system.

The corresponding operations provided by this service component are *upload* and *download*, which are publicly accessible via WSDL and defined as follows:

```
upload(filename, input-data) : void
download(filename) : output-data
```

Usually the input- and output-data is transfered using files identified in the working directory by their filenames as specified by the operation's arguments. It should be noted that these filenames may be different from the client-side and again are subject to configuration of the service provider. Moreover, the transfered files are typically archives (e.g. zip or rar) in order to minimize the volume of the data and the number of SOAP calls to process by transferring mostly just a single input and a single output file. Multiple up- and downloads are supported as well e.g. in order to download intermediate results or to refine the input while running interactive applications.

The actual data transfers are performed by utilizing SOAP with attachments (SwA) as introduced in Section 2.1. This also facilitates data transfers of volumes up to two gigabytes for a single file. Generally, it should be noted that data transfers utilizing SwA compared to common mechanisms such as FTP may incur considerable performance drawbacks. More details about data transfers and their performance issues are discussed in [*Benkner et al.*, 2003a].

Data staging

The data staging service component supports pushing and pulling of input and output data between two services. Opposed to the data transfer service component, which passively stores and returns data upon request, this service component actively establishes connections to other services, but again only upon a client request. The data staging service component provides operations to push or pull data from (or to) its local working directory to (or from) another service by using the data transfer service component (upload or download) of this other service. This implies that the data staging service component is required only on the service that initiates the data staging (push or pull), while the data transfer service component has to be available on the destination of the staging.

Internally, this service component also uses the working directory on the local file system of the service provider as a resource and again the access to this directory is performed transparently to the client and with full configured surveillance of the service provider.

The actual operations of this service component are literally *push* and *pull*, which are publicly accessible via WSDL and defined as follows:

```
push(final-EPR, final-filename, staging-filename) : void
pull(final-EPR, final-filename, staging-filename) : void
```

The arguments of both operations are identical, but internally different tasks are performed. Commonly the *final-** arguments refer to the service the data is finally pushed to or pulled from, while the *staging-filename* parameter refers to the service initiating the data staging. The *final-EPR* (endpoint reference) comprises a URL and a session identifier, while all filename parameters are used to identify the files on the final service and the staging service.

In case of the *push* operation, the data staging service component uses the file identified by the *staging filename* argument from its local working directory and uploads it to the destination service. Contrarily, the *pull* operation performs a download from the service identified with the *final-EPR* to the staging service using the given filenames. Please note that both operations do not have any in- or output-data as arguments, because the actual data transfer occurs only between the staging and the final service.

Data streaming

The data streaming service component allows streaming of output data using just http in order to provide an alternative output data download method. The main purpose in this context is to improve performance as well as provide a direct download option to be used in Web applications. It should be mentioned that this comes with the drawback of bypassing the SOAP security, as direct downloads via http or https do not use SOAP and its associated security mechanisms.

Alike the data transfer and data staging service component, this service component internally uses the working directory as a resource with equal access constraints, i.e. transparent access for clients and fully customizable by the service provider.

Opposed to the other data transmission service components, the actual transmission of data provided by this service component is performed directly on the transport layer using a http or https channel. Therefore, the client requests a URL reference to the output data via a regular SOAP operation. The returned URL points to a servlet, which is part of this component, and also comprises an identifier of the output data. The URL can only be used once and thereof is also referred to as a one-shot URL. Given this URL the client is then able to download the output data directly via http or https only.

The corresponding operation provided by this service component is literally *download* and defined as follows:

```
download(filename) : output-data-url
```

The operation returns an URL, which actually comprises the address of the streaming servlet and an immutable universally unique identifier (UUID) [*Leach et al.*, 2005] as link to the actual data. The servlet enforces that the URL is just used once by removing the link (UUID) on the first access. Generally, accessing this URL occurs in addition to the service operations provided via SOAP and just relies on the actual transport protocol specified in the URL (http or https). Hence it should be noted that the security mechanisms applied with this kind of data transmission is limited to the transport layer's capabilities only.

QoS management

The QoS management service component realizes the Quality of Service support. In particular this service component provides a high-level QoS negotiation interface to clients and utilizes a scheduling system as its resource. The QoS management adopts a reservation-based approach; thus, the underlying scheduling system has to support advance reservation.

The operations provided by the QoS management service component are defined as follows:

```
requestQoSOffer(qos-request, request-descriptor) : qos-offer
cancelQoSOffer(qos-offer) : void
confirmQoSOffer(qos-offer) : void
```

From an abstract point of view these operations enable a client to negotiate certain QoS attributes by requesting an offer based on certain constraints to cancel received offers or to confirm a satisfying offer. The Quality of Service support will not be detailed any further in this chapter, because the entire Chapter 5 is devoted to this key distinguishing feature.

Error recovery

The error recovery service component enables support for application-level recovery from errors using checkpointing and restarting. If a native application is able to be re-started from a certain checkpoint, i.e. the application supports error recovery, this service component provides appropriate operations to utilize this application functionality by clients.

The operations defined in the WSDL of the error recovery service component comprise the following:

```
checkpointUpload(filename, checkpoint-data) : void
checkpointDownload(filename) : checkpoint-data
restart() : void
```

The client can use these operations to upload or download checkpointing data using an optional filename to specify a particular checkpoint. The *checkpointDownload* is typically used upon a crash of an application job and the *checkpointUpload* is usually used to migrate the checkpointed job to another service on a different machine. The *restart* operation in fact restarts the application.

The actual data transmission linked to the *checkpointUpload* and *checkpointDownload* operations is realized by utilizing the data transfer service component. Consequently, this also comes with the implications of the data transfer service component, mainly by relying on SOAP with attachments (SwA).

Query execution

The query execution service component is essentially required for data services. It provides capabilities to access different data sources such as relational databases, XML databases or flat files. The operations required therefore, are identical to the ones of the application execution service component. This implies that a data service has the same access pattern as an application service by starting a query with the *start* operation, monitoring the query execution with the *getStatus* operation or killing the query-execution with the *kill* operation.

The configuration of the query execution service component includes how many data sources are utilized, which public schema is exposed in case of data mediation as well as which additional evaluation hosts are used by the distributed query processing (DQP) support. As initially mentioned, the data access, integration and mediation capabilities of VGE are beyond the scope of this work and the corresponding query execution service component is outlined for the sake of completeness only.

Additional components

The VGE service infrastructure adopts, as mentioned in the beginning of this section, an open component model approach. Consequently, new service components can be realized and integrated to extend the functionality of VGE services. Similar to the WSRF there are no specific requirements for the implementation of an additional service component, except that it has to be implemented as Web service.

Subsequently, some of remaining the components are outlined briefly. Compared to the described service components so far, these are less extensive in terms of their functionality, their provided operations in the WSDL or their associated resource.

The **monitoring** service component provides an operation (*getInfo*) to retrieve general status information about a service. This mainly comprises static service information such as a name and other facts about the underlying data or application resource, as well as dynamic information such as the service utilization.

The **Web portal** service component provides an administrative Web interface for the service provider, mainly to adapt minor configuration details, examine deployment issues or just check the logs. The information gained from the Web interface can be retrieved by WSDL operations as well.

Security service components are typically realized as SOAP handlers implementing a certain security facility (c.f. Section 2.4.1). Typically, these service components use security credentials as their resources. More details about security and in particular its SOAP handlers will be discussed in Section 5.5.

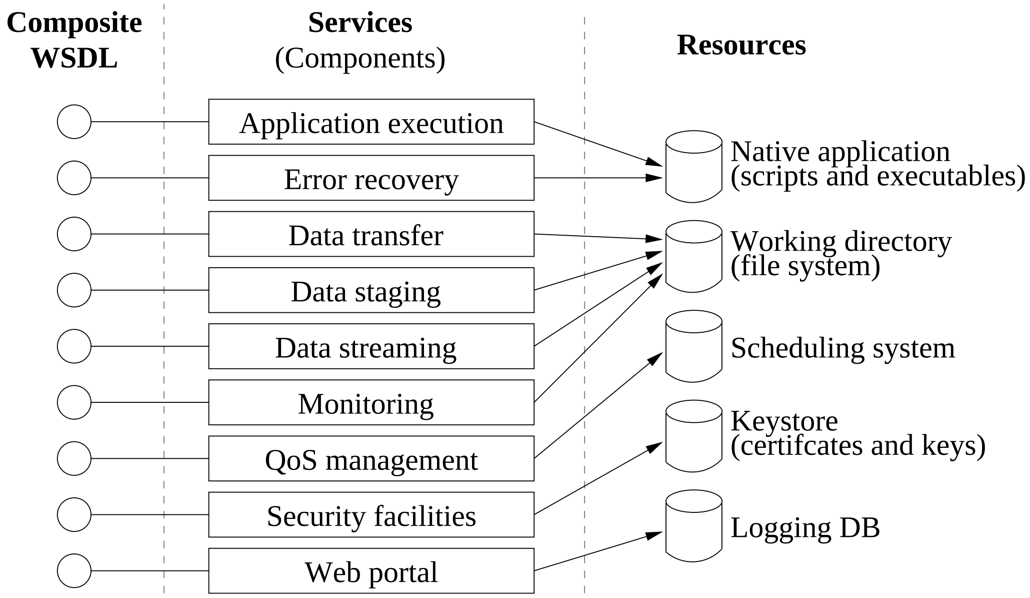


Figure 3.6. Sample VGE service components

The illustration in Figure 3.6 shows a complex application service with all mentioned service components (except obviously the query execution component) as well as their associated resources.

3.3.3 Service hosting

VGE services are composed of service components which are embedded in the VGE service component framework, which utilizes Apache Axis[4] as Web services framework. Furthermore, Apache Tomcat[5] serves as a Web application hosting environment. The VGE software distributions comes with a preconfigured hosting environment, an integrated Axis version and, additionally, a graphical deployment tool

[4] Apache Axis, `http://ws.apache.org/axis/`

[5] Apache Tomcat, `http://tomcat.apache.org/`

to configure and deploy VGE services conveniently. The actual provisioning process is detailed in Section 3.5.

A typical VGE service hosting scenario is illustrated in Figure 3.7 and comprises a data service and an application service.

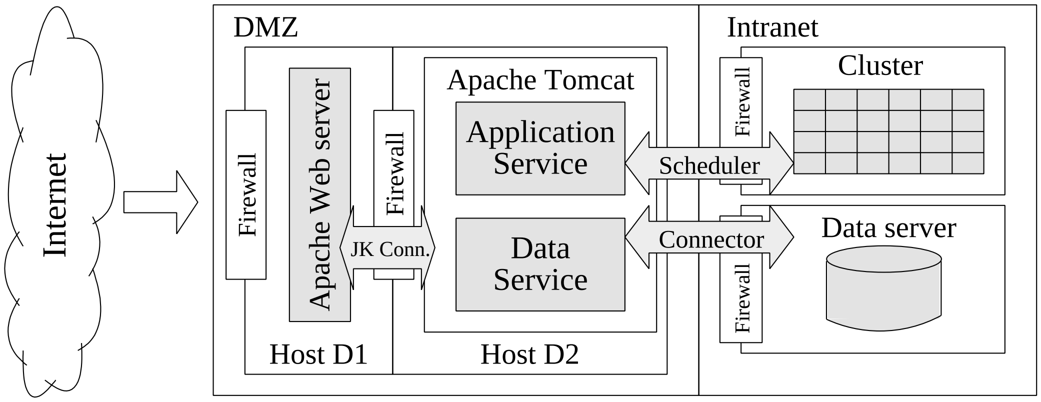


Figure 3.7. VGE service hosting

The data and application services are hosted in an appropriate service hosting container such as Apache Tomcat, which is typically installed on a host in the demilitarized zone (DMZ) of the service provider's organization. Optionally, on a second host in the DMZ, an Apache web server is used as an entrance point from the Internet to access the VGE services via a Tomcat connector[6] between the Apache web server and the Tomcat server. The actual resources, which are located in the Intranet or another private network, are utilized by the services through an according facility, such as a scheduling system or database connector.

3.4 Client infrastructure

The companion piece to the VGE service infrastructure is the VGE client infrastructure. It constitutes a framework to simplify the development of Grid client applications that access and interact with VGE services. The client infrastructure supports the development of complex standalone applications as well as the integration in existing applications by relying on implementations and toolkits in several programming languages.

Basically, client applications could be built entirely from scratch by only utilizing the composite WSDL description of a VGE service, but this would constitute access to VGE services on the lowest possible level. Furthermore, such a low-level interaction

[6] Apache Tomcat connector, `http://tomcat.apache.org/connectors-doc/`

with Web services is rather time-consuming with respect to its development, because the client developer has to handle all the Web and Grid service details. Finally such an approach typically results in complex and error-prone client applications. In order to address this obstacle, the VGE client infrastructure provides different programming toolkits for several programming languages such as Java, C# and C++. This enables the client developer to access VGE service based on these toolkits and hides the complexities of Grid and Web service technologies.

The VGE client infrastructure consists of the following main building blocks:

- Libraries
- Development environment
- Documentation
- Samples

The libraries are provided for several programming languages, such as Java, C++ or C# (.Net), whereas the Java implementation is used in the majority of cases. The development environment supports the client developer to easily compile and execute new code by having all the necessary libraries included and configuration options already set accordingly. Finally, the documentation comprises detailed instructions about the examples as well as guidelines for integration in existing applications.

3.4.1 High-level Client API

The construction of Grid client applications which access VGE data and/or application services is supported by a high-level client API with bindings for Java, C# and C++. The main purpose of the client API is to hide the details of interacting with remote Grid services and their associated data sources or native applications.

Figure 3.8 depicts a layered view of the client API. The foundation of the client API constitutes a transport and messaging protocol layer, which provides mechanisms to generate and process (secured) messages according to protocols such as SOAP and http. Built upon these low-level APIs, support for different programming language is provided again by different levels of abstraction, i.e. Stub-level, Proxy-level and Agent-level abstraction. Each abstraction consists of a set of classes to deal with different aspects of the VGE middleware.

The agent-level forms the top abstraction layer, which comprises distinct components that perform a sequence of remote invocations to complete an entire task in an autonomous fashion and thus are referred to as agents. Agents are provided to automatically process tasks such as the execution of an application job, to negotiate QoS or to run a query at a remote data service. The proxy level abstraction consists

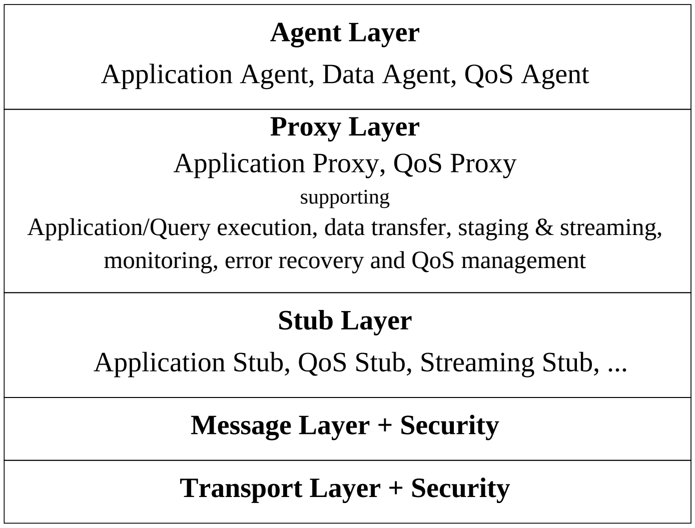

Agent Layer
Application Agent, Data Agent, QoS Agent
Proxy Layer
Application Proxy, QoS Proxy
supporting
Application/Query execution, data transfer, staging & streaming,
monitoring, error recovery and QoS management
Stub Layer
Application Stub, QoS Stub, Streaming Stub, ...
Message Layer + Security
Transport Layer + Security

Figure 3.8. VGE Client API

of an application and a QoS proxy. The Application-Proxy supports operations to manage the application or query execution, to initiate data transfers or staging, to do monitoring and to enable error recovery. The QoS proxy exposes operations to perform a QoS negotiation. Opposed to agents, which execute their tasks automatically, the proxies provide different operations and, as a consequence, interactions with the user are supported. Finally, the Stub layer includes an application and a QoS stub comprising the low level methods of the remote services including the details of security and ID handling, which is hidden on the layers above.

3.4.2 Sample client

Given the different abstraction layers of the VGE client API, various methods for discovering and selecting services are supported. The simplest method constitutes the selection of a specific service by using its address (URL). More sophisticated methods including service discovery via registries or performing a QoS negotiation are provided by according agents and proxies. The service discovery requires to specify a set of service attributes (e.g. an application category) which is then used to query a VGE registry. As a result, the client retrieves a set of services, which match the supplied attributes. The most sophisticated service selection is performed within a QoS negotiation process, which is detailed in Chapter 5.

Listing 3.1 comprises a simple excerpt of a Java code, which connects to an application service and uploads input data, starts the remote applications, polls the status and finally downloads the results. For the sake of simplicity this example is kept very simple and details, as for example the error handling, are set aside.

```
// Creates new application proxy based on certain properties
Properties props = new Properties();
props.setProperty("client.job.uri","http://localhost:9090/HelloWorld/scs/");
props.setProperty("client.base.proxy.exception", "true");
AppProxy someProxy = new AppProxyImpl(props);

// Uploads a local file to the remote service using the proxy
someProxy.upload(new File("upload.dat"), "inputABC");

// Start the remotely deployed and configured, native application
someProxy.start();

// Polling the status of the application
AppProxyState proxyStatus = AppProxyState.ERROR;
String appStatus = null;

while (proxyStatus != AppProxyState.FINISHED) {

        // Obtains status information of the remote job
        // and prints out proxy and app status
        appStatus = someProxy.getApplicationStatus();
        System.out.println("----->␣App␣status␣info:␣" + appStatus);
        System.out.println("----->␣Proxy␣status␣info:␣" + proxyStatus);
}
// Creates filehandler for downloading the result
DataHandler data = (DataHandler) someProxy.download("output.dat");

File outFile = new File("output.dat");
FileOutputStream out = new FileOutputStream(outFile);
data.writeTo(out);
```

Listing 3.1. Sample VGE client application

An example that connects to a data service would look similar, except for the fact that the input data comprises a query to the data source and the output would contain the query result.

3.4.3 Additional features

The VGE system is used in miscellaneous contexts and thereof additional features have been developed with respect to certain environments such as programming language, interface or platform. In the following some of these additional features are outlined briefly.

A **command-line client** has been realized in order to be used in script-based client applications (also Web applications realized in PHP or Perl). Each operation of the VGE services can be invoked separately using different command line parameters. The output of each operation is stored in appropriate files to be further processed.

In contrast to the low-level command-line client, a high-level **Web-based client** has been implemented based on Java servlets and Java server pages (JSP). This client interface comprises a graphical user interface (GUI) in a Web browser, which supports the management of multiple jobs remotely, i.e. the jobs are handled persistently in this Web application, rather than in a local client application. Basically the GUI supports "drag'n'drop"-style user interaction to submit inputs, start jobs, obtain status information and download outputs.

Finally, other client bindings should be mentioned as well. The .NET client API realized in C# enables even Office applications to utilize VGE Grid services in corresponding macros. Typically, in spreadsheet processing (e.g. with Excel), as often used in financial applications, compute intensive algorithms may be utilized via VGE services. Also the integration of VGE services in C++ applications is supported by utilizing the gSOAP Web services framework.

In summary, different VGE client interfaces exist to enable client application developers to make use of VGE services in a convenient way.

3.5 Service provisioning

Modern information technology employs the term *provisioning* to describe the initialization of a part or an entire infrastructure and all involved tasks therein. This usually comprises configuration, setup and instantiation of the required systems, and the organization of all sorts of security related issues, in particular, user access to resources.

The VGE service provisioning refers to the overall process of initializing a VGE service starting with the construction of the service and its service component selection to the final instantiation of the service in its hosting environment. In order to support this process, the VGE service infrastructure comprises an according provisioning environment including the VGE deployment tool. This graphical tool automates the service provisioning as much as possible by guiding the user through the entire service setup procedure. Furthermore, the graphical user interface of the VGE deployment tool enables service providers to conveniently customize and setup VGE services, without being burdened with the details of Grid and Web technology.

Basically, service provisioning is structured into a configuration and a deployment phase. The configuration comprises the selection and customization of the service components as well as the configuration of the targeted hosting environment. The deployment consists of the service preparation, its incorporation in the hosting environment and finally the instantiation of the service. The latter, in particular the entering of a prepared service in the application container, is also commonly referred to as technical deployment, which is usually supported by appropriate tools of the respective hosting environment.

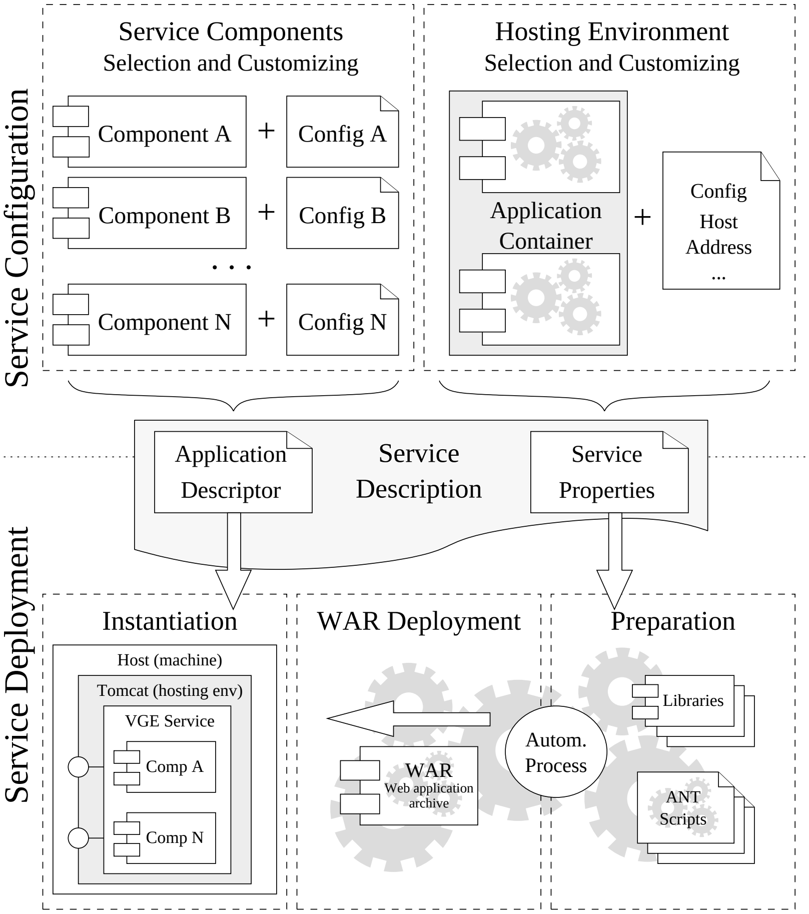


Figure 3.9. VGE service provisioning

A comprehensive graphical representation of the service provisioning process is captured in Figure 3.9. The figure also reveals further details of the configuration and a deployment phase which are discussed subsequently.

The **service configuration** initially comprises the selection of the required functionality of the service, which determines the required service components. Each service component has to be configured with respect to its used resource, e.g. the native application has to be specified. The second step of the configuration includes the specification of the hosting environment for the newly constructed service by definition of the host address or the location of helper-tools. The deployment tool stores the entire configuration information supplied by the user in the service description. Internally the service description distinguishes between application- or data-related information required at runtime and service properties required for the preparation and deployment.

The actual **service deployment** is performed fully automatically, as soon as all required configuration information has been made available and stored in the application- or data descriptor and service properties. The first part of the deployment is preparation, which actually creates a deployable Web application archive (WAR). This rather extensive process is performed by ANT-scripts[7] and comprises the creation of the directory structure and the various configuration files, the selection of the required libraries as well as, eventually, the construction of an according Web application archive. The technical deployment of the Web application archive relies on helper tools of the Web application container which also trigger the final instantiation of the service.

In the following the service configuration and deployment are discussed.

3.5.1 Configuration

The service configuration is performed individually for each service and constitutes the first phase of the overall provisioning process, which is depicted in the upper part of Figure 3.9. The configuration mainly consists of the selection and customizing of the used service components (i.e. service construction) as well as the targeted hosting environment.

The output of the service configuration phase is the service description, which comprises all necessary information required in the service deployment phase. In particular the configuration of the service components is stored in the application descriptor (or data descriptor in case of a data service), while the hosting environment configuration feeds into the service properties, respectively.

VGE services are configured in a static fashion which implies that changes of the configuration can only be applied in the context of a re-deployment of the service. But the entire configuration information is stored in the service description, which can be loaded in the deployment tool on re-deployment, which also eases the configuration of similar services.

Subsequently, the service configuration is further detailed comprising the service component configuration and the hosting environment configuration.

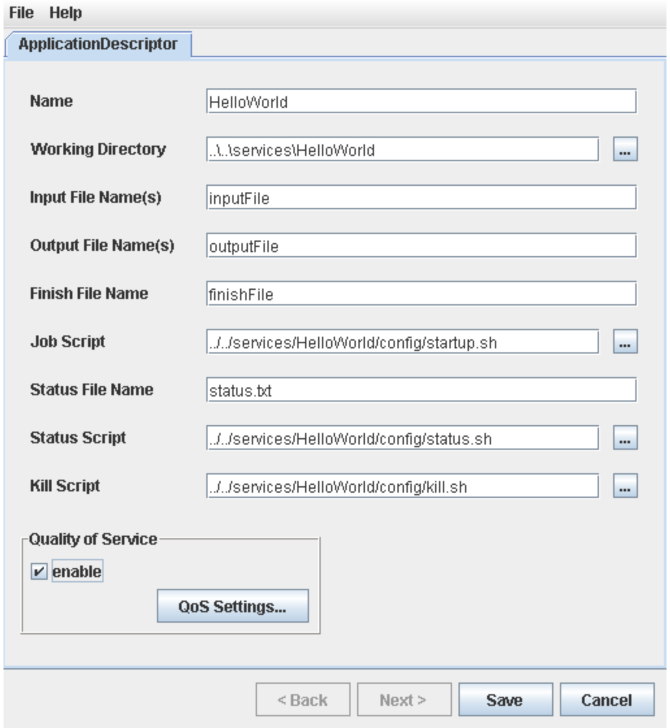
Service component configuration

The service component configuration comprises the selection and customization of the service components. The corresponding information is eventually stored in the application or data descriptor. The minimal specification of service components for a VGE application or data service requires the following service components:

Application service = *Application execution* + *data transfer* OR *staging*

Data service = *Query execution* + *data transfer* OR *staging*

[7] Apache Ant is a Java-based build tool, `http://ant.apache.org/`



Name: HelloWorld

Working Directory: ..\..\services\HelloWorld

Input File Name(s): inputFile

Output File Name(s): outputFile

Finish File Name: finishFile

Job Script: ../../services/HelloWorld/config/startup.sh

Status File Name: status.txt

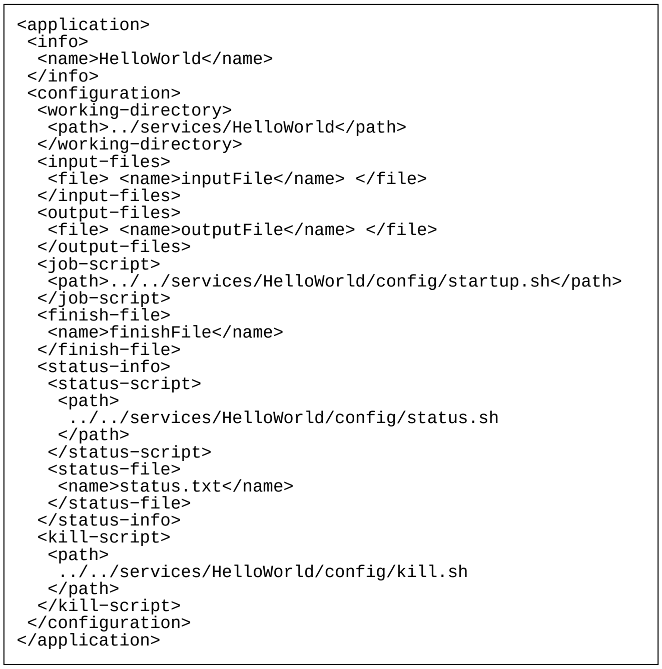
Status Script: ../../services/HelloWorld/config/status.sh

Kill Script: ../../services/HelloWorld/config/kill.sh

Quality of Service

☑ enable

(a) VGE Deploymenttool



```
<application>
 <info>
  <name>HelloWorld</name>
 </info>
 <configuration>
  <working-directory>
   <path>../services/HelloWorld</path>
  </working-directory>
  <input-files>
   <file> <name>inputFile</name> </file>
  </input-files>
  <output-files>
   <file> <name>outputFile</name> </file>
  </output-files>
  <job-script>
   <path>../../services/HelloWorld/config/startup.sh</path>
  </job-script>
  <finish-file>
   <name>finishFile</name>
  </finish-file>
  <status-info>
   <status-script>
    <path>
     ../../services/HelloWorld/config/status.sh
    </path>
   </status-script>
   <status-file>
    <name>status.txt</name>
   </status-file>
  </status-info>
  <kill-script>
   <path>
    ../../services/HelloWorld/config/kill.sh
   </path>
  </kill-script>
 </configuration>
</application>
```

(b) Application descriptor

Figure 3.10. VGE service component configuration

An according minimal application descriptor usually comprises the specification of filenames or patterns for input-, output-, and status-files. Moreover, scripts have to be defined in order to start and optionally kill the job execution as well as to obtain status information. The corresponding data descriptor of a data service contains information about the underlying data sources.

Figure 3.10(a) depicts a screenshot of the deployment tool showing all mentioned scripts and files related to an exposed native application. This information is finally stored in the application descriptor exemplified in Figure 3.10(b).

Hosting environment configuration

The hosting environment configuration comprises the selection and customization of the targeted application container, in which the VGE service should be hosted. This information is supplied by the user via the deployment tool and stored in the service properties. The minimal specification of the hosting environment requires the host address (URL) and user credentials, which grant authorization for the deployment procedure.

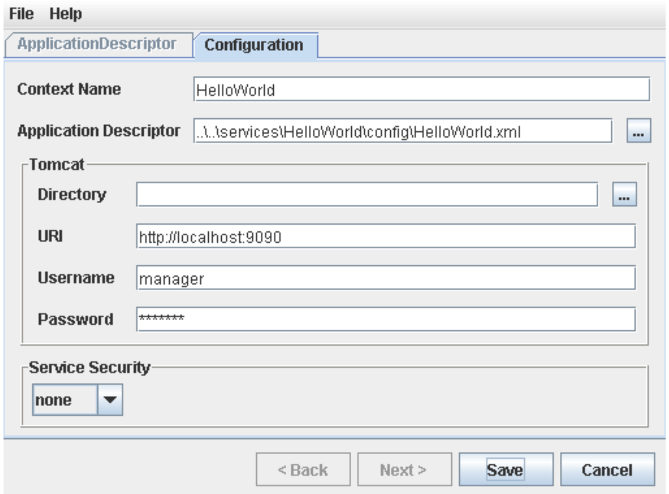
Figure 3.11(a) depicts a screenshot of the deployment tool showing the minimum specification of the targeted hosting environment (Tomcat). This information is stored in the service properties exemplified in Figure 3.11(b).

3.5.2 Deployment

In contrast to the service configuration, the service deployment is a fully automatic process, which finally results in a VGE service that is being setup and instantiated in a running Web application container (hosting environment). The bottom half of Figure 3.9 depicts the structure of the service deployment with three phases: the preparation of the Web application archive (WAR), the actual WAR-deployment and the instantiation of the VGE service. In the following these phases are further detailed.

Preparation

The preparation phase covers the entire creation process of the Web application archive (WAR) and is performed automatically by a set of ANT-scripts. The WAR-creation process is being customized by the service properties, which have been created as a result of the service configuration. The Web application archive comprises all necessary libraries and configuration files a VGE service requires. The preparation process (i.e. the ANT scripts) firstly generates all configuration files and, secondly, incorporates the configuration files and libraries as determined by the selected service components in the WAR-file.



Context Name: HelloWorld

Application Descriptor: ..\..\services\HelloWorld\config\HelloWorld.xml

Tomcat

Directory:

URI: http://localhost:9090

Username: manager

Password: *******

Service Security: none

(a) VGE Deploymenttool

```
#HelloWorld sample service
service.name=HelloWorld
service.application.descriptor=
  ../../HelloWorld/config/HelloWorld.xml
service.config.dir=../../HelloWorld/config
service.tomcat.dir=
service.protocol=http
service.host=localhost
service.port=9090
service.manager.user=manager
service.manager.pwd=manager
service.remote.dist=
service.components=ApplicationService
```

(b) Service properties

Figure 3.11. VGE hosting environment configuration

The following configuration files have to be generated:

- Web application deployment descriptor (web.xml)
- Web service deployment descriptor (server-config.wsdd)
- Web service definition language (WSDL) documents for each component
- Web service-internal properties

All these files are created automatically according to the information supplied in the service configuration and do not have to be written manually as with other Web services frameworks.

The second part of the preparation phase is the actual buildup of the WAR-file including the incorporation of the generated configuration files and the libraries. The file-structure of the Web application archives follows the servlet specification[8], which also determines the exact location of all included files. The included libraries contain external third party developments such as the used Web services framework Apache Axis as well as the internal VGE libraries for each service component. Finally, the created archive constitutes a deployable WAR-file.

WAR deployment

The actual Web-deployment of a WAR-file is usually performed by according helper-tools of the targeted Web application container. In case of VGE Apache Tomcat is used as default Web application hosting container, mainly because it is a free open-source Java hosting platform. Due to the flexible configuration of VGE other hosting environments such as JBoss[9] are supported as well. The only requirement of the hosting environment is the support for hot deployment and undeployment of Web applications (i.e. WAR-files) at runtime. The rationale of a hot deployment approach is not to interfere with existing services running in the same container as opposed to cold deployment, which requires a restart of the application container in order to deploy new services.

Tomcat supports the live-incorporation of Web applications and encapsulates this underlying complex process within its manager servlet, which is utilized by VGE through ANT. More precisely, in order to deploy and also to undeploy or re-deploy VGE services in running Tomcat servers, the provisioning environment comprises ANT-scripts which contact the manager servlet of Tomcat to initiate the WAR-deployment. The underlying complex procedure with the service entering the application container and being made accessible is performed entirely automatically and transparently to the user.

The main benefit of the WAR-file approach being used in VGE is the hot-deployment possibility of the entire Web application which embeds the Web service implementation and the Web service framework (WS-framework). Opposed to this mechanism, Web services frameworks usually provide their own mechanisms to deploy and undeploy services, which rely on their own implementations (such as the Axis-Admin servlet). This usually comes with the drawback of less scalability due to all services running in a single Web application and the dependency on a particular WS-framework. VGE is also open and customizable with respect to the applied WS-framework. Apache Axis is being used by default, but other WS-frameworks such as Sun's JAX-WS-RI[10] are supported as well.

[8] http://jcp.org/aboutJava/communityprocess/final/jsr154/index.html

[9] JBoss, http://www.jboss.org/

[10] JAX-WS-RI, Java API for XML Web services - reference implementation, https://jax-ws.dev.java.net/

Instantiation

The service instantiation is also an important characteristic of the VGE services in comparison to other Web and Grid services. Usually Web services are instantiated upon the first request which, in case of complex services, results in delayed responses upon the first usage of a particular service. The instantiation of VGE services is triggered by the deployment and consequently, there are no delays due to instantiation upon the first request. This feature is especially important to enable elasticity of the required services, i.e. if additional services are deployed upon heavy utilization of other services and being stressed right after the deployment.

The instantiation of the VGE services is initiated internally by a Web application listener, which is being informed of a deployment event by the application container. The listener then simulates an internal status request, which requires all necessary libraries and classes to be loaded and the configuration to be processed. This also implies that potential misconfigurations or error-setups are being detected upon deployment before the first actual client request arrives.

In summary, service provisioning is a complex process, comprising an interactive configuration phase and an automatic deployment phase. Each phase is furthermore structured into several steps, which are shown in Figure 3.9. The configuration phase comprises the service component and hosting environment customization, while the deployment phase is structured by the preparation, the WAR deployment and the instantiation of the service.

3.6 Client provisioning

The VGE client infrastructure, as outlined in Section 3.4.1, comprises a high-level client API for the development of individual Grid client applications. Besides this development approach, VGE provides a generic Web-based Grid client application to manage remote jobs via VGE services. This Web-client is also subject to an according provisioning as described in the following.

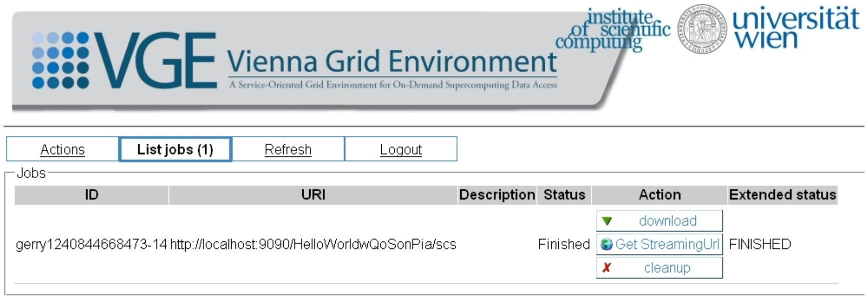
The VGE Web client provisioning refers to the entire process performed in a VGE client environment to set up and initialize a generic Web-based Grid client application. The main purpose of this generic Grid client application is to enable users to interact with VGE services and manage their jobs accordingly without installation, integration or even development of client software. The VGE Web client can be accessed by a Web browser and provides all necessary operations to manage remote jobs running in VGE services.

In order to set up such a generic Web client in the VGE client environment, an according Web application hosting environment such as Tomcat, as well as a corresponding configuration are required. The deployment process is performed automatically by ANT-scripts in a similar fashion as presented for the service-side. It comprises the following basic steps:

1. Creation of the required configuration (e.g. Web application deployment descriptor) and properties
2. Buildup of the client Web application archive (client-WAR) including all necessary configuration and Java libraries.
3. Deployment of the client WAR-file in the hosting environment

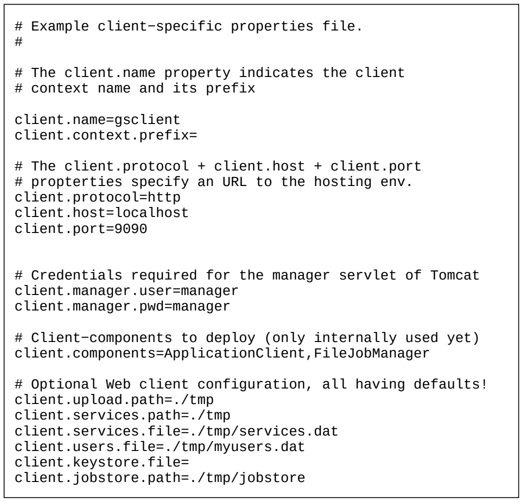
The Web client provisioning follows a structure similar to the VGE service provisioning, except that it is less complicated.

Figure 3.12(a) shows a sample screenshot of the Web client and the according configuration is exemplified in Figure 3.12(b).


VGE Vienna Grid Environment
A Service-Oriented Grid Environment for On-Demand Supercomputing Data Access
institute of scientific computing
universität wien
Actions
List jobs (1)
Refresh
Logout
Jobs
download
Get StreamingUrl
cleanup

ID	URI	Description	Status	Action	Extended status
gerry1240844668473-14	http://localhost:9090/HelloWorldwQoSonPia/scs		Finished	download Get StreamingUrl cleanup	FINISHED

(a) VGE Web client



```
# Example client-specific properties file.
#

# The client.name property indicates the client
# context name and its prefix

client.name=gsclient
client.context.prefix=

# The client.protocol + client.host + client.port
# properties specify an URL to the hosting env.
client.protocol=http
client.host=localhost
client.port=9090


# Credentials required for the manager servlet of Tomcat
client.manager.user=manager
client.manager.pwd=manager

# Client-components to deploy (only internally used yet)
client.components=ApplicationClient,FileJobManager

# Optional Web client configuration, all having defaults!
client.upload.path=./tmp
client.services.path=./tmp
client.services.file=./tmp/services.dat
client.users.file=./tmp/myusers.dat
client.keystore.file=
client.jobstore.path=./tmp/jobstore
```

(b) Client properties

Figure 3.12. VGE client configuration

In contrast to the VGE service provisioning, the Web client configuration is less complex and does not require a separate deployment tool. Moreover, the configuration of such a Web client is usually performed by experienced administrators which actually prefer editing text-files and executing ANT-scripts compared to entering information in a graphical user interface.

3.7 Summary

This chapter presented a comprehensive overview of the Vienna Grid Environment. VGE is a secure service oriented Grid environment for the on-demand provisioning of HPC applications and data sources as Grid services. The description comprised details of the architecture and the infrastructure on the client- and service-side. A particular emphasis has been put on the service component model and all relevant service components as well as on the complex provisioning and deployment process.

Chapter 4

QoS Model

This chapter describes the basic approach to model Quality of Service support for Web and Grid Services. The motivation for Quality of Service support as introduced in the previous chapters can be concluded from the requirements of specific application domains which make QoS support mandatory. A representative example is the health domain, where sensible compute-intensive applications aim to support patient-specific and time-critical medical procedures (e.g. surgery). These applications necessitate QoS support to be successfully applied in the daily practice of healthcare.

Initially an overview of the QoS support model is presented, including the basic requirements and the used terminology. Subsequently the QoS support model is being detailed with respect to the general capabilities of the QoS support as well as the QoS offer generation in response to a QoS request. In particular, the components of the QoS support model are identified and their interactions as well as the overall inputs and outputs are discussed.

Requirements

The basic requirements of the QoS support can be drawn from a motivating example-application in the health domain: a neurosurgery support application, which utilizes compute-intensive non-linear image registration methods to simulate the brain-shift phenomenon during surgery to coordinate surgical navigation. In order to use such a computationally demanding application in practice, the availability of sufficient resources to run the application and timely deliver results has to be guaranteed and contracted in advance. Sufficient resources are usually available on HPC platforms provided by supercomputing centers, which substantiate the requirement for exposing the native application as a service, being used remotely. Contracting the service usage is typically subject to a business context (i.e. clients pay for the service usage) and the contract is established by negotiating a service level agreement.

Given the mentioned representative neurosurgery support application from the health domain, the basic requirements of the QoS support are guarantees of the response time and price. The QoS support enables the establishment of a contract that specifies guarantees about an application execution in terms of QoS attributes for the response time and the price. The response time guarantees require according estimations and pre-booking of resources. Prices are subject to negotiation and flexible business models of service providers which expose native applications as services. As a consequence the QoS support has to provide the following capabilities:

- Negotiation of QoS attributes (e.g. response time and price).
- Establishment of an according agreement between a client and a service (e.g. SLA).
- Resource capacity estimation (e.g. runtime, CPUs, memory) in advance.
- Advance resource reservation (e.g. number of nodes/CPUs).
- Flexible pricing (e.g. pay-as-you-go).

Generally, the QoS support infrastructure has been designed for arbitrary QoS attributes (e.g. a certain trust level or a minimum availability level), but the focus of this work has been put on response time and price guarantees of HPC applications. This is also a major distinguishing characteristic to existing work in the QoS domain. Most related work such as [*Menasce and Casalicchio*, 2004] are concerned with rather basic Web applications which exhibit almost constant response times in the range of seconds. Contrarily, this work deals with heavily varying response times of HPC (simulation) applications in the range of minutes, hours and even days, dependent on the application-specific input parameters and utilized hardware infrastructure.

Terminology

The QoS model relies on similar notational conventions and terminology as used in existing standards and specifications such as the Web Services Level Agreement (WSLA) language specification [*Heiko et al.*, 2003] and the Web Services Agreement (WSA) specification [*Andrieux et al.*, 2007]. The general objective of both specifications is to provide a standard for service level agreements (SLAs) used with Web services. The concrete naming of many terms therein varies and thus, no common terminology can be concluded. As a consequence, this model defines and uses the following terms:

QoS parameters: A QoS parameter is defined as a QoS-relevant input parameter to the QoS model, which affects the performance of the underlying application.

Typically, QoS parameters comprise application-specific meta data to be used by the QoS model for an accurate prediction of the QoS capabilities. The concept of a QoS parameter is only barley defined in existing standards, but may be related to an *agreement creation constraint* of the WSA specification.

QoS attributes: A QoS attribute is defined as a guaranteed quality the service is capable to offer and it is determined by the QoS model. Opposed to QoS parameters, which are the input to the QoS model, the QoS attributes are outputs of the QoS model. For example, the service quality to assure may be the time or price of a certain computation and the concrete value may be 10 minutes or 5 €. In the WSLA specification a QoS attribute is referred to as *SLA parameter* and the concrete value is formulated using a *Service Level Objective*, while the QoS attribute equivalent in the WSA specification is named *guarantee* or *guarantee term*.

QoS capability model: The QoS capability model aims to determine a set of QoS attributes (i.e. concrete values of these attributes) and, in particular, a range of values for each QoS attribute that can be assured by the service, based on a set of concrete QoS parameter values.

QoS offer generation: The QoS offer generation defines the behavior of the QoS support when responding to the given client-inputs, which comprise QoS parameters and requested QoS attributes. The output of the offer generation process is a set of offered QoS attributes (i.e. values for these QoS attributes), that also fulfill the clients request. The interaction of the client and the service in the context of potentially repeated QoS offer generations is also referred to as QoS negotiation.

Neither the Web Services Level Agreement (WSLA) language specification nor the Web Services Agreement (WSA) specification detail the actual procedure how an agreement is established, i.e. how a potential negotiation is performed. As a consequence, no equivalents to the QoS capability model or the QoS offer generation exist in the WSLA or the WSA specifications.

4.1 QoS Capability Model

This section outlines the QoS capability model M, which aims to determine a set of QoS attributes that can be assured by the service, based on a set of QoS parameters. The QoS capability model is being presented along the lines of the more complex model introduced in [*Pattnik et al.*, 2003] in the context of autonomic computing. The QoS capability model facilitates a mapping of QoS parameters P to QoS attributes A, which is captured by the relation β as presented in 4.1.

$$M : \beta \subseteq P \times A \tag{4.1}$$

The relation β specifies valid input-output pairs or in other words: Valid input

QoS parameters to output QoS attribute mappings. In contrast to a function, the relation β permits an output variability which is very common to most IT-systems (i.e. for a given input $p \in P$ diverse mappings to A exist). This very high-level abstraction defines a generic functional behavior, which is also adhered to by the implementation of the QoS capability model as presented in the next chapter.

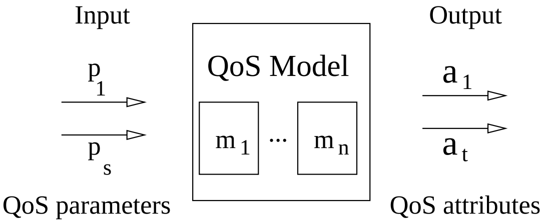


Figure 4.1. QoS Model Input-Output.

The inputs and outputs of the QoS model are also shown in Figure 4.1, which specifies QoS parameters as inputs and QoS attributes as outputs. Equivalently to Relation 4.1, Figure 4.1 shows an arbitrary number of QoS parameters (p_1 ... p_s) as input to the QoS model and an also arbitrary number of QoS attributes (a_1 ... a_t) as output. The QoS capability model internally relates these inputs to outputs utilizing an orchestration of QoS sub-models (m_1 ... m_n), which aim to reduce the overall complexity of the implementation as outlined in Chapter 5.

Subsequently, the QoS capability model is being exemplified, using sample QoS parameters and QoS attributes as well as QoS sub-models composed of mathematical functions.

QoS capability example

The example describes the QoS capability model in the context of an HPC application that solves a number of equations. The application is assumed to be able to utilize parallel compute nodes with an efficient scaling behavior. Two input QoS parameters (p_1 and p_2) are used: a resource measure (number of computing nodes, e.g. on a cluster) and a computation effort measure (number of equations to solve). The output of the QoS capability model in this example are two QoS attributes (a_1 and a_2) which represent the total computation time and the price for the computation. The QoS capability model itself comprises two sub-models (m_1 and m_2) which interdependently calculate the QoS attributes with mathematical formulas based on the QoS parameters.

Given this context an exact definition of the input QoS parameters (i.e. their value ranges) as well as the functional behavior of the QoS capability sub-models are outlined. Subsequently, the QoS capability model is applied to derive all capable sets of QoS attributes.

The example assumes the following definitions related to the input QoS parameters as well as two output QoS attributes:

$$\begin{array}{rcl} P & = & \{p_1,\ p_2\} \\ p_1 \in P_1 & \dots & \text{number of computing nodes} \\ p_2 \in P_2 & \dots & \text{number of equations to solve} \\ P_1 & = & \{2,\ 4,\ 8\} \\ P_2 & = & \{100,\ 200\} \\ \\ A & = & \{a_1,\ a_2\} \\ a_1 & \dots & \text{total computation time (in minutes)} \\ a_2 & \dots & \text{price of the computation (in Cent)} \end{array}$$

These definitions assume that the application can be executed utilizing 2, 4 and 8 computing nodes as well as that it is able to solve 100 and 200 computations. Besides the input QoS parameters, their ranges and the output QoS attributes, the QoS capability sub-models are defined with simple functions that eventually calculate (=predict) the runtime and price:

$$\begin{array}{rcl} M & = & \{m_1,\ m_2\} \\ m_1 & \dots & \text{computation time prediction model} \\ m_2 & \dots & \text{price prediction model} \\ m_1 & = & \dfrac{p_2}{\ln(p_1) + 1} \\ m_2 & = & p_1 m_1 \end{array}$$

The model functions assumed in this example represent simple functions for the computation time prediction and the price prediction. The computation time is assumed to scale with increasing resources in a logarithmic manner. The price calculation is based on a fixed pricing of the total amount of resources required (number of CPUs times calculation time). The interdependency of the models in this example is specified by the dependency of m_2 on the result of m_1, i.e. the price prediction is dependent on the computation time prediction.

In the next step all model input combinations P^c are calculated and based on this complete set of inputs the total set of capable QoS attributes A^c is computed. Please note that this illustrative example is not applicable for a general case with infinite sets and/or arbitrary models. For the sake of simplicity this example only deals with finite sets and simple mathematical models as follows:

$$\begin{aligned} P^c &= P_1 \times P_2 = \{\langle 2,100\rangle, \langle 2,200\rangle, \langle 4,100\rangle, \langle 4,200\rangle, \langle 8,100\rangle, \langle 8,200\rangle\} \\ A^c &= \{\langle 59,118\rangle, \langle 118,236\rangle, \langle 42,168\rangle, \langle 84,336\rangle, \langle 32,256\rangle, \langle 65,520\rangle\} \end{aligned}$$

The set of all model input combinations P^c is being calculated using all QoS parameter combinations ($P_1 \times P_2$), comprising a total of six different input combinations (e.g. 2 computing nodes and 100 equations to solve as represented in the first element of P^c).

Subsequently, the set of all capable QoS attributes A^c is being calculated based on the set of all model input combinations P^c. A^c comprises all QoS attribute tuples that the service is able to ensure. In particular, A^c contains six elements, each comprising a tuple of computation time and the price for the computation. E.g. the first element of A^c states that the computation can be performed within 59 minutes with a price of 1.18 €.

4.2 QoS Offer Generation

This section outlines the QoS offer generation performed by the service in response to a given client request. The input to this process is supplied by the client and comprises QoS parameters as well as *requested* QoS attributes. The output of the QoS offer generation is a set of *offered* QoS attributes. In other words, the offer generation process responds to a given client request (input) with an appropriate offer (output). The offered QoS attributes usually meet both conditions: the service is capable to ensure them and they fulfill the client's request.

The main objective of the QoS offer generation to provide certain QoS attributes that satisfy the client's request is accomplished by using basic set theory. More precisely, the QoS offer generation specifies that the offered QoS attributes are calculated by the intersection of the capabilities of the service and the requested QoS attributes of the client. The set of capabilities a service is able to ensure A^c is represented as set of n-tuples as defined in the previous Section 4.1. The set of requested attributes A^r is assumed to be represented also as set of n-tuples consisting of all QoS attributes tuples that fulfill the client's request.

The intersection of the client's request and the service's capabilities, as defined by the QoS offer generation to determine the offered QoS attributes, can be formulated as follows:

$$A^o = A^r \cap A^c \tag{4.2}$$

The equation 4.2 defines the intersection of the requested A^r and capable QoS attributes A^c resulting in a set of QoS attributes A^o that the service actually offers. Consequently, all offered QoS attribute tuples meet both: The request of the client and the capabilities of the service to ensure these QoS attributes.

The set of capable QoS attributes is obtained from the QoS capability model presented in Section 4.1, which indicates that the QoS offer generation is dependent on the QoS capability model, but not vice versa, i.e. the QoS capability model is independent from the QoS offer generation. The set of requested QoS attributes has to be supplied by the client in the same format as the QoS capability model provides the set of capable QoS attributes. The same format is required due to the intersection of the requested and capable QoS attributes.

The following example illustrates the approach by modeling the QoS attributes as distinct dimensions in a multidimensional space. Thus, a geometrical shape can be derived for both sets of QoS attributes (requested and capable). Figure 4.2 illustrates the approach with three dimensions (i.e. a set of three QoS attributes a_1 to a_3) for the requested and the capable QoS attributes. Each QoS attribute has a minimum and a maximum value, hence an according intersection of requested and capable QoS attributes can be derived in order to compute the offered QoS attributes. For the sake of simplicity all minimum values are set identically to 0.

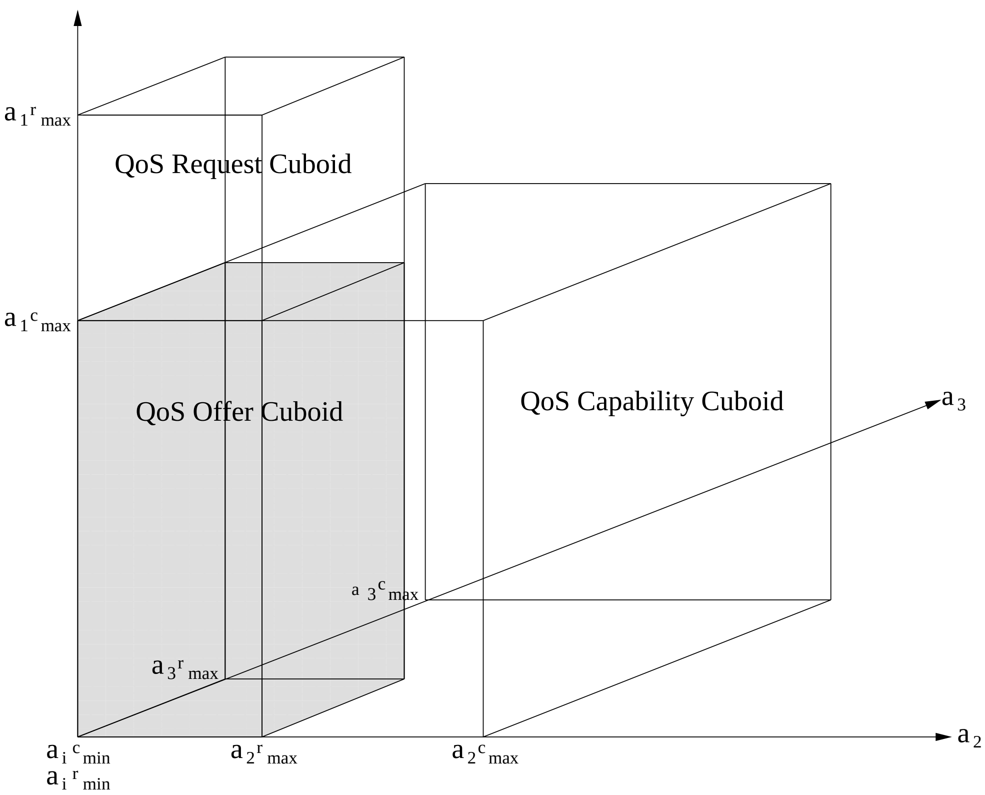


Figure 4.2. Multidimensional request and capability intersection.

This graphical example illustrates the set of all capable QoS attributes as QoS capability cuboid and the set of requested QoS attributes as QoS request cuboid. The QoS offer generation specifies that the intersection of these sets results in a set of offered QoS attributes, which is depicted by the QoS offer cuboid.

QoS offer generation example

The explicatory example started in Section 4.1 is continued here. The derived set of capable QoS attribute tuples A^c, which describe the total computation times and prices, have been taken over, while the client's request A^r is newly defined as follows: The total computation time must not exceed 60 minutes and the price must not exceed 2.5 € (= 250 Cents). Finally the corresponding intersection of A^c and A^r is computed and the resulting set of offered QoS attribute tuples A^o can be concluded as follows:

$$\begin{aligned}
A^c &= \{\langle 59, 118\rangle, \langle 118, 236\rangle, \langle 42, 168\rangle, \langle 84, 336\rangle, \langle 32, 256\rangle, \langle 65, 520\rangle\} \\
A^r &= \{\langle a_1^r, a_2^r\rangle : 0 \leq a_1^r \leq 60 \cap 0 \leq a_2^r \leq 250\} \\
\\
A^o &= \{\langle 59, 118\rangle, \langle 42, 168\rangle\}
\end{aligned}$$

This illustrative example shows that given the client's request[1] the QoS offer generation (i.e. the intersection of the capable and the requested sets of QoS attributes) can be applied to determine the offered QoS attributes.

[1] Note that the requested set of QoS attributes A^r actually comprises 15.311 tuples/combinations if a_i^r are natural numbers.

Chapter 5

QoS Support

This chapter describes the Quality of Service support proposed in this thesis. Support for Quality of Service is a basic requirement of specific application domains such as the health sector. Compute-intensive applications in the health domain target support for time-critical medical procedures (e.g. surgery), which make QoS support mandatory. This chapter addresses such applications and presents a comprehensive description of the Quality of Service support infrastructure and its realization. The main objective therein is to provide a proof of the concepts defined in the QoS support model as introduced in Chapter 4 and to demonstrate that Quality of Service can be established in a service-oriented environment.

The provision of native HPC applications as Grid services that also support Quality of Service accomplishes an added value for users and providers of these services. Grid application users benefit by performing a negotiation and agree on certain QoS guarantees in advance to the actual usage of a service on a case-by-case basis. Service providers achieve an additional business value by providing offers and potentially selling the usage of their services in a competitive market.

The QoS support addresses the special requirements of compute-intensive and time-critical HPC applications, which necessitate guarantees on the execution time and the price. Consequently, the focus besides security has been set on time and price guarantees, which are represented by distinct negotiable QoS attributes and associated with appropriate QoS models; hence, the implemented QoS support infrastructure comprises QoS models for time and price. Another important requirement is a facility to provide advance resource reservation in order to book a specific resource in advance.

This chapter has been initially derived from [*Benkner and Engelbrecht*, 2005, 2006] and from works focusing on generic QoS negotiation in the medical domain in [*Middleton et al.*, 2007; *Benkner et al.*, 2007].

This chapter is organized as follows: Initially, an overview of the basic QoS support scenario is presented by outlining the QoS support within a single service, which is referred to as *micro QoS management*, and the negotiation process between a client and one or more services, which is referred to as *macro QoS negotiation.* Then a more detailed description of the micro QoS management presents the framework for the QoS models which predicts certain QoS attributes, the required resource reservation capabilities as well as a discussion of the underlying QoS management strategies. Subsequently, the macro QoS negotiation is presented including details about different negotiation approaches. Finally, this chapter discusses the security infrastructure in the context of Quality of Service.

5.1 Overall Scenario

The overall QoS support scenario comprises one client interacting with one or more QoS-aware services in order to establish an agreement on certain qualities of a service. According to [*MacLaren*, 2003] and existing standards such as the Web Services Level Agreement (WSLA) language specification [*Heiko et al.*, 2003] and the Web Services Agreement (WSA) specification [*Andrieux et al.*, 2007] this interaction is defined as *QoS support establishment phase* and takes place in advance to the actual service usage. Being more precise, the actual QoS support establishment phase in this work is structured into a macro QoS negotiation, when a client interacts with one or more services by exchanging requests and offers, and a micro QoS management, when one service attempts to generate an offer in response to a client's request.

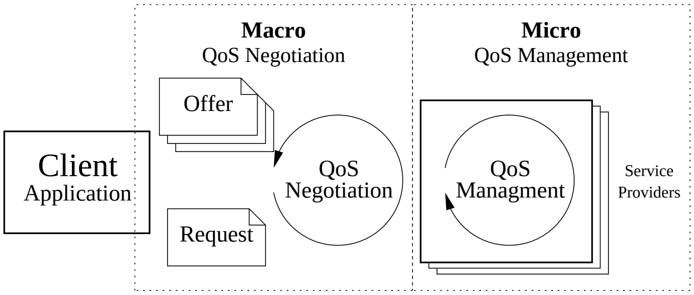


Figure 5.1. QoS support scenario

Figure 5.1 illustrates the big picture of the QoS support scenario. On the left-hand side the macro QoS negotiation is shown, where a client negotiates with one or more service providers by exchanging requests and offers and finally, establishes a QoS contract with one service provider. On the right-hand side the figure depicts the micro QoS management with each service provider individually attempting to satisfy the client's request.

In the following, this section briefly outlines the micro QoS management and macro QoS negotiation.

Micro QoS Management

In this thesis the process of generating a QoS offer upon a client's request is referred to as "microscopic" QoS management. A QoS-aware service internally relies on a set of QoS models[1] to provide a set of QoS guarantees, expressed by QoS attributes. The QoS support provides QoS attributes for the price of the application execution on a specific service as well as for the scheduled time of the application execution (i.e. begin- and end-time). In order to ensure the on time execution of the application the required resources have to be available and consequently pre-booked using a reservation-based resource management system.

The reservation-based approach distinguishes significantly from other best-effort- or priority-based approaches as targeted by most traditional QoS approaches [*Zheng*, 2001], because it guarantees the exclusive availability of the required resources in advance. As a consequence, an appropriate resource management system that supports advance reservation is required. Further details will be discussed in Section 5.2.

Macro QoS Negotiation

The process of requesting and generating offers with certain QoS guarantees is initiated by a client with one or more service providers and takes place in advance to the actual service usage. A client wishing to run an application has to choose a service provider which will run the job. Obviously, the selection process is dependent on the Quality of Service offered by each service. The QoS support infrastructure enables clients to negotiate with multiple services before they choose a single service provider to establish an agreement with. This negotiation process between a client and the service provider(s) is referred to as "macroscopic" QoS negotiation.

The negotiation approach opens a wide range of business options to clients and service providers. As soon as a client starts a negotiation with multiple service providers to run a specific job, a specific electronic market (e-market) is instantly created, where the market participants (service providers) contest against each other about selling a specific product, which in this case is the application execution for a specific job. The economic implications of such electronic markets (c.f. eBay[2]) are subject to ongoing intensive research [*Nissanoff*, 2006] and go far beyond the scope of this work, but the technical details of the implementation as well as a brief economic analysis are discussed in Section 5.4.

[1] QoS models in this chapter correspond to QoS sub-models introduced in Chapter 4.

[2] eBay, `http://www.ebay.com/`

5.2 Microscopic Quality of Service

This section presents the details of the microscopic Quality of Service management (Micro QoS), which are mainly derived from [*Benkner and Engelbrecht*, 2006]. As stated in the overview the micro QoS management concentrates on providing QoS attributes to ensure guarantees for the scheduled time of the application execution (i.e. begin- and end-time) and its price. Figure 5.2 illustrates the micro QoS management, in particular, the used QoS models and the QoS manager as well as the overall inputs and outputs.

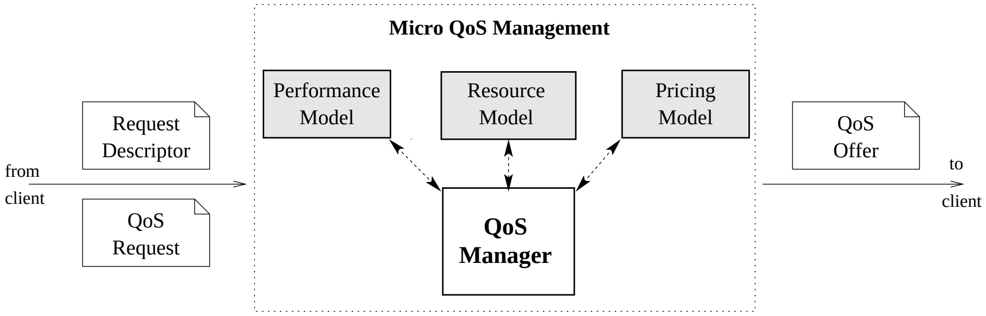


Figure 5.2. Micro QoS Management [*Benkner and Engelbrecht*, 2005]

The figure shows three QoS models for the estimation of the application's runtime, the resource allocation and the price calculation, which are depicted by the *performance model*, the *resource model* and the *pricing model*, respectively. The *resource model* interfaces a concrete resource management system to provide capabilities for advance resource reservation. These components are utilized by the central *QoS manager*, which organizes the orchestration and interaction of the QoS models.

The input and output of the micro QoS management, which is received from and delivered to a client, shows the request in form of an application-specific *request descriptor* and a *QoS request*. The request descriptor contains QoS parameters with a information about the application job, and the QoS request specifies the requested QoS attributes such as preferred execution time or maximum price. The client inputs are processed by the micro QoS management, which usually responds to the client with a *QoS offer* comprising appropriate QoS attributes as initially requested.

5.2.1 QoS Attributes, Parameters and Models

The micro QoS management relies on QoS models, QoS attributes and QoS parameters. Generally, QoS attributes and parameters are used as input and/or outputs to QoS models, while the composition of all QoS models (plus the QoS manager) constitute the overall micro QoS support. In the following all these QoS components (QoS attributes, QoS parameters and QoS models) are described in further detail.

QoS Attributes

The QoS support focuses on guaranteeing a certain execution time and price for a specific application. The actual execution time is defined with the begin- and end-time of a specific application job. Consequently, the actual execution time comprises two QoS attributes, which are subject to negotiation: begin-time and end-time. Usually a client specifies the begin-time request for a certain job as the earliest possible point in time, when the application can be launched. Typically this refers to the time when all necessary input-data is available. The end-time specifies the latest possible point in time, when the application should be finished and the results should be available in the client's working directory as introduced in Section 3.3. The price defines a specific maximum double value in a defined currency (e.g. €), which must not be exceeded.

Given the focus of the micro QoS management the following negotiable QoS attributes are supported:

- begin time (point in time)
- end time (point in time)
- price

The supported QoS attributes feed into XML-based documents, which are passed back and forth between client and service throughout the negotiation process. The QoS request and the QoS offer are defined as QoS descriptors following the Web Service Level Agreement (WSLA) specification by IBM [*Heiko et al.*, 2003].

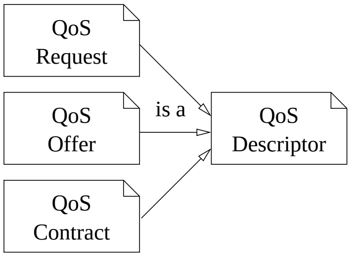
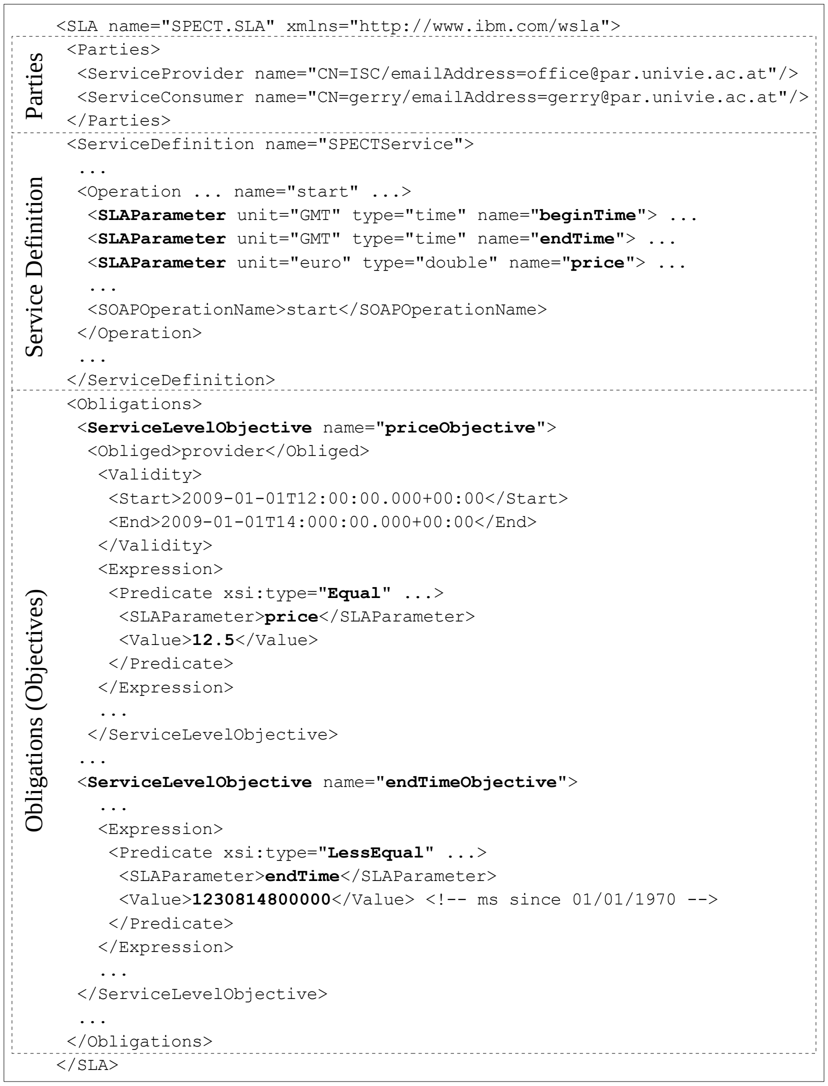


Figure 5.3. QoS Descriptors

A QoS descriptor represents a (potential) agreement on a single service usage between a service consumer (client) and a service provider. Depending on the state of a QoS negotiation, Figure 5.3 shows that a QoS descriptor is either a QoS request, a QoS offer, or a QoS contract.

Web Service Level Agreements: A QoS descriptor as exemplified in Figure 5.4 consists of three main blocks: parties, service definition, and obligations. The parties block comprises information about the signatory parties involved, the service definition block specifies the subject matter of the agreement (i.e. details of the service and SLA parameters) and the obligations block defines the objectives associated with each parameter. Subsequently, each block of a QoS descriptor is being described in further detail.



```
<SLA name="SPECT.SLA" xmlns="http://www.ibm.com/wsla">
 <Parties>
  <ServiceProvider name="CN=ISC/emailAddress=office@par.univie.ac.at"/>
  <ServiceConsumer name="CN=gerry/emailAddress=gerry@par.univie.ac.at"/>
 </Parties>
 <ServiceDefinition name="SPECTService">
  ...
  <Operation ... name="start" ...>
   <SLAParameter unit="GMT" type="time" name="beginTime"> ...
   <SLAParameter unit="GMT" type="time" name="endTime"> ...
   <SLAParameter unit="euro" type="double" name="price"> ...
   ...
   <SOAPOperationName>start</SOAPOperationName>
  </Operation>
  ...
 </ServiceDefinition>
 <Obligations>
  <ServiceLevelObjective name="priceObjective">
   <Obliged>provider</Obliged>
    <Validity>
     <Start>2009-01-01T12:00:00.000+00:00</Start>
     <End>2009-01-01T14:000:00.000+00:00</End>
    </Validity>
    <Expression>
     <Predicate xsi:type="Equal" ...>
      <SLAParameter>price</SLAParameter>
      <Value>12.5</Value>
     </Predicate>
    </Expression>
    ...
   </ServiceLevelObjective>
  ...
  <ServiceLevelObjective name="endTimeObjective">
    ...
    <Expression>
     <Predicate xsi:type="LessEqual" ...>
      <SLAParameter>endTime</SLAParameter>
      <Value>1230814800000</Value> <!-- ms since 01/01/1970 -->
     </Predicate>
    </Expression>
    ...
  </ServiceLevelObjective>
  ...
 </Obligations>
</SLA>
```

Parties
Service Definition
Obligations (Objectives)

Figure 5.4. QoS Descriptor example

The parties block defines all contracting parties, which are usually extracted from the security certificates of users and service providers (c.f. CN, common names in the example). The service definition block defines all operations subject to the agreement and a set of SLA parameters, which are defined through the supported QoS attributes. The example QoS descriptor refers to a service named *SPECTService* and defines three SLA parameters (*beginTime*, *endTime* and *price*) for the operation *start*. Furthermore, the service definition block specifies the overall contract duration as well as metrics and types for each parameter, which are neglected in the example. The last block is concerned with obligations, which actually comprises a list of objectives. Each objective is linked to an obliged party, has an according validity and defines an expression that is associated with a defined SLA parameter. In the example the SLA parameter *price* has to be *equal* to *12.5 €* and the *endTime* of the job execution must not exceed *01 Jan 2009, 14:00:00.000 GMT*, which corresponds to *1.230.814.800.000* milliseconds since *01 Jan 1970, 00:00:00.000 GMT*.

Further details about the Web Service Level Agreement can be found in its language specification [*Heiko et al.*, 2003].

QoS Parameters

QoS parameters comprise meta information about the application job and the used hardware. QoS parameters are used as inputs to the QoS models, which either have to be supplied by the client (application-specific information) or by the service provider (machine-related information). The respective QoS parameters are stored in different descriptors: The machine descriptor comprises all hardware-related information, while the request descriptor consists of application-specific meta information about a concrete job.

QoS Models

A QoS model is usually associated with one or more QoS attributes and supports the prediction of concrete values for these QoS attributes in advance. Moreover, a QoS model may also just predict an intermediate internal QoS argument which is then used as input to another QoS model. Such an internal QoS argument is neither a QoS parameter nor a QoS attribute and it is not visible outside the QoS support framework in the QoS offer. The QoS infrastructure utilizes three QoS models in order to predict the application's runtime (in particular the begin and end-time of a certain job) and the price. Therefore, the following QoS models are supported:

- Performance Model
- Resource Model
- Pricing Model

In a nutshell: the performance model predicts the application's runtime and corresponding resource requirements for a specific application job. The resource model enables the booking of resources in advance and the pricing model calculates the price of the specific service usage.

Implementations of these QoS models are dependent on the concrete application, the service provider's resource management and/or the business strategy. As a consequence, the micro QoS management encapsulates the implementations of these QoS models. The QoS manager just uses appropriate interfaces to interact with the QoS models.

5.2.2 Performance Model

The main purpose of the performance model is the estimation of the application's runtime and other machine-relevant indicators such as memory or disk space in advance to a potential job execution.

Modeling the performance of HPC applications is a complex undertaking, which goes far beyond the scope of this work. The QoS support introduced in this chapter is based on the assumption that it is possible to implement a performance model for a particular application in order to estimate the application's runtime for a specific job in advance to a potential job execution.

Performance Model Implementation

The QoS support infrastructure does not prescribe the actual nature of performance models, since each application is different. For many applications it might be difficult to build a general analytical performance model. Thus, the micro QoS support specifies only an abstract interface for performance models which defines a single method to estimate the performance (c.f. Figure 5.5).

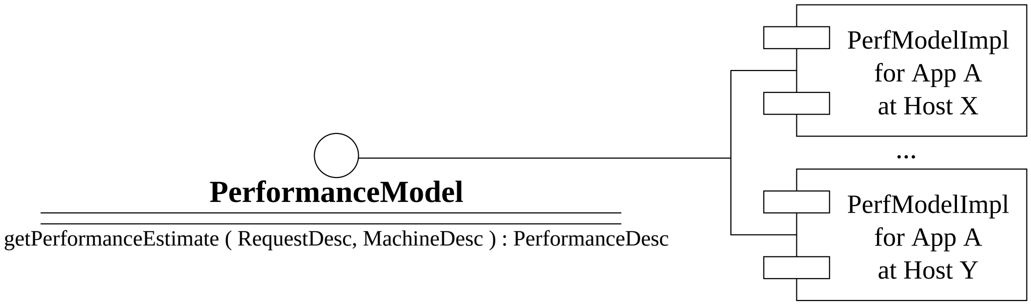


Figure 5.5. Performance Model Details

Figure 5.5 depicts the interface of the performance model and certain implementations for an application A running on different machines (X and Y).

Performance Model Input/Output

The major prerequisite for the advance runtime estimation of a specific application job are appropriate performance model inputs. These comprise information about the application job itself in the request descriptor, as well as about the hardware to use in the machine descriptor. This meta information is also referred to as application- and machine-specific QoS parameters which are supplied to the performance model by the client (request descriptor) and the service provider (machine descriptor). The output of the performance model is stored in the performance descriptor, which contains the prediction of the actual runtime required for a specific job.

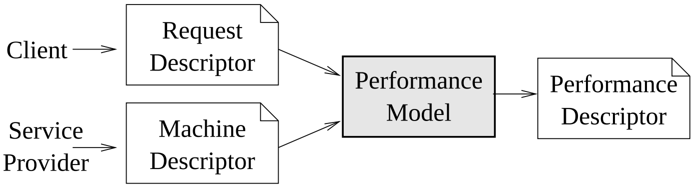


Figure 5.6. Performance Model Input/Output

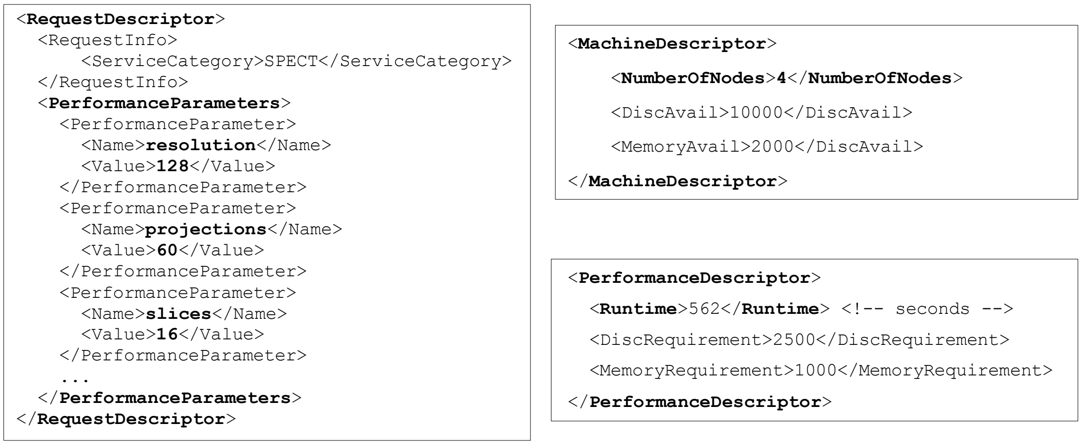
As shown in Figure 5.6 the *performance model* takes a *request descriptor* and a *machine descriptor* as input and returns a *performance descriptor*. The *request descriptor* comprises application-specific QoS parameters which contains meta-data about a specific service request that has to be supplied by a client during QoS negotiation. For example, in the case of an image reconstruction service, QoS parameters typically include image size and required accuracy.

The *machine descriptor*, supplied by the service provider via the QoS manager, comprises machine-specific QoS parameters which specify the resources (number of CPUs to use, available memory and disk space, etc.) that could be offered for an application job.

The *performance descriptor* returned by the performance model usually contains the estimated execution time and other parameters like the number of processors used to execute a job, required memory, and required disk space.

Example: Parallel MPI application

An example set of descriptors which constitute the input and output of the performance model implementation used in the experimental evaluation in Chapter 7 is depicted in Figure 5.7. In this case a parallel MPI application has been used and a corresponding performance model implementation has been provisioned in order to estimate the runtime for specific jobs parameterized with the machine size (i.e. number of used computing nodes of PC cluster).



```xml
<RequestDescriptor>
  <RequestInfo>
      <ServiceCategory>SPECT</ServiceCategory>
  </RequestInfo>
  <PerformanceParameters>
    <PerformanceParameter>
      <Name>resolution</Name>
      <Value>128</Value>
    </PerformanceParameter>
    <PerformanceParameter>
      <Name>projections</Name>
      <Value>60</Value>
    </PerformanceParameter>
    <PerformanceParameter>
      <Name>slices</Name>
      <Value>16</Value>
    </PerformanceParameter>
      ...
  </PerformanceParameters>
</RequestDescriptor>
```

```xml
<MachineDescriptor>
    <NumberOfNodes>4</NumberOfNodes>
    <DiscAvail>10000</DiscAvail>
    <MemoryAvail>2000</DiscAvail>
</MachineDescriptor>
```

```xml
<PerformanceDescriptor>
  <Runtime>562</Runtime> <!-- seconds -->
  <DiscRequirement>2500</DiscRequirement>
  <MemoryRequirement>1000</MemoryRequirement>
</PerformanceDescriptor>
```

Figure 5.7. Sample performance model input/output descriptors

Figure 5.7 exemplifies a request descriptor, a machine descriptor and a performance descriptor. The request descriptor comprises all necessary meta data related to the performance of a specific job. In this particular case the native application utilizes a medical imaging reconstruction algorithm and its runtime and other performance attributes are dependent on the following QoS parameters: image resolution, number of projections, and number of slices. These parameters are specific to a certain application, while the QoS framework supports arbitrary QoS parameters.

The machine descriptor, which is generated by the QoS manager, comprises the number of computing nodes as well as available disk space and memory. The output of the performance model is shown in the performance descriptor, comprising the runtime of this particular application job as well as the required disc space and memory.

The QoS manager may execute the performance model repeatedly with different machine sizes (number of computing nodes in the machine descriptor) in order to meet the constraints of the client. But this is subject to an algorithm applied in the QoS manager, which will be discussed in Section 5.3.

The performance model implementation used in the example is based on an analytical approach, which predicts the QoS attributes, such as the runtime, using a formula. If the provision of an analytical performance model for a certain application is not feasible, a performance model implementation may rest upon a database. In this case the database is used to relate typical problem sizes (i.e. QoS parameters) to resource needs like main memory, disk space and execution time (i.e. QoS attributes). Initially, the database is populated with data from representative test cases and later on, it may be expanded dynamically using the data from monitoring actual runs.

In summary, the first presented QoS model is used to estimate the performance of a specific application. In particular the main purpose of the performance model is to predict the application's runtime in advance to a potential job execution based on a set of QoS parameters. A performance model implementation is specific to an application and a certain machine (hardware). Moreover, it has to implement an abstract interface provided by the QoS infrastructure.

5.2.3 Pricing Model

The pricing model is a QoS model that is used to perform all calculations that are subject to pricing the usage of a service. The price of using a service may be dependent on different factors such as the required resources or a license needed to run the underlying application. The emphasis of this QoS support has been to explore the issues involved in creating a flexible approach with pricing models, in order to enable service providers to individually determine the price of a service usage.

Pricing Model Implementation

Similar to the performance model, the QoS support does not prescribed the actual implementation of the pricing model, since the business background of each service provider may be different (e.g. industry vs. academia). Consequently, the QoS support only defines an abstract interface for the pricing model in order to provide a flexible pricing policy to be defined by each service provider.

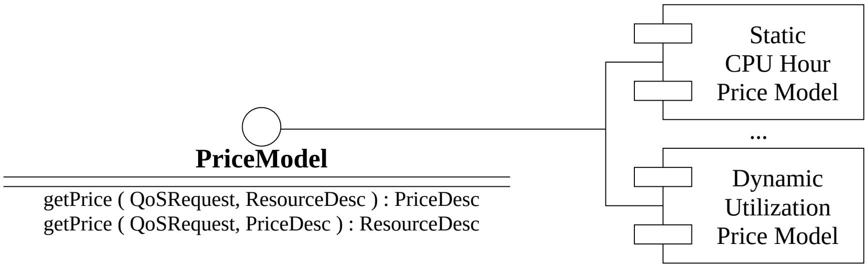

PriceModel
getPrice (QoSRequest, ResourceDesc) : PriceDesc
getPrice (QoSRequest, PriceDesc) : ResourceDesc
Static CPU Hour Price Model
...
Dynamic Utilization Price Model

Figure 5.8. Pricing Model Details

Figure 5.8 shows the pricing model interface and the available pricing model implementations. The pricing model interface specifies two operations to determine either the price based on a concrete resource allocation (i.e. resource descriptor) or potential resource allocations for a given price. The pricing model implementations follow the utility computing model [*Parkhill*, 1966], which defines charging for computing resources is based on the actual usage rather than on a flat-rate basis.

The QoS support system supports two pricing models: a fixed pricing model, where an individual price for each service request is determined by the amount of resources required (i.e. CPU-hour) as well as a dynamic pricing model, where the price is obtained dynamically dependent on the required resources and their current utilization. The first model applies a fixed rate on the actual resource consumption similar as offered by the Amazon Elastic Compute Cloud (EC2) [3].

Both pricing model implementations as well as further implications are discussed in the context of the experimental evaluation in Chapter 7.

Pricing Model Input/Output

The input and output of the pricing model are organized as follows: Typically, parameters, which in this case affect the pricing, are used as inputs in order to compute or estimate QoS attributes as outputs. Outputs from other QoS models may also be used as inputs to this model.

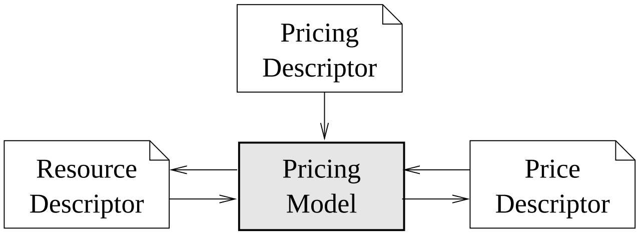


Figure 5.9. Pricing Model Input/Output

As shown in Figure 5.9, the pricing model is a two-way model which means that it can be used to calculate the price for a given resource allocation or feasible resource allocations for a given price. In either way the pricing descriptor (not to confuse with the price descriptor) comprises the service provider's configuration related to the pricing model (e.g. base price for a CPU hour in case of a fixed pricing model).

Price based on resource allocation: In this case the input to the pricing model is a resource descriptor with an explicit resource allocation which is generated on the fly by the resource model upon request of the QoS manager. The pricing model returns a price descriptor with the concrete price for the given resource allocation.

Resources allocations based on price: In this case the input to the pricing model is a price descriptor specifying the client's time and price constraints as well as the estimated runtime as obtained from the performance model. Given these inputs, the pricing model identifies potential resource allocations, which meet the clients price and time constraints. The main purpose is to reduce the number of resource allocations to be checked with the resource model, if they can be made available, by filtering potential resource allocations with prices that exceed the client's constraints.

The invocation of the pricing model in either way is performed by the QoS manager and is subject to a certain strategy as discussed in detail in Section 5.3.

[3] Amazon EC2, `http://aws.amazon.com/ec2/`

5.2.4 Resource Model

The resource model provides a uniform interface to the underlying scheduling system. This interface provides all operations to model and manage resource allocations for a certain job on a specific machine. Together with the performance model, which estimates the runtime for a specific job, the resource model uses the runtime as additional input to determine a concrete time slot for the job execution, i.e. the QoS attributes for the begin- and end time of a specific job execution.

Two-Phase-Commit Protocol

Alike the other QoS models the resource model is utilized by the QoS manager which in particular initiates queries for temporary resource reservations during the QoS negotiation and potentially confirms these temporary reservations. This process is also referred to as *two-phase commit protocol* in transaction-based distributed systems [*Skeen and Stonebraker*, 1983].

In this case a two-phase commit protocol is applied while the QoS negotiation: In the first phase the QoS manager queries for required resources and in case of availability, the resource model temporarily books the resources. Then the scheduled reservation is supplied to the client and only upon an explicit confirmation within a certain short time period, the temporary reservation is being made permanent in the second phase. The temporary reservation ensures that free resources are not being reported to clients more than once (c.f. over-booking) and the expiration of temporary reservations after a short period of time prevents unused reservations. A more detailed discussion about the implications of advance resource reservations can be found in [*Snell et al.*, 2000].

The interaction involved in the resource reservation between the QoS manager and the resource model corresponds to a dialog stated in the context of the two-phase commit protocol in [*MacLaren*, 2003]:

```
User: "Can I have 32 processors at 2:00pm for 2 hours?"
System: "No, there are only 20 free"
User: "Can I have 16 processors at 1:00pm for 5 hours?"
System: "Yes, but confirm within 60 seconds if you want them"
User: "I'll take them"
```

A realization of such a communication between a user and a system, which in this case is comparable to the QoS manager using the resource model, is highly sophisticated and obviously not possible in a generic way. But even if the system answers the first question with a simple ”no”, the last part can can be referred to as resource negotiation.

The resource model as defined in the context of the QoS infrastructure aims to provide operations to interface a concrete scheduling system as well as to enable resource negotiation and advance reservation following a two-phase commit protocol.

Resource Model Interface

The resource model interface provides mechanisms to obtain information about the actual availability of computing resources (e.g. number of free processors on a machine for a certain time period) as well as to book resources for specific jobs in advance. Furthermore, operations for already booked jobs are provided in order to acquire job status information or cancellation of jobs in order to free resources.

Figure 5.10 depicts the resource model interface and concrete implementations for a concrete underlying scheduling system. The resource model interface specifies a set of operations, which have to be implemented individually for each scheduling system.

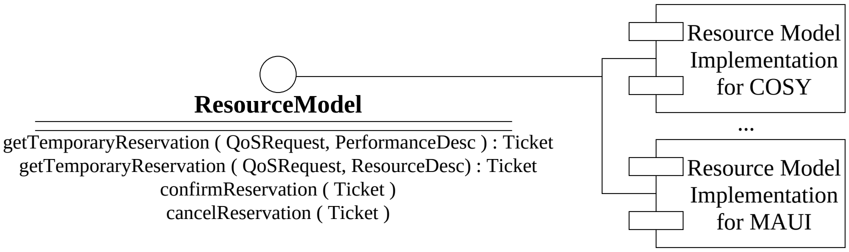

ResourceModel
getTemporaryReservation (QoSRequest, PerformanceDesc) : Ticket
getTemporaryReservation (QoSRequest, ResourceDesc) : Ticket
confirmReservation (Ticket)
cancelReservation (Ticket)
Resource Model Implementation for COSY
...
Resource Model Implementation for MAUI

Figure 5.10. Resource Model Details

The operations of the *ResourceModel* interface include two kinds of retrieval/-querying of a temporary resource reservation (*getTemporaryReservation*) as well as a confirmation (*confirmReservation*) or cancellation (*cancelReservation*). Temporary resource reservations are identified by a *Ticket* and may be retrieved based on the client's QoS constraints (*QoSRequest*) and either on the output of the performance model (*PerformanceDesc*, first operation) or on the output of the pricing model (*ResourceDesc*, second operation).

Generally, scheduling systems that support advance reservation can be distinguished by the granularity of their reservation support, i.e. resource- and time-based (core-grained), user- and/or group-based and finally job-based (fine-grained) [*MacLaren*, 2003]. This QoS infrastructure relies on a fine-grained reservation support and associates one reservation with exactly one job. As a consequence, an implementation of the resource model for a concrete scheduling system has to provide an appropriate fine-grained reservation support. Typically, either the scheduler's native job-based reservation capabilities are utilized or a mapping of resource reservations to jobs (i.e. map core-grained reservations to fine-grained reservations) has to be implemented additionally.

As initially shown in Figure 5.10, resource model implementations for two scheduling systems with support for advance reservations are provided: One resource model implementation for the Maui scheduler from Cluster Resources Inc. [*Jackson et al.*, 2001] and one for the COSY scheduler from NEC [*Cao and Zimmermann*, 2004]. In the following, both scheduling system and their linking to the resource model are briefly outlined.

Interfacing with COSY

COSY is a lightweight implementation of a local job scheduling system that supports both queue scheduling and advance reservation. COSY queue scheduling utilizes the well-known first-come-first-serve (FCFS) algorithm with aggressive backfilling mechanisms and priority management. Advance reservations with COSY are job-based supporting start and latest completion time. The actual booking of resources for a specific job follows a two-phase-commit protocol as described earlier in this section.

COSY has been implemented in C++ and provides a command-line interface as well as a Java API library. This Java API library is utilized by the COSY specific resource model implementation. Moreover, the mechanisms provided by the COSY Java API support the querying and booking of resources in advance. If resources are available at the requested time a ”ticket” for the requested time-slot is issued, which has to be confirmed within a certain short time frame.

In summary, COSY suits well in this QoS support system because it requires a job-based advance reservation that is requested and confirmed with a two-phase commit protocol. COSY natively supports this two-phase commit protocol. More information about COSY can be found in [*Cao and Zimmermann*, 2004].

Interfacing with MAUI

As a second scheduling system that is supported by this QoS system, Maui has been selected. Maui is a wide-spread batch scheduling system with a rich range of functions, including advance reservation.

Maui is a batch scheduling system that is well suited for different Linux-based high performance computing platforms. It uses aggressive scheduling policies in order to optimize resource utilization and minimize job response time. It allows a high degree of configuration for reservation policies. Maui possesses an advance reservation infrastructure allowing sites to control exactly when, how, and by whom resources are used.

Due to the flexible configuration of Maui's reservation infrastructure, it can be utilized similar to COSY, except that reservations are booked immediately with a single commit compared to the two-phase-commit of COSY. As a consequence, the two phases have to be emulated by the Maui-specific resource model implementation. If a resource reservation is not confirmed within a specific short time frame, the resource model implementation has to cancel the reservation, which is also supported by Maui.

In summary, Maui scheduling system provides more functionality than required, but due to its flexible configuration, it suites well to be supported by this QoS infrastructure. More information about Maui can be found in [*Jackson et al.*, 2001].

5.2.5 QoS Manager

The QoS manager is the central component of the service-side QoS infrastructure which interacts with all previously mentioned QoS models (performance model, pricing model and resource model). The main objective of the QoS manager is to generate a QoS offer given the client's request. Subsequently, the overall picture of the microscopic QoS management is being detailed with the information gained in the recent sections.

Figure 5.11 sketches the complete picture of the microscopic QoS management, its input/output and the internal interaction between the QoS models via the QoS manager (i.e. performance model, pricing model and resource model). The QoS management receives a QoS request and a request descriptor from a client and generates a QoS offer, which is eventually returned to the client.

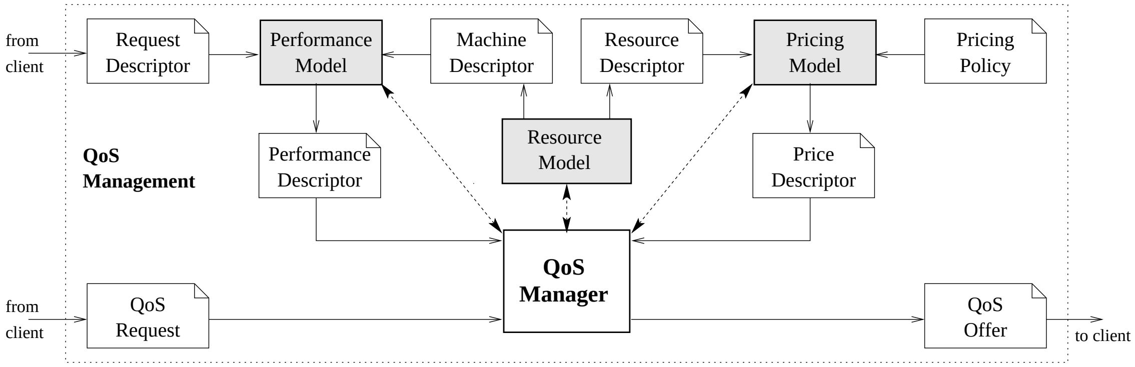
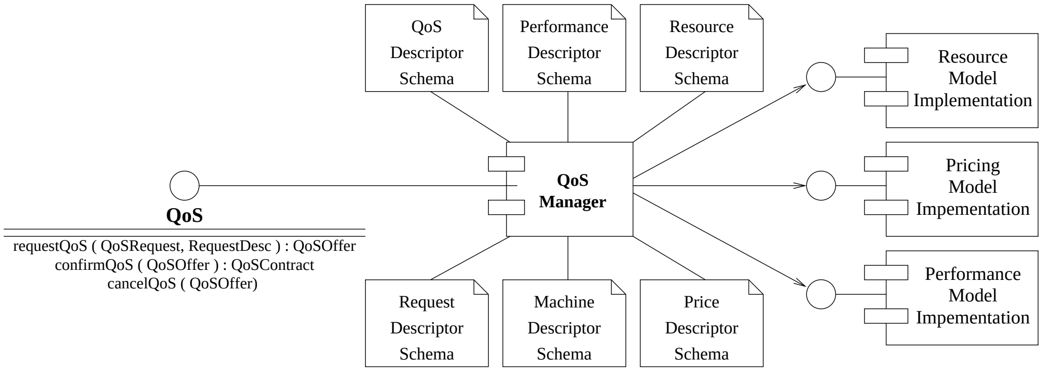


Figure 5.11. Micro QoS management details [*Benkner and Engelbrecht*, 2006]

A QoS request contains the desired constraints for the QoS attributes, while the request descriptor, which mainly serves as input to the performance estimation, comprises metadata about an application's input data. Internally the QoS management aims to determine whether the client's QoS constraints can be fulfilled utilizing the QoS models, i.e. the performance model to estimate the runtime, the resource model to book resources and the pricing model to determine the price of a service usage.

Figure 5.12 sketches an architectural view of the QoS management including all interactions of the QoS manager with the QoS models via their interfaces. The actual implementations of the QoS models are hidden as these are subject to a specific application, machine and/or pricing strategy. The central QoS manager also provides an interface, which consists of the basic QoS operations for a client (i.e. WSDL operations). These operations include requesting a new QoS offer (*requestQoS*) and its confirmation (*confirmQoS*) or cancellation (*cancelQoS*). The request for a new QoS offer (*QoSOffer*) is based on the client's constraints (*QoSRequest*) and the meta information about the application job (*RequestDesc*). All used descriptors are specified by corresponding XML schemes.



QoS Descriptor Schema
Performance Descriptor Schema
Resource Descriptor Schema
Resource Model Implementation
QoS Manager
Pricing Model Impementation
QoS
requestQoS (QoSRequest, RequestDesc) : QoSOffer
confirmQoS (QoSOffer) : QoSContract
cancelQoS (QoSOffer)
Performance Model Impementation
Request Descriptor Schema
Machine Descriptor Schema
Price Descriptor Schema

Figure 5.12. Micro QoS management implementation

In order to achieve the overall objective to generate an appropriate offer to the client, the QoS manager relies on heuristics that consider the outcome of all involved QoS models. In this situation a heuristic is preferred to accomplish results (i.e. offers) within a reasonable time and without consuming large amounts of computing power as perhaps required with more generic methods. A concrete heuristic is referred to as *QoS management approach* and specifies the orchestration of the QoS models and their interactions with the QoS manager. This most challenging issue is addressed by different QoS management approaches as discussed in detail in the next section.

5.3 QoS Management Approaches

This section describes approaches of the QoS manager to interact with all involved QoS models to generate a QoS offer based on the given client constraints. The presented approaches are being discussed in terms of individual algorithms, which are executed by the QoS manager. Two distinct algorithms are proposed, each realizing the offer generation in a customized fashion with a specific focus. This section has mainly been derived from [*Benkner and Engelbrecht*, 2005].

In general, the QoS manager aims to perform a multiobjective optimization, which is defined as the process of optimizing two or more conflicting objectives simultaneously [*Sawaragi et al.*, 1985]. In case of the QoS management the client usually specifies such conflicting objectives with constraints for the desired execution time and its price. The QoS manager addresses this situation by applying customized algorithms.

The QoS management approaches implement the main challenge in this context to generate a QoS offer based on the client's request. The offer generation process necessitates an according orchestration of all involved QoS models as well as the handling of appropriate interactions between these QoS models. This process is conducted according to an algorithm, which basically specifies the sequence of interactions with the following involved QoS models:

- Performance Model
- Pricing Model
- Resource model

The algorithms also consider the specific characteristics of parallel applications, which are executable with different machine sizes. As a consequence, the interaction sequence with the QoS models may be performed iteratively, parameterized with the machine size (e.g. number of computing nodes on a cluster).

General Considerations

The QoS manager considers the outcome of the performance model, the availability of resources via the resource model, and the service provider's pricing model to decide whether the client's QoS constraints can be fulfilled. If this is the case, the QoS manager generates a corresponding QoS offer, which is then returned to the client. Each interaction with a QoS model requires an invocation of a specific operation of this QoS model, which affects the overall performance of QoS manager algorithm. The impact on the overall performance is even more important in case of parallel applications, which are executable with different machine sizes, because then, a specific QoS model may be invoked several times for each machine size.

Consequently, the time consumption of invoking QoS models has been investigated and the experiments presented in Chapter 7 have shown that operations invoked with the resource model, which are actually executed in the underlying scheduling system, tend to take significantly more time compared to operation invocations with the other QoS models. Obviously the time consumption of a certain QoS model operation invocation is dependent on a number of external factors, such as the machine load, which are far beyond the scope of this work.

However, the invocation of a specific QoS model impacts the overall performance of the micro QoS management and this will be taken into account in the subsequent discussion of the different QoS manager approaches.

Primary Focus

In the following different approaches of the QoS manager to provide offers to clients are discussed in detail by presenting the algorithms applied in the QoS manager. The main distinguishing aspect of these algorithms is their primary focus on a certain client constraint, which is either time or price. This leads to the following approaches:

- Prime time approach
- Prime price approach

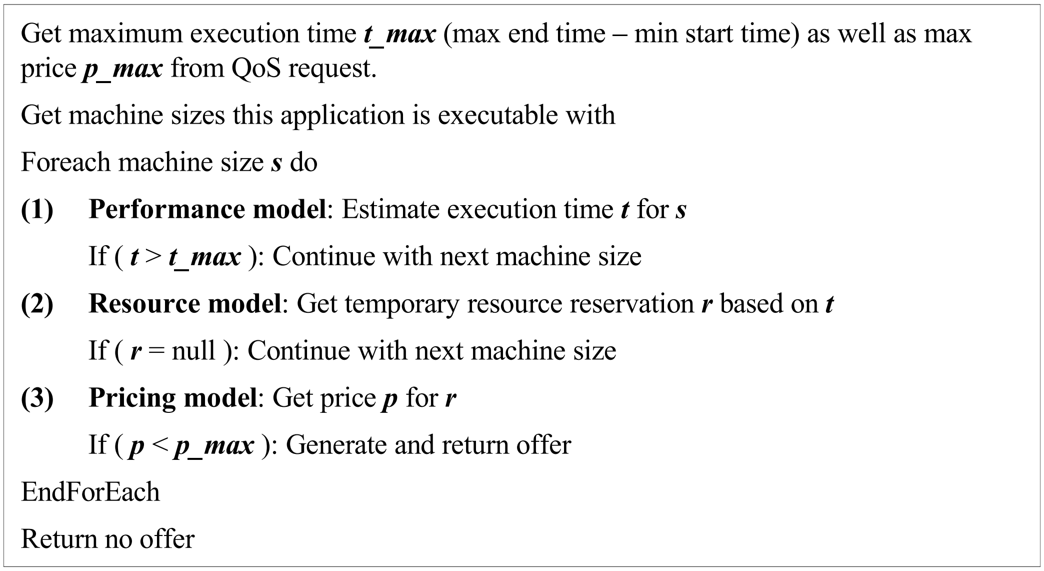
The *prime time approach*, which in this context has nothing to do with the primary viewing time in TV, indicates the primary focus on the client's time constraints, while the *prime price approach* puts the emphasis on the price constraint of the client. Both approaches aim to meet all client constraints and their naming just indicates their primary focus, which means that the primary client constraint is considered prior to the other constraint(s) in a corresponding algorithm.

Subsequently, both approaches are being discussed in detail, mainly by presenting corresponding algorithms, which are executed by the QoS manager. It should be noted, that the terms "QoS manager approach" and "QoS manager algorithm" are being used interchangeably, even if the approach rather describes the overall strategy/concept of the QoS manager, with an algorithm being in place to execute the strategy.

5.3.1 Prime Time Approach

The prime time approach focuses primarily on the client's time constraints. To fulfill the time constraints first, the QoS manager applies an algorithm that estimates and books the required resources first. Only if a prediction of the required resources as well as their availability can be ensured along the lines of the client's time constraints, the algorithm continues and checks if the price constraint can be met as well.

The applied algorithm of the prime time approach is depicted in Figure 5.13, showing mainly the sequence of the invoked QoS models, the continuation criteria as well as the overall iterative fashion in case of a parallel application.



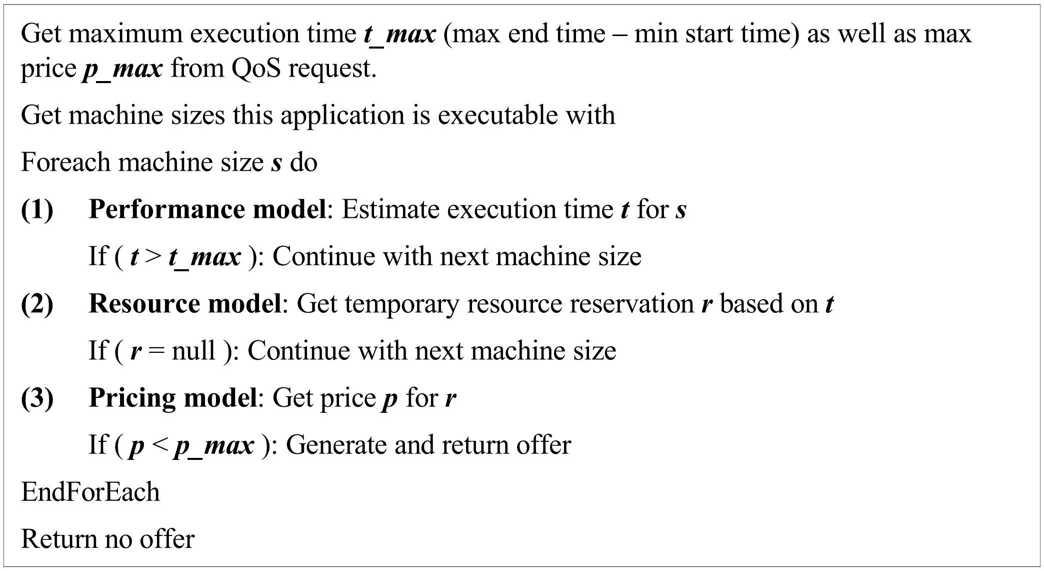
```
Get maximum execution time t_max (max end time – min start time) as well as max
price p_max from QoS request.
Get machine sizes this application is executable with
Foreach machine size s do
(1)   Performance model: Estimate execution time t for s
      If ( t > t_max ): Continue with next machine size
(2)   Resource model: Get temporary resource reservation r based on t
      If ( r = null ): Continue with next machine size
(3)   Pricing model: Get price p for r
      If ( p < p_max ): Generate and return offer
EndForEach
Return no offer
```

Figure 5.13. Prime time algorithm

For each machine size the application is executable on, the prime time algorithm guides the QoS manager as follows: (1) Firstly, the performance model is executed to obtain the estimated execution time with the current machine size. If the execution time exceeds the maximum execution time from the client's constraints the algorithm starts over with the next machine size[4]. (2) If the time constraints can be met, the QoS manager continues and contacts the resource model to check whether the required resources (i.e. machine size) can be made available. If the resources cannot be made available, the algorithm attempts the next machine size (if available). Assuming the required resources are available, the QoS manager creates a temporary resource reservation. (3) Finally, given the start- and end-time of the reservation, the pricing model is invoked to obtain the price of the resource allocation. If the service provider's price also meets the client's price constraint the QoS manager generates a corresponding QoS offer, which is then returned to the client or cancels the temporary resource reservation, if the price constraint of the client can not be met.

If any of the client's constraints cannot be met given all available machine sizes, the QoS manager may be configured to return the closest offer to the client or none. In any case, the client has to confirm a QoS offer in order to establish a QoS contract. The confirmation of a QoS offer is part of the QoS negotiation which is subject to be discussed in Section 5.4. Moreover, the confirmation is required for the temporary resource reservation to become a permanent reservation (c.f. two phase commit protocol described in Section 5.2.4).

[4] This is based on the assumption that the machine sizes are ordered ascending and that increasing the machine size causes a reduction of the execution time



```
Get maximum execution time t_max (max end time – min start time) as well as max price p_max from QoS request.
Get machine sizes this application is executable with
Foreach machine size s do
(1)   Performance model: Estimate execution time t for s
      If ( t > t_max ): Continue with next machine size
(2)   Resource model: Get temporary resource reservation r based on t
      If ( r = null ): Continue with next machine size
(3)   Pricing model: Get price p for r
      If ( p < p_max ): Generate and return offer
EndForEach
Return no offer
```

Figure 5.14. Prime price algorithm

5.3.2 Prime Price Approach

The prime price approach focuses primarily on the client's price constraint. In this case the QoS manager applies an algorithm that estimates the required resources and identifies potential resource allocations that meet the price constraint first. Only if a prediction of the required resources as well as resource allocations can be ensured along the lines of the client's price constraint, the algorithm continues and checks if a resource allocation can be booked at all.

Analogue to the prime time approach, the algorithm applied in the prime price approach is shown in Figure 5.14.

For each set of resources the prime price algorithm leads the QoS manager as follows: (1) Firstly, the QoS manager executes the performance model equally to the prime time algorithm and compares the estimated execution time with the maximum runtime derived by the time constraints of the client. (2) If the client's time constraints can be met, the QoS manager continues and executes the pricing model. With the prime price algorithm the pricing model is supplied with the client's price constraints and the estimated runtime given in the performance descriptor. The pricing model returns one or more possible resource allocations (i.e. time frames with begin- and end-time) in which the application could be executed not exceeding the specified price constraint. (3) Assuming that the pricing model returns one ore more resource allocation time frames, the QoS manager checks with the resource model if a concrete resource allocation can be made available. If this is the case, the QoS manager generates a corresponding QoS offer, which is returned to the client.

The overall procedure if the client's constraints cannot be met once or at all is the same as with the prime time algorithm.

5.3.3 Comparison

The prime time and the prime price algorithms are depicted in Figure 5.15, which also contrasts both approaches with their sequence of invoking the involved QoS models enforced by the central QoS manager. The loops indicate again the optionally iterative nature of the algorithms in case of parallel applications, which may be executed with different machine sizes.

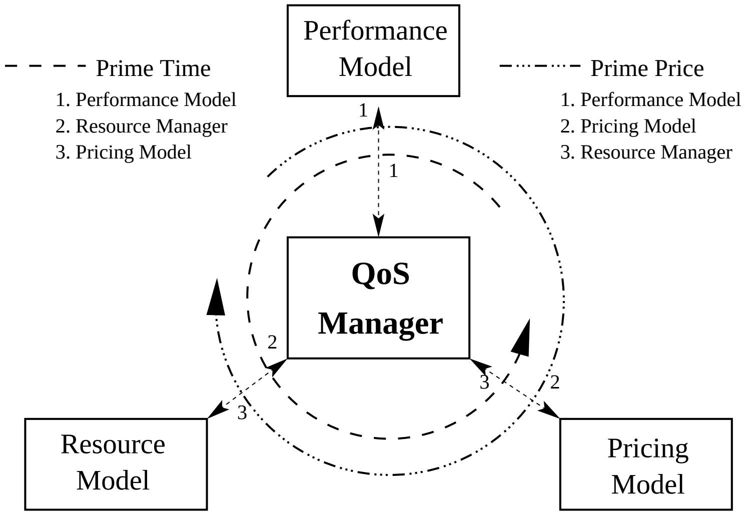


Figure 5.15. QoS management approaches [*Benkner and Engelbrecht*, 2005]

Both algorithms are subject to be configured individually with each service at the time of the deployment; hence, it is up to each service provider to decide which approach will be used. Typically, a commercial service provider will prefer the approach focusing on the pricing. While an academic service provider such as a University has no commercial background and consequently less interest in pricing and thereof will more likely prefer the prime time approach.

However, the applied algorithms of both approaches have been investigated with respect to the time consumption of the involved models. Table 5.1 shows how much time was consumed by the QoS models and the QoS manager while performing according to the prime time and prime price algorithm.

The table mainly proves the assumption made in the beginning of this section that most of the time with the prime time algorithm is consumed by the resource model. In contrast, the prime price approach shows a more balanced time distribution. However, the total absolute time required for the generation of an offer with the QoS

	Prime time	Prime price
Performance Model	25%	25%
Resource Model	40%	25%
Pricing Model	18%	30%
QoS Manager	17%	20%

Table 5.1. QoS management approaches comparison

management was not significantly differing between the applied algorithms; hence, the overall time consumption gave no indication which algorithm should be preferred in general.

Furthermore, this experiment showed that an offer generation can be conducted (at all) using one of the presented algorithms, which eventually underpins the concept of using different QoS models and the QoS manager to enforce a certain algorithm.

Summary

The QoS management approaches implement the main challenge of the microscopic QoS support to generate a QoS offer for a given QoS request. This process relies on an algorithm which basically specifies the sequence of the interactions with the involved QoS models. Two such algorithms have been presented, each focusing primarily on a certain aspect of the client's constraints, i.e. time or price.

5.4 Macroscopic Quality of Service

This section details the macroscopic Quality of Service negotiation. The macro QoS support describes to process of a client interacting with one or more services - all exposing the same native application - in order to determine the best suited service given its QoS constraints. In contrast to the Micro QoS support, which describes how an offer is generated within a single service, the Macro QoS support is a client-driven procedure, where a client tries to negotiate with one or more services in order to obtain a suitable offer and optionally wrap up a deal. The potential service candidates are usually located by utilizing a registry service as provided by the VGE system.

Clients may utilize sophisticated negotiation strategies, such as auctions in order to invite service providers to underbid each other and finally reveal a fair price among the auction participants. On the other hand service providers are able to follow a certain pricing model in order maximize their profit.

This sections initially details the basic QoS negotiation process which follows the request-offer model, which will be introduced in 5.4.1, and the client handling of multiple offers, when negotiating with more than one service. Initially a single offer-request attempt is discussed, followed by the description of a client requesting offers in a round-based fashion guided by a certain negotiation strategy based on auctions. Therefore, a number of auctions are discussed and their applicability in this context is investigated.

5.4.1 Basic QoS Negotiation

The basic QoS negotiation can be outlined as the client's attempt to establish a QoS contract with a service provider. In this context, the client utilizes a negotiation approach that follows a request-offer-model as outlined in Section 5.1 and discussed in further detail subsequently.

The basic QoS negotiation scenario is shown in Figure 5.16. A client requests a new offer from each candidate service with initially creating a request and passing this request to each service. Each contacted service may return a QoS offer to the client consecutively. The client then has to decide which offer is most suitable and finally confirm this specific offer.

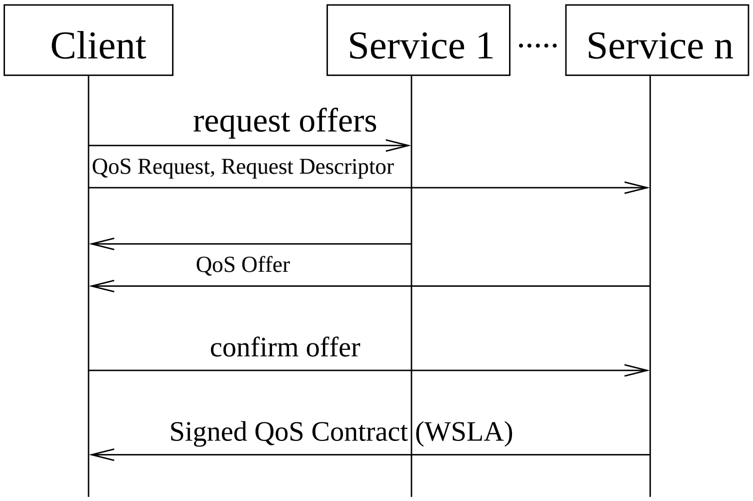


Figure 5.16. Basic QoS Negotiation [*Benkner and Engelbrecht*, 2005]

The details of this basic QoS negotiation process can be outlined as follows: Initially the client generates a request descriptor containing meta information of the concrete application job as well as a QoS request following the WSLA specification with the actual constraints for the SLA parameters (i.e. begin- and end-time as well as the requested price). Please note that, SLA parameters in the context of the QoS negotiation correspond to QoS attributes in the micro QoS management. The client then requests a new offer from each service passing along the QoS request and the request descriptor. The Micro QoS support on the service side individually tries to achieve the client's constraints and generates an appropriate QoS offer which is then returned to the client.

Since clients usually negotiate with multiple services on a low level of trust, each QoS offer has a short expiration time and finally, it is up to the client to confirm a specific offer before it expires. If a QoS offer is confirmed by the client, a QoS contract is established and signed by both parties. Alternatively, the client may not confirm any of the offers and restart the negotiation with different parameters (i.e. with a new QoS request). This is referred to as advanced QoS negotiation that consists of an iterative offer requesting process, which will be discussed in further detail in Section 5.4.2.

Offer Assessment

Assuming the client negotiates with more than one service provider, the most interesting issue in this context is the evaluation of the received offers. This can either be an interactive process, when the final decision which service should be confirmed is made by a human in front of a computer, or an automated process, when a certain pre-defined assessment procedure decides which QoS offer is considered as the best suitable. This work focuses on an automated offer-assessment process which is also referred to as scoring. Each QoS offer is associated with a score in order to automatically rank all QoS offers by their score. Contrarily to an interactive human decision this is also feasible in the context of advanced negotiation strategies, which rely on multiple rounds of offer-requests and -assessments.

An automated offer-quality evaluation or offer-scoring relies on the assessment of the involved SLA parameters by comparing each SLA parameter from all offers individually. For instance the begin-times of all offers are compared and assessed against each other following the predicate defined in the initial request (c.f. QoS requests and offers follow the WSLA specification as described in [*Heiko et al.*, 2003]): Scoring a begin-time of 01-01-08 12:15pm exceeds a begin-time of 01-01-08 12:10pm if the initial request defines a begin time of 01-01-08 12:00pm with the predicate "greater". In general, comparing the price is more illustrative: The score of a more favorable price exceeds a less favorable price (i.e. a low price is better than a high price, except the unlikely case if the predicate is defined differently).

After the definition of the individual assessments of each SLA parameter, the major question remains, how an accumulative assessment of all SLA parameters can be addressed. The overall score of an offer V_i is a numeric value, where i is the index of the offer. This overall score is generated by the sum of the individual scores of each SLA parameter v_j, where j is the index of the SLA parameter and n the total number of SLA parameters. Consequently, the total scoring value is defined in equation 5.4.1.

$$V_i = \sum_{j=0}^{n} v_j \tag{5.4.1}$$

The scoring values for each SLA parameter v_j can either be defined following a certain procedure, such as the percentage assessment, or by an individual formula. Furthermore, an according weighting of each SLA parameter can be defined to specify the preferences among the SLA parameters individually. E.g. if the price is more important than the time, the weighting of the price would exceed the weighting of the time. Subsequently, the assessment method performed in the context of the macro QoS negotiation is specified as well as an illustrative example is presented.

Percentage assessment: This procedure foresees that for each SLA parameter p_j a score is determined whereas in general a more distant value p_j from the requested value p_j^{req} results in a higher score. The most distant value corresponds 100% whilst a very close value of p_j to the requested one results in a very low score (towards 0%). In order to determine proportional scores the minimum and maximum has to be determined in advance among the requested and received offers. Consequently, a percentage assessment for an SLA parameter achieves the scoring value of v_j as defined in Equation 5.4.2.

$$v_j = \left| \frac{p_j^{req} - p_j}{p_{j_{max}} - p_{j_{min}}} \right| \tag{5.4.2}$$

A pure percentage assessment procedure implies that all involved SLA parameters have the same weight. A potential drawback of this approach could be that large intervals between the minimum and maximum of one SLA parameter and small intervals between the minimum and maximum of another SLA parameter may cause unwanted side-effects on the total assessment. Thereof an extension of the initial total scoring formula 5.4.1 with individual weighting factors is discussed in the following.

Weighted assessment: This assessment mechanism allows to specify the preferences among the SLA parameters more precisely by associating a corresponding individual weighting factor w_j for each SLA parameter. The extended total scoring is depicted in equation 5.4.3.

$$V_i = \sum_{j=0}^{n} w_j \ v_j \tag{5.4.3}$$

In this case different weightings of the SLA parameters are considered by an according weighting factor w_j. Usually the sum of all weights is defined as 1.

In the following an example illustrates the QoS offer assessment.

Offer Assessment Example

The offer assessment example describes how the percentage assessment is applied to different offers. The example assumes three SLA parameters (p_1 to p_3) for the price, the begin time and the end time with corresponding predicates. These SLA parameters are specified by the client in the QoS request.

$$\begin{array}{lcllcllcl}
p_1 & \ldots & \text{price} & p_2 & \ldots & \text{begin time} & p_3 & \ldots & \text{end time} \\
 & & & & & & & & \\
p_1^{req} & \leq & 15 \text{ €} & p_2^{req} & \geq & 12:00 & p_3^{req} & \leq & 14:00
\end{array}$$

The example furthermore assumes three service providers A, B and C, each generating an individual QoS offer containing certain values for all SLA parameters, which fulfill the client's request.

$$\begin{array}{lcllcllcllcl}
\text{Offer a} & & & \text{Offer b} & & & \text{Offer c} & & & \text{Offer d} & & \\
p_{1_a} & = & 15 \text{ €} & p_{1_b} & = & 7 \text{ €} & p_{1_c} & = & 13 \text{ €} & p_{1_d} & = & 9 \text{ €} \\
p_{2_a} & = & 12:00 & p_{2_b} & = & 12:15 & p_{2_c} & = & 12:15 & p_{2_d} & = & 12:00 \\
p_{3_a} & = & 13:30 & p_{3_b} & = & 13:00 & p_{3_c} & = & 13:00 & p_{3_d} & = & 13:30
\end{array}$$

Given these offers and the request, it is obvious that *Offer a* will be the worst rated, while *Offer b* is the best. The interesting offers are c and d due to their contrary SLA parameters (i.e. *Offer c* represents the fast and expensive job execution and *Offer d* states a cheap and long-lasting one). However, the minimum and maximum for all SLA parameters can be derived as follows:

$$\begin{array}{lcllcllcl}
p_{1_{min}} & = & 7 \text{ €} & p_{2_{min}} & = & 12:00 & p_{3_{min}} & = & 13:00 \\
p_{1_{max}} & = & 15 \text{ €} & p_{2_{max}} & = & 12:15 & p_{3_{max}} & = & 14:00
\end{array}$$

Finally, the actual offer scoring can be conducted according to the formula in Equation 5.4.2 as follows:

$$\begin{array}{lcllcllcllcl}
v_{1_a} & = & 0 & v_{1_b} & = & 1 & v_{1_c} & = & 0.25 & v_{1_d} & = & 0.75 \\
v_{2_a} & = & 0 & v_{2_b} & = & 1 & v_{2_c} & = & 1 & v_{2_d} & = & 0 \\
v_{3_a} & = & 0.5 & v_{3_b} & = & 1 & v_{3_c} & = & 1 & v_{3_d} & = & 0.5 \\
 & & & & & & & & & & & \\
V_a & = & 0.5 & V_b & = & 3.0 & V_c & = & 2.25 & V_d & = & 1.25
\end{array}$$

In this example the offer of service provider b is clearly assessed as best offer, following by c. This example does not explicitly use any weighting factors, i.e. an equal weighting of the SLA parameters is assumed. In the following a weighting, which favors the price is assumed as follows:

$$w_1 = 0.7 \qquad w_2 = 0.15 \qquad w_3 = 0.15$$

$$V_a^w = 0.075 \qquad V_b^w = 1.0 \qquad V_c^w = 0.475 \qquad V_d^w = 0.6$$

In this case the offer of service provider b remains best scored, but the second best scoring is achieved by service provider d. In any case the offer assessment is a central issue in the decision process of the client which offer may be confirmed. In this work only the mentioned strategies are discussed and applied, while generally even more sophisticated approaches exist to assess multiple criteria as within a QoS offer.

5.4.2 Advanced QoS Negotiation

The advanced QoS negotiation adopts a customized approach following the round-based request-offer model as briefly addressed in the previous section. This approach follows a similar protocol as defined in the agent theory by the FIPA standards [*Raja et al.*, 2008]. Furthermore, the advanced QoS negotiation allows the adoption of different auction models for single goods as specified in [*Vickrey*, 1961], which actually have their origin in economic science.

The first step towards advanced QoS negotiation comprises an increase of potential requests a client may pass to a service provider as specified in a single request-offer model utilized in the basic QoS negotiation. With the advanced QoS negotiation a client requests a new offer from all involved service providers for several times. On each offer request the client assesses all received offers and creates a new request for the next round based on this assessment. This process may be iteratively repeated until a certain condition is fulfilled. The overall round-based advanced QoS negotiation is depicted in Figure 5.17.

The actual number of bidding-rounds, the details of the offer assessment in each round as well as the eventual termination conditions are subject to a concrete implementation which follows a distinct auction model. In the following, general issues related to auctions as well as their overall applicability in the context of the Grid are discussed in further detail.

Grid Auctions

In general an auction is defined as the process of buying and selling items by offering them up for bid, taking bids, and then selling it to the winning bidder. In this context the item is the execution of a specific application job by a service and the services are bidding for the execution of this job. The winning bidder is a service, and a QoS contract is exchanged between the client and the winning bidder service. The

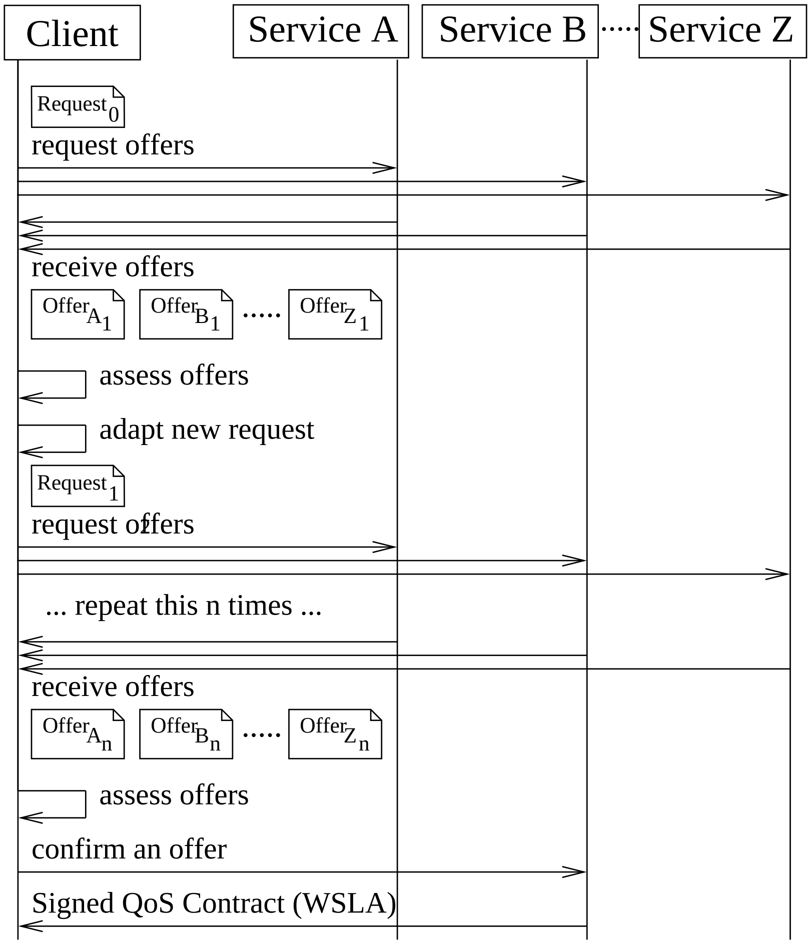


Figure 5.17. Advanced QoS Negotiation

actual bidding-process is round-based and the number of rounds and consequently the termination conditions are subject to a concrete auction model.

Contrarily to economic auctions which just consider the price to determine the best bid, all qualities of a service are assessed in this context in order to score the bids. A concrete job execution to be performed by a service is associated with a number of qualities, such as the actual time of the job execution and its price. These qualities may be assessed and scored along the lines of an according assessment strategy such as the percentage assessment introduced in the previous section. In any case the bid assessment is subject to a concrete auction model as outlined in the following.

Given this brief introduction to auction models, they appear applicable in the context of advanced QoS negotiation as depicted in Figure 5.17. Subsequently, the following auction models are discussed in further detail:

- First-Price Sealed-Bid auction

- English auction
- Dutch auction
- Vickrey auction

The description of these auction models comprises an overview of each model as well as details about the applicability of a model in the context of the advanced QoS negotiation. Furthermore, each auction model is examined with respect to the actual number of bidding rounds, the corresponding termination condition, and the concrete bid assessment.

First-Price Sealed-Bid auction

The First-Price Sealed-Bid auction constitutes an auction where bids can be submitted until a pre-defined deadline, but only one bid per bidder, which corresponds to a single round of bidding, and the best bid, which is determined by the best price, wins instantly. Bids are submitted in a sealed fashion, which implies that all bidders create their bids without any knowledge of the other bids. Moreover, the winning bidder has to fulfill exactly the conditions as specified in its bid (i.e. pay/charge the specified price) as opposed to other auction models, where e.g. only the price of the second best bid must be paid.

The main point of criticism in the context of this auction design as described by [*Ausubel*, 2003] is the strategic behavior of the bidders utilizing the knowledge about the number of auction participants. If the auction participants know the total number of bidders the situation is as follows: The fewer bidders involved in such an auction the lower overall price is achieved. A bidder simply assumes that given less bidders the chances increase that a lower bid wins. This issue is often referred to as the main obstacle to prevent the applicability of this auction model in a business context with a known number of participants.

In the context of advanced negotiation the number of service providers is only known to the client which initiates the auction and not to the auction participants, which are in this case the bidders. Consequently, this issue does not have any effects on the finally achieved contract.

In summary, the First-Price Sealed-bid auction can be applied for the advanced QoS negotiation and it complies with the request-offer model, which is executed just once. The involved service providers create their offers (bids) individually upon a client request without any influence from other service providers or their concrete bids. The client then assesses all bids according to a distinct assessment strategy and declares the winner by exchanging a contract with the winning service provider.

Dutch auction

The name of the Dutch auction has its origin in its use on dutch flower markets. This auction quickly arranges a price and thus it is well suited for many similar goods or perishable goods. The actual auction starts with an exorbitant price, but the price is decremented recurrently within constant timeframes. The first auction participant which accepts the current price is the winner of the auction. The winner of the dutch auction also has to pay the accepted price not as opposed to other auction models, where e.g. the second best price has to be paid.

The Dutch auction is strategically equivalent to the First-Price Sealed-Bid auction, which means that the price achieved may depend on the number of auction participants [*Vickrey*, 1961]. But as the number of participants is in general not known to the participants themselves, this should not have any effects on the finally achieved contract.

The applicability of this auction design in the context of the advanced QoS negotiation is limited due to the fact that the only changing quality within the auction is the recurrently decreasing price with time. This is also implied by the general focus of this auction model, which is on similar goods, as opposed to the Grid, where a lot of different goods (even similar jobs with varying quality of service) are sold. Thus, it can be assumed that only a few auction participants exist for a concrete job and a Dutch auction would eventually achieve a low price.

In summary, it can be concluded that the applicability of this auction model is fairly limited and will not be further investigated. On the other hand it should be noted, that the dutch auction may be a well suited model in a similar context to sell compute cycles, as currently offered in the Amazon Elastic Compute Cloud (Amazon EC2)[5].

Vickrey auction

The Vickrey auction is also referred to as Second-Price Sealed-Bid auction. It corresponds mainly to the first-price sealed-bid auction with the only difference that the winner has to pay the second best price. The background of this auction model is to elicit the participants' true willingness to pay as well as to limit strategical acting. Assume a participant's bid exceeds his own willingness to pay, he is at risk that another participant bids similar, and finally he may win and be forced to buy at a loss. And vice-versa, if a participant underbids his own willingness to pay, he runs the risk that another bidder wins and in the end buys the item at a lower price than the amount this bidder would be willing to pay. Consequently, in this type of auction it is in the participants' best interest to submit a truthful bid [*Vickrey*, 1961].

[5] Amazon EC2, `http://aws.amazon.com/ec2/`

The basic procedure of the Vickrey auction is equivalent to the First-Price Sealed-Bid auction and consequently, the implementation of this procedure is identically. The only difference is that the winning participant just has to fulfill the second best scored bid. As it might not be possible for the winner to meet certain qualities of another bidder (e.g. due to different resource constraints) this auction model cannot be applied in the context of the advanced QoS negotiation at all.

In order to exemplify this situation two service providers are assumed to bid up for the execution of a job with a certain price at a given time. Service provider *a* offers to run the job in 2 hours with a price of 0.2 €. Service provider *b* offers to run the job in 1 hour with a price of 0.4 €. The client emphasizes a fair price more than on the execution time and consequently assesses that service provider *a* has the best offer. Given the auction design of the Vickrey auction, service provider *a* has now to meet the second-best offer, which specifies less time for the execution of the job than this service provider originally offered. Thus, service provider *a* might not be able to meet the execution time e.g. due to limited resources.

In summary, this auction model is not applicable in the context of advanced QoS negotiation and will not be subject to further investigation in the context of this work.

English auction

The English auction is probably the most well-known kind of auction. Auction participants submit bids as long as a minimum of two competing bidders continue to bid. As soon as no bidder is overbidding the last submitted bid, the bidder with the final bid wins the auction. Moreover, the winning bidder has to fulfill exactly the conditions as specified in its bid (i.e. pay/charge the specified price). Along this basic definition of the English auction, there are a number of specific variants: auctions with general time-limits (e.g. eBay) or time-limits per bid or auctions with closed or open bids.

The English auction system has a number of advantages and disadvantages for all parties concerned. The so called Winner's curse may occur in such an auction due to strong competition with inexperienced participants carried away in the heat of the moment, which may favor the seller with a high price. On the other hand the item on sale can be bought for much less than its value, if bidding is slow, and rings can take advantage of the nature of English auctions. A general disadvantage for all involved participants is the fact that everyone must be in communication over the course of the auction, which can be expensive and difficult.

All the mentioned issues related to the English auction are negligible when applying this auction model in the context of the advanced QoS negotiation. The basic procedure of the English auction follows a closed round-based approach, where bidders submit bids in each round. Bids are submitted in a sealed or closed fashion, which implies that all bidders create their bids without any knowledge of the other bids. At the end of each round the highest bid is identified and the participants are invited to submit bids that exceed this bid in the next round.

In summary, the applicability of the English auction model in the context of the advanced QoS negotiation appears to be most promising. The auction-design complies with the round-based request-offer model. The involved service providers create their offers (bids) individually upon a client request without any influence from other service providers or their concrete bids. The client then assesses all bids according to a distinct assessment strategy and invites the service provider for another bidding-round. As soon as no more bids are submitted, the winner is declared by the client and finally a contract between the client and the winning service provider is exchanged.

Summary

The analysis showed that only the first-price sealed-bid auction model and the English auction model are applicable in the context of the advanced QoS negotiation, with the English auction model to be more-suited due to its round-based fashion. As a consequence, the advanced QoS negotiation support realized in the context of this work fully applies the closed-bid English auction model.

5.5 Security

This section details the security issues related in the context of Quality of Service. Usually security is considered as part of the QoS support, but it is realized separately and independently of the previously described QoS components. However, the motivation for a comprehensive security solution comes with applications that deal with confidential data (e.g. health data). As a consequence, the objectives of the security described here are to enable two network entities, which are usually a client and a service, to communicate in a secure manner and protect the exchanged potentially confidential information. This includes mechanisms for authentication, authorization and privacy, which are generally referred to as security facilities. In order to realize these facilities encryption algorithms, PKI (public key infrastructure) and TLS (transport layer security) are required as introduced in Section 2.4. Furthermore, compliance to Web services security standards is maintained.

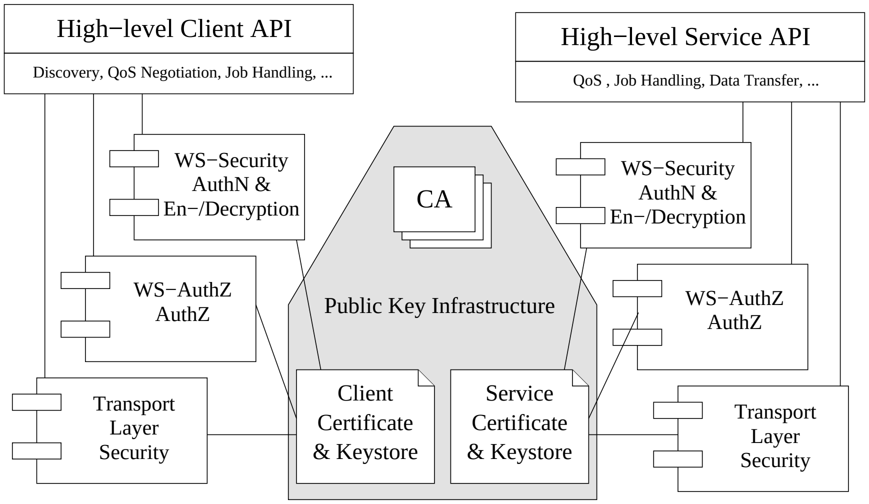
In the following the security approach is described in further detail. This sections starts with an overview that comprises the security architecture, the applied security protocol as well as the used standards and software. Subsequently, further details about the implementation of the security facilities are presented including authentication, authorization and encryption, as well as the mechanisms used therein.

5.5.1 Overview

The security overview outlines the security architecture, the security protocol, the maintained standards and the used software, which all incorporate into the realization of the SOAP message processing.

Security architecture

The security architecture is based on a X.509-compliant public key infrastructure (PKI) in order to issue certificates for clients and services by different trusted certification authorities (CAs). The certificates are utilized to ensure transport layer security as well as for authorization, authentication and privacy on the message layer (WS Security). All these security facilities are implemented in distinct components, which are used by according client- and service-side high-level APIs and its underlying components.



High−level Client API
Discovery, QoS Negotiation, Job Handling, ...
High−level Service API
QoS , Job Handling, Data Transfer, ...
WS−Security AuthN & En−/Decryption
CA
WS−Security AuthN & En−/Decryption
WS−AuthZ AuthZ
Public Key Infrastructure
WS−AuthZ AuthZ
Transport Layer Security
Client Certificate & Keystore
Service Certificate & Keystore
Transport Layer Security

Figure 5.18. Security architecture

Figure 5.18 illustrates the security architecture. A major objective of the architecture is to hide the complexity of the security mechanisms used from the client-user and service provider. This objective is achieved by encapsulating the security facilities in distinct components, which are utilized by a high-level API.

Security protocol

The security protocol comprises several levels of protection, representing again the mentioned security facilities. Each client-to-service communication starts with the authentication, continues with authorization, encrypts the message and finally delivers it through a secure transport channel. On the service-side the message is decrypted, authenticated and authorized, before the service is actually invoked.

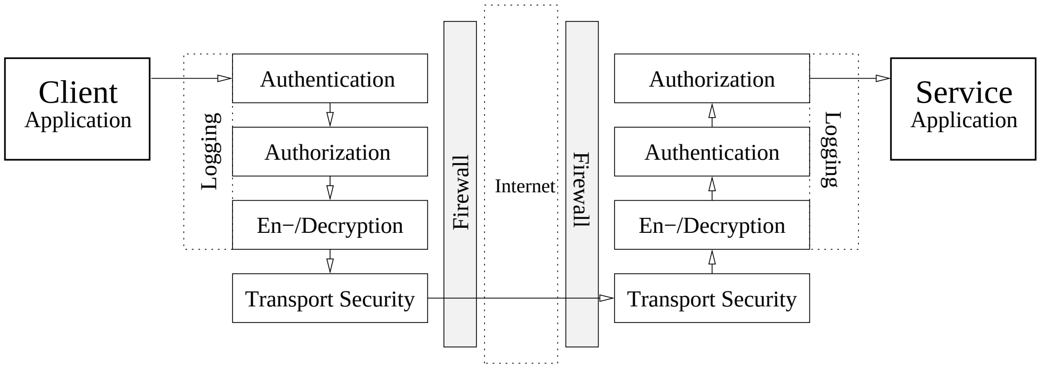


Figure 5.19. Security protocol

Figure 5.19 illustrates the security protocol including the entire stack of activities related to the security facilities involved in a secure communication between a client and a service. As the entire communication is initiated by the client, no compromises regarding service-side firewalls have to be made, because all mechanisms rely on standard Internet-protocols such as https.

Security Standards and Software

The security facilities are in line with security specifications and meta-data exchange formats such as WS Security [WS-Security], WS Trust [WS-Trust], WS Policy [WS-Policy], WS SecureConversation [WS-SecureConversation], extensible access control markup language [XACML] and the security assertion markup language [SAML] as well as transport layer security [TLS]. All these specifications are implemented by according software. The security standards and software stack is illustrated in Figure 5.20. The WS-* specifications are still evolving and the software used in this context is also under permanent development. Consequently, the entire security solution is adapted consecutively in order to meet arising new specifications and their standardization.

The security software includes RSM[6] and E2E[7], which are developments in the context of EU projects, in order to implement the according security specifications. These developments rely on open source software including Apache XML secu-

[6] RSM is a relationship management software that enables attribute-based access control (ABAC) with SAML tokens. It was developed in the context of the EU Aneurist project. http://www.aneurist.org/

[7] E2E is an End-to-End message encryption software that enables secure message exchange. It was developed in the context of the EU GEMSS project. http://www.gemss.de/

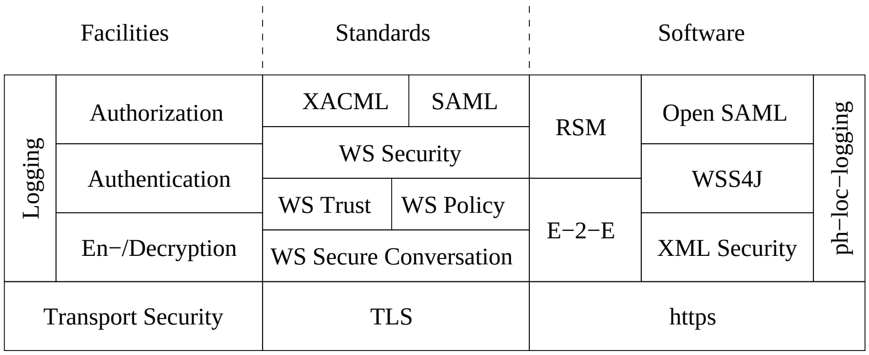


Figure 5.20. Security standards and software stack

rity implementing XML digital signatures and encryption [XML-Signatures] [XML-Encryption], Apache Web services security 4 Java (WSS4J)[8] and the open security assertion markup language (OpenSAML)[9] developed in the context of the Internet 2 project.

Apart from the handling of the security on certain levels the logging is also an essential facility of the security solution. In order to monitor and audit a service an appropriate logging system is required. In this security solution the Sourceforge phloc-logging system is used. This also comes with the benefit of a GUI-based administration.

Security Request-Response Processing

The processing of all the security facilities in the context of Web services is performed by Web service handlers. Each SOAP request and response passes a chain of handlers, which may modify the SOAP message according to its purpose. A SOAP message handler is subject to a distinct implementation which is dependent on the concrete Web services framework. In this case, the open-source Web services framework Apache Axis[10] is used. Each SOAP message handler realizes a certain functionality which processes an according security facility.

Figure 5.21 illustrates the security handler chain by distinguishing between incoming and outgoing SOAP handlers for the request and response message. The request creation is referred to as outgoing SOAP handler, the request processing as incoming SOAP handler to the service, while the response creation is an outgoing SOAP handler and the response processing at the client is an incoming SOAP handler. The classification with incoming and outgoing SOAP handlers also refers to the flow of

[8] http://ws.apache.org/wss4j/

[9] http://www.opensaml.org/

[10] http://ws.apache.org/axis/

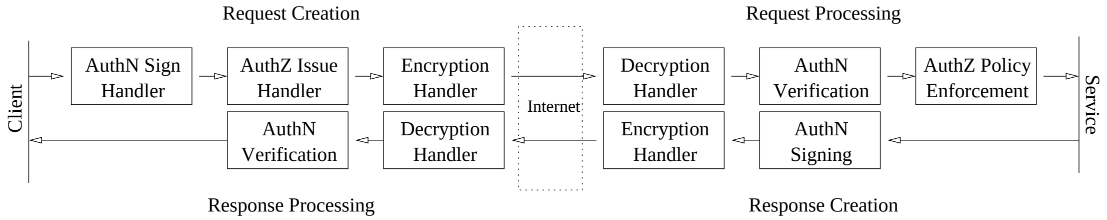


Figure 5.21. Security handler chain

the SOAP message, which may either be incoming or outgoing from/to a client or a service.

To the alert eye it will become apparent that the authorization facility is realized in handlers of the request-handling-chain only, because the response is associated explicitly to the request in RPC-based systems and does not need an extra authorization. In case of asymmetric SOAP calls no response messages will occur.

5.5.2 Authentication and authorization

Authentication and authorization are very closely related security facilities as introduced in Section 2.4.1, whereas a simplified definition of authentication is the verification process of the identity and authorization represents the decision whether to grant access to a resource or not. The basic approach of implementing both facilities is based on the usage of a security token service (STS), which is able to issue and verify signed security tokens with associated attributes. The signature of such a token is used to prove the authenticity (authentication), while the attributes are used in the context of attribute-based access control (ABAC, authorization).

In the following the security token service as well as the applied protocol in the context of authentication and authorization are detailed.

Security token service

A security token service (STS) is a separate Web service entity with an associated X.509 certificate that provides a WSDL-interface with operations to request and verify SAML-compliant security tokens. The issuance and verification of these SAML tokens is performed according to pre-defined trust relations to other STSes (i.e. their certificates) and with respect to the given attributes for specific users and/or services. The relations, attributes and certificates are stored in a LDAP directory service, such as openDS[11]. Generally, two implementations of the STS exist, one simple Web service

[11] Open Source Java LDAP Directory Service, http://www.opends.org/

only prototype, developed in the context of this work with limited functionality, and one full functional Web service implementation provided with the RSM software. The latter also comprises a Web interface for the management of the relations of associated users, services and their corresponding attributes, as well as its own trust relations to other security token services.

Usually, one STS instance per virtual organization (VO) exists, which implies that all users and services of the VO are managed in this STS, which is then run by an STS administrator. The relation of one STS to another is subject to the applied trust relation model and the used certificate (i.e. the issuing CA). The trust model can be either web- or hierarchy-based and the certificates of the STSes can be issued by the same or different CAs. In any way, the most common approach is, that either all STSes have certificates from a single CA and implicitly trust each other (hierarchical approach adopted in the simple STS implementation) or the certificates are issued by different (non-bridged) CAs and each STS has to explicitly trust another STS (web approach, adopted in the RSM STS). The latter approach comes with the advantage of no implicit trust, but also with the disadvantage of an increasing complexity of the relation-management as all STSes have to import certificates of all other STSes they want to be considered as trusted.

The applied STS trust relation is subject to a concrete application domain, such as the medical domain where certificates are issued by national e-health infrastructures and thus, only web-based models can be applied. Contrarily for research purposes and in closed projects, privileging all STSes with certificates from a single CA is sufficient and reduces management complexity. The concept of security token services with respect to federated authentication and authorization has been introduced in [*Iacono and Rajasekaran*, 2008].

AA protocol

The AA protocol (authentication and authorization) describes the procedure performed on each SOAP message delivered from a client to a service and vice versa. The actual AA protocol is implemented using appropriate SOAP handlers at both communication parties (c.f. request-response processing as introduced prior), which then utilize security token services as required. In the following, the SOAP message authentication and authorization are outlined:

1. Firstly the client requests a security token (i.e. a SAML token) from the STS the client is associated with by signing the request to the STS with his own certificate. This process assumes that the client is registered at the STS, i.e. the STS considers the client's certificate as trusted.

2. The client's STS issues a SAML token with its own signature to assure the authenticity and affiliation of the client to a certain relation. Furthermore, the

associated attributes (e.g. role) are confirmed by the STS and incorporated in the security token.

3. The client incorporates the signature and the relation along with the corresponding attributes of the STS in the outgoing SOAP request message header in order to prove its identity to the destination service and to be authorized as defined by the relation and the attributes.

4. The service receives the SOAP request message and firstly tries to verify the incorporated signature of the client's STS by requesting a verification of the signature at the STS the service is associated with. This assumes that the client and service belong to different security domains (i.e. different VOs).

5. The service's STS verifies the signature, if - and only if - the client's STS certificate is known and considered as trusted, and acknowledges the verification request of the service.

6. Given the service's STS acknowledgment, the service can be sure that the given SOAP request message is authentic and can be forwarded to the authorization.

7. The service-side authorization checks if the given relation and attributes (e.g. role) are allowed to perform the action as defined in SOAP message, which is actually the checking against a certain policy (policy enforcement). Only if this is acknowledged, the SOAP request execution is continued.

A simplified example of a SAML token that is incorporated in a SOAP message header is shown in Listing 5.1. In this example the SAML token states the *holder-of-key* confirmation method defined in the *SubjectConfirmation* element, which in general attaches constraints to an assertion. In this case the assertion is valid (within its lifetime of 15 minutes) as long as the possession of the private key corresponding to the public key certificate can be proved. This comes with the advantage that the STS is not required to perform access control for the security token issuance. Moreover, this implies that anyone can request a security token, but only the holder of the private key can make use of it in order to authenticate and get authorized by pairing the security token to the service request using the private key.

The usage of security token services for authentication and authorization comes with the main benefit, that the communication parties do not have to know each other in advance to be authenticated. Furthermore, both communication parties can have certificates issued by different CAs and no bridging etc. is required, the STSes only have to establish an implicit or explicit trust relation. The main benefit for the authorization is that the management of the attributes and the policy enforcement is subject to each VO and thus, no loss of access control occurs at each site. The entire management of authentication and authorization is performed in an abstraction layer, which is realized via the STSes of each organization.

```
<saml:Assertion Issuer="Institution-A">
  <saml:Conditions NotBefore="2009-01-01T12:00:00"
                   NotOnOrAfter="2009-02-01T12:15:00" />
  <saml:AttributeStatement>
    <saml:Subject>
      <saml:NameIdentifier NameQualifier="RelationshipName">
        <!-- Sample-Relation -->
      </saml:NameIdentifier>
      <saml:SubjectConfirmation>
        <saml:ConfirmationMethod>
          urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
        </saml:ConfirmationMethod>
        <ds:KeyInfo>
          <!-- Certificate of the user -->
        </ds:KeyInfo>
      </saml:SubjectConfirmation>
    </saml:Subject>
    <saml:Attribute AttributeName="Role">
      <saml:AttributeValue>Researcher</saml:AttributeValue>
    </saml:Attribute>
  <ds:Signature>
    <!-- Signature of the STS on "saml:Assertion" -->
  </ds:Signature>
</saml:Assertion>
```

Listing 5.1. Sample SAML token

5.5.3 Encryption

Encryption represents an important security facility in order to establish confidentiality on the message layer as introduced in Section 2.4. Generally, all data transfers with SOAP messages between communication parties such as a client and a service are considered to be highly confidential and have to be protected from eavesdropping using end-to-end security mechanisms. This implies that sensitive data (i.e. at least the SOAP message body) is encrypted all the way from SOAP message creation until the processing on the service-side. Relying on transport layer security only cannot ensure end-to-end security in case of intermediate hosts/servers.

Generally, hybrid encryption approaches are applied, which combine symmetric and asymmetric mechanisms, similar to the transport layer security protocol. TLS uses fast symmetric encryption for the communication and exchanges the secret key in advance encrypted with public-key-methods (c.f. TLS handshake, Section 2.4). This encrypted secret key exchange implies that the client has to know the destination service, in particular has to be in possession of a verifiable public key, in advance. This is true with TLS due to prior performed TLS handshake, but not in case of a SOAP message exchange, if the certificates of both communication parties are issued by different and unrelated CAs.

Similar to the implementation of the STS, two solutions are targeted in this context: key exchange utilizing security token services (STS) or additional key exchange operations (KEO).

Key exchange with STS

This approach assumes the presence of security token services with established trust relations with all other STS instances (implicitly or explicitly). Upon issuing a SAML token, the STS can add the certificate of the STS belonging to the targeted domain to its token issue response. The client (implicitly using an according SOAP message handler) generates a random secret key SK, encrypts the entire SOAP message body using this secret key and finally encrypting SK with the public key from the remote STS certificate. An example of a resulting SOAP request message is shown in Listing 5.2

```
<soapenv:Envelope>
  <soapenv:Header>
    <wsse:Security>
      <wsse:BinarySecurityToken>
        <!-- Certificate of remote STS -->
      </wsse:BinarySecurityToken>
      <xenc:EncryptedKey>
        <!-- SK, encrypted for remote STS -->
      </xenc:EncryptedKey>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <!-- Service request, encrypted with SK -->
  </soapenv:Body>
</soapenv:Envelope>
```

Listing 5.2. Sample encrypted SOAP request message

On the service-side the corresponding SOAP handler is not able to decrypt this message on its own, and thus it requests an encryption of the secret key (SK) with its STS. The remote STS is able to decrypt the secret key using its own private key and eventually returns the SK to the service. Given the secret key, the service, its SOAP handler respectively, is now able to decrypt the SOAP request message and continue the service processing. The SOAP response message is also encrypted using the secret key initially created by the client.

The major implication of this solution is that there is no ad-hoc security, because it relies on security token services, their established trust relation and the knowledge of the client's targeted service (i.e. remote STS).

Key exchange with KEO

This approach assumes that no security token services are used and secret key exchange is performed via additional key exchange operations (KEO) incorporated in the services. Furthermore, this approach assumes, that both communication parties have been issued with certificates from a single CA or CAs that have a direct trust-relation (e.g. via a bridge-CA) in order to be able to verify the certificate of the particular communication partner.

The additional functionality incorporated in the service comprises operations to exchange and verify the public keys of both parties as well as exchanging a public-key-encrypted secret key. The client (i.e. its SOAP handlers) firstly requests the certificate from the destination service sending along its own certificate. As the client's certificate is issued by a trusted authority (either the same as the service certificate or a CA with an explicit trust relation), the service is able to verify the client's certificate (authentication). A similar process occurs vice versa to enable the client to verify the service's certificate. Given the verified service certificate, the client generates a random secret key (SK) and sends it encrypted using the public key of the service's certificate. Now both communication parties are able to encrypt the entire SOAP message using the secret key. A reference implementation of the key exchange operations (KEO) has been developed with the E2E software, which has been successfully utilized in this context.

The main disadvantages of this approach are the limitations that both communication parties have to have certificates from the same or trusted CAs as well as the priorly established handshake (certificate verification and key exchange), which requires extra SOAP communication.

The used mechanism to enable encryption on the message layer is subject to the application domain. Generally, the first approach with STSes is more flexible (different VOs, enabling authentication and authorization), but comes with a complex management. The KEO-approach is rather simple, but is limited to a single CA or trusted CAs and provides authentication only.

5.5.4 Logging

Logging has been introduced in the context of auditing in Section 2.4 and it refers to the process of generating log messages (also known as audit records, audit trails, event-logs, etc.) of important events happening in IT-systems. The handling of the gained logging data (mostly log-files) is also referred to as log management, which aims to deal with large volumes of log messages by applying log collection, aggregation, long-term retention and log analysis. Furthermore, log management creates a foundation to enable accounting, auditing, intrusion detection or provenance, which are not further discussed here. In practice, logging frameworks are used, which support the logging itself and provide capabilities to improve the log management.

Web services are usually composed of a number of different components and 3rd party software, which may use different logging frameworks, which may result in a complicated log management. For example, determining the chain of events of a particular user interacting with a number of services may get complex with inspecting a number of different log-files at a single site, but also at different sites. Furthermore, most logging frameworks follow the common log file format that several Web servers use. The representation used in the common log file format is subject to configuration, while the actual items are related to web content and have their origin in the Unix operating system. The common log file format comprises the following information-items:

- **remotehost**: Remote hostname (or IP address if DNS hostname is not available, or if DNSLookup is turned off).
- **rfc931**: The remote logname of the user (ident).
- **authuser**: The username as which the user has authenticated himself.
- **date**: Date and time of the request.
- **request**: The request line exactly as it came from the client.
- **status**: The HTTP status code returned to the client.
- **bytes**: The content-length of the document transfered.

These information items can also be used in the context of Web or Grid services, but typically, more important context information such as user- or service information is not available.

The issues related to both the distribution of services and the unavailability of user- and/or service-context information are addressed in the logging subsystem used in the context of this QoS system. The logging system has been designed and implemented in order to support linking to other existing logging frameworks and to define arbitrary contexts to enable user- or service-specific logging. Furthermore, the entire logging system has been independently released as Sourceforge project phloc-logging[12] [*Helger*, 2008].

In the context of the QoS system, the logging has been utilized with a separate context for users and services. Furthermore, all other logging frameworks used such as Log4J[13], Commons Logging[14] or plain Java Util Logging have been linked to the phloc-logging system in order to provide aggregation of log messages based on users and services. This enables improved log management and forms a basis for accounting and other security facilities which are not discussed in this context.

[12] `http://sourceforge.net/projects/phloc-logging`

[13] `http://logging.apache.org/log4j/`

[14] `http://commons.apache.org/logging/`

5.6 Summary

This chapter presented a comprehensive overview of the Quality of Service support proposed in this thesis. The QoS support enables the on-demand provision of native HPC applications as QoS-aware Grid services, which constitutes an added value for service providers by potentially selling their services and for consumers by negotiating certain QoS guarantees in advance to the actual usage of a service on a case-by-case basis.

The description of the QoS system detailed the QoS support within a single service (micro QoS management) and the negotiation process between a client and one or more services (macro QoS negotiation). As a separate but related issue, this chapter also discussed the security infrastructure.

Chapter 6

Projects

This chapter highlights the practical use of the presented work by demonstrating its application in two different research and development projects. These projects have shown that the developed concepts and solutions can be applied in reality to improve research or enable the use of high-performance applications on a large scale.

In the following, the EU projects GEMSS[1] [*Benkner et al.*, 2005a] and Aneurist[2] [*Arbona et al.*, 2007] are presented. GEMSS dealt with Grid-enabling of compute intensive medical simulation applications by setting up a testbed with QoS-enabled Grid services exposing these applications. Aneurist developed an IT infrastructure in the biomedical domain, which similarly to GEMSS, aims to expose high performance applications as Grid services, but also distributed medical data.

Each project is introduced by presenting the major objectives, its context and related facts. Moreover, the contributions of this work to each project are described.

6.1 GEMSS - Grid-enabled medical simulation services

The research project GEMSS (Grid-enabled medical simulation services) funded by the European Commission has demonstrated the early adoption of Grid technology in health science. One of the main achievements of the project has been the provisioning of six medical prototype applications as Grid services in an according testbed. A corresponding Grid infrastructure has been developed considering the

[1] EU GEMSS project, `http://www.gemss.de/`

[2] EU Aneurist project, `http://www.aneurist.org/`

specific requirements of using medical services in a Pan-European Grid testbed such as legal constraints, business issues and Quality of Service.

The GEMSS project commenced in 2002 and successfully finished in 2005. The GEMSS consortium comprised eleven partner organizations from industry and academia including University clinics. The diverse medical prototype applications address maxillo-facial surgery simulations, neuro-surgery support, radio-surgery planning, inhaled drug-delivery simulation, cardiovascular simulation and tomographic image reconstruction [*Jones et al.*, 2004].

Subsequently, the major objectives of the GEMSS project, its scope and context as well as the major contributions of the presented work are discussed in further detail. General achievements of the GEMSS project are presented in [*Berti et al.*, 2003; *Benkner et al.*, 2004c].

Objectives

The main objective of the GEMSS project was to design and develop a secure Grid middleware for the on-demand provisioning native applications, in particular the mentioned six medical prototype applications, as Grid services. Moreover, enabling medical practitioners and researchers to use these Grid services in their clinical environment demonstrated new prospects to improve healthcare.

The key features of the developed GEMSS middleware have been on-demand provisioning of services, negotiable Quality of Service, flexible business models, and security to ensure privacy of patient data as well as compliance to EU law.

6.1.1 Scope and context

The GEMSS project as stated in the main objective was about designing and developing a Grid middleware addressing the health context, its specific requirements including QoS, business and legal issues as well as its diverse applications and their characteristics. This section briefly outlines the business context, the GEMSS applications, its legal constraints and the resulting required security.

Business Grid

The GEMSS project adopts a business Grid model, which corresponds to service providers offer services in an economic context and clients consume these services according to agreed market conditions. This implies that clients pay a certain price for the service consumption, but also that service providers have to make sure that the services behave according to a prior agreed contract (e.g. with respect to completion time or availability). For example, a surgery support application has to be available and deliver its results in an almost real-time during surgery.

Opposed to the business Grid approach, the traditional Grid model, which is also known as academic model due to its origin in academia, shares resources freely with a fair use policy. In the business Grid model, users do not provide resources in exchange for participation in the Grid; they are expected to consume services from service providers and pay for them. On the other hand, service providers are competitors, which implies that they do not distribute information about the availability of their resources, but wait upon client request to execute certain applications and propose a corresponding service level agreement.

GEMSS applications

The GEMSS medical prototype applications include maxillo-facial surgery simulations, neuro-surgery support, radio-surgery planning, inhaled drug-delivery simulation, cardiovascular simulation and tomographic image reconstruction. These applications typically comprise compute-intensive and parallel algorithms and tools, such as parallel Finite Element Modeling (FEM), parallel Computational Fluid Dynamics (CFD) and parallel Monte Carlo simulation.

The diverse GEMSS applications are targeting different challenges from distinct medical sectors:

Maxillo-facial surgery planning comprised pre-operative treatment planning of patients with in-born deformations of the mid-face. This application analyzes and simulates the deformations created by special screws tightly fixed to the head with a distraction device in advance to the maxillo facial surgery. The deformations become visible as a result of a parallel finite element analysis.

Neuro-surgery support aimed to predict the brain-shift-phenomenon during neuro-surgery. This is especially important to provide real-time coordinates for the surgical navigation and/or radiation. The actual computation is based on a precise parallel non-linear image registration algorithm.

Radio-surgery simulation addressed the treatment planning for cancer destruction considering different beam weighting and orientation. This approach differs from conventional treatment analysis and shows better coverage by utilizing monte-carlo simulation methods.

Inhaled drug delivery simulation dealt with virtual drug delivery to the lung. It utilizes a combination of computational fluid dynamics (CFD) and one-dimensional models to optimize delivery of inhaled drugs to the lung.

Cardio-vascular system simulation simulated the entire cardio-vascular system for improved treatment plans and surgical procedures by encompassing a compartmental approach, coupled structural mechanics and fluid dynamics.

Advanced image reconstruction improved fully 3D iterative image reconstruction for SPECT and MRI imaging by utilizing a hybrid MPI and OpenMP parallel OS-EM reconstruction algorithm. This application will be discussed in further detail as it is also utilized in the context of the experimental evaluation in Chapter 7.

The GEMSS Grid testbed exposed the mentioned applications as Grid services, which are hosted on HPC facilities such as PC clusters or other parallel computing platforms to facilitate their computational requirements. The GEMSS applications have varying requirements with respect to performance and QoS, but basically, a rather small number of time-consuming jobs has been tackled by the GEMSS testbed.

Each GEMSS application can be separated in a local interactive client component and a remote compute kernel, which can be executed in batch mode given according input data. This is required to facilitate a service oriented architecture.

Security and legal issues

A considerable number of legal issues and security requirements are linked to using new distributed Grid technology in healthcare. GEMSS aimed to examine both with respect to European regulations and their impact on the overall GEMSS system.

In the context of studying legal issues patient specific data processing in a distributed environment such as a Pan-European Grid testbed turned out to have significant impact. The investigations yielded that EU directive 95/46 applies to wholly or partly automated processing of personal data [*Herveg and Poullet*, 2003], which is the case in all six GEMSS applications.

The EU directive 95/46 defines among others a controller (legal representative, i.e. client) as the one responsible for personal data and a processor (i.e. service provider). The controller has to observe its associated personal data at all time and also has to have a legal contract with all potential data processors (services). Furthermore, upon each job an electronic agreement between the controller and the processor is required, which proofs that a service provider has accepted prior agreed legal responsibilities associated with processing of a certain computational job.

Technically, this situation implies the use of negotiable service level agreements (SLAs), while the legal requirement for all involved participants (clients and service providers) demands them to have a written and signed contract exchanged, before the first digital contact is made.

According to EU directive 95/46 the level of technical security required is determined appropriate to the risks represented by the data processing and the nature of the data. In order to address this, the state of the art and cost of implementation should be considered. Due to the very sensitive nature of personal health data, the level of security applied in the GEMSS system has been set to maximum given the practical costs of its implementation. This is also referred to as best practice security [*Middleton et al.*, 2005].

The security infrastructure in GEMSS consisted of using SSL on the transport layer (i.e. https) and Web services security (WS-Security) on the message layer. Therefore, an X.509-compliant public key infrastructure has been set up to enable authentication, authorization and encryption for users, code and services. The according certificate policy and the certificate practice statement of the GEMSS CA was in line with directive 1999/93EC, which specifies a community framework for electronic signatures.

Authorization in GEMSS was enforced for each service based on access rights that were associated with the individuals' certified identity. These access rights had been assigned prior in accordance to a specific business process and enforced on the service-level by the dynamic access control module (c.f. GEMSS architecture).

An end-to-end (E2E) security protocol has been developed and implemented to ensure message-level privacy, i.e. the message originator is authenticated and the message itself has not been tampered. The E2E security mechanisms are based on the Web service security specifications introduced in Section 5.5.

Architecture

The GEMSS middleware follows a service oriented architecture comprising multiple Grid clients and Grid service providers, one or more registries and a single certificate authority. Grid clients usually load the GEMSS client software, which enables them to use the GEMSS middleware, to interact with GEMSS services that are provided by service providers.

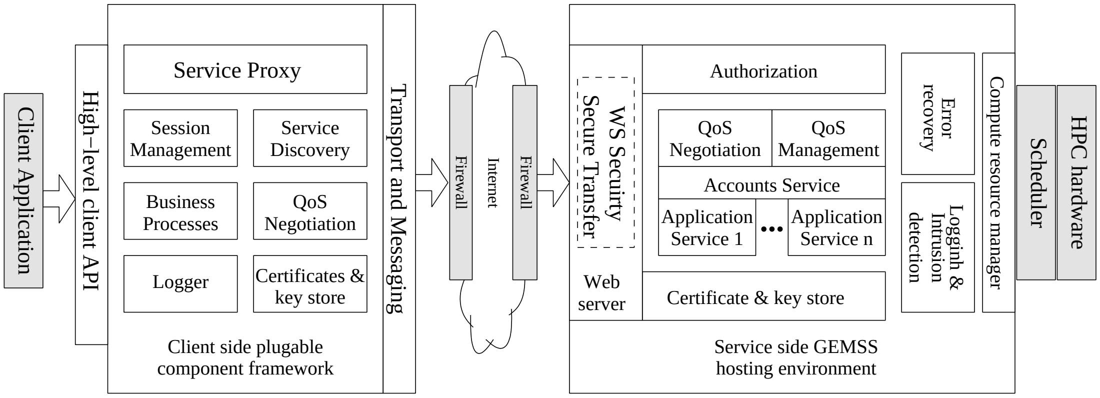


Figure 6.1. GEMSS architecture [*Benkner et al.*, 2004a]

Client applications are responsible to handle the creation of the service input data and present the service output data in a feasible way. On the other hand, service providers expose compute-intensive medical application kernels on clusters or other HPC hardware as Grid services to be consumed by Grid clients.

The GEMSS architecture is shown in Figure 6.1 comprising a client and a service infrastructure. Subsequently, both are outlined briefly.

The **client infrastructure** is based on a component model supporting pluggable GEMSS client components each realizing a certain functionality. On top of these client components and the component framework a high-level client API is provided, which is usually used by the Grid client application.

The **service infrastructure** exposes Web services that encapsulate native applications. Each application has to install its server-side application code and a set of scripts to be run in batch mode. The provision of applications as services is based on the concept of generic application services [*Benkner et al.*, 2004a]. A generic application service represents a customizable software component providing generic operations for remote job management, error recovery, and QoS support.

The interaction between the client- and service-side is performed securely via SOAP over https and with encrypted SOAP messages. The main processing to secure SOAP is being accomplished by the transport and messaging on the client-side and the Web Services security and secure transfer on the service-side.

6.1.2 Contribution

Certain contributions to GEMSS have their origin in the Vienna Grid Environment, but also vice versa. Major elements developed in the context of this work have been applied in the Grid infrastructure utilized in the GEMSS project, but also ideas and developments from GEMSS have been picked up in VGE. Generally, the GEMSS developments have been more focused on the medical domain's requirements and their applications, while the VGE developments pursued a more generic policy with no particular domain in the back.

The major contributions of this work presented in the previous chapters to the GEMSS project can be grouped by following areas:

- Generic service infrastructure
- Quality of Service
- Client components

The most innovative features of the GEMSS system especially with respect to QoS have been published in [*Benkner et al.*, 2005b, 2006, 2007; *Middleton et al.*, 2007]. Subsequently, the generic service infrastructure and the QoS support of GEMSS are being described in further detail.

Service infrastructure

As initially mentioned, the GEMSS architecture has been heavily influenced by the VGE architecture and vice versa. Consequently, GEMSS and VGE share common components of their rather similar service-oriented architectures. The second main influence of the GEMSS architecture mainly came from the GRIA project[3][*Surridge et al.*, 2005], which was also an early adopter of a service-oriented architecture. The architecture realization of GEMSS results in according infrastructures for the service- and the client-side. This work rather emphasizes on contributions to the GEMSS service infrastructure, while GRIA mainly affected the client-side.

The basic GEMSS service infrastructure has been adapted from corresponding elements of VGE service infrastructure. This implies that GEMSS services are also based purely on Web services technologies, which has proven to be forward-looking at the time when the project and its architecture design started.

Concerning the actual development the entire hosting and provisioning environment including its deployment tool and the concept of the generic application services have been adjusted to fit the requirements of the GEMSS services and its hosting. Components developed exclusively in GEMSS such as the accounting service or security components have been incorporated by customizing the existing service infrastructure appropriately.

The GEMSS service infrastructure has been setup and demonstrated successfully with a testbed running six different medical HPC applications as Grid service. Consequently, the applicability of the presented service infrastructure has been proven in the health domain.

Quality of Service

The Quality of Service support in GEMSS was required from the general setup of the project, in particular due to the special requirements of the medical domain and its involved applications. At the same time the QoS support was also one of the most challenging design and development areas of the entire project.

The basic Grid model of GEMSS follows a business concept with service providers offering services and clients consuming these services in a Grid market situation. This implies that clients and service providers have to establish an agreement concerning the actual service consumption and its linked conditions. As previously derived this was also an essential requirement originating from legal constraints of the medical sector.

In order to address the required QoS support in GEMSS the QoS support model developed in the context of this work has been adopted. In particular, the microscopic QoS management features including the entire service-side process of generating a QoS

[3] Grid resources for industrial application, http://www.gria.org/

offer and finally the confirmation which results in an according QoS contract have been utilized in GEMSS. Furthermore, building upon the micro QoS the macroscopic QoS negotiation relying on the English auction model has been extended with a client agent to be used in a fully automated way.

The **microscopic QoS management** supports the generation of a QoS offer based on initial constraints (QoS request) and request-specific information (request descriptor) as well as the confirmation of a certain QoS offer to become a QoS contract. Internally this process relies on advance reservation of resources, resource capacity prediction and flexible resource pricing models as explained in detail in Chapter 5. The reservation-based approach distinguishes significantly from other best-effort- or priority-based approaches, but it is the only solution to ensure the exclusive availability of the required resources in advance, which is strongly required by the GEMSS applications and its usage in the medical context.

A particular emphasis has been put on performance modeling of the involved applications to develop a feasible resource prediction of a specific job in advance to its actual execution. The experimental results of these investigations are presented in Chapter 7.

The **macroscopic QoS negotiation** represents a high-level client-to-service provider negotiation. This process comprises a client to request for and services to offer certain QoS guarantees in a competitive environment. Typically, a client negotiates with a number of services to achieve the best deal. Therefore, the client utilizes certain advanced strategies such an English auction. In a round-based fashion potential service providers are invited to underbid each other and in the end the winner of the auction is awarded with the execution of a certain application job. The finally agreed terms and conditions are determined in an according Web service level agreement (WSLA).

Given the general setup of the GEMSS project the overall QoS solution presented in the context of this work was very well suited. Furthermore, no alternatives have been available taking into account the special requirements of the diverse medical applications in a business context.

Further contributions

A number of contributions with respect to separate developments and/or the reuse of existing components to the GEMSS project can be found in the client infrastructure. The GEMSS client utilizes a service proxy component as basic abstraction of a remote application service, which has been adapted from VGE to serve as a GEMSS client component. Furthermore, other GEMSS client components realized certain functionality that has been originally developed in the context of the VGE client, including client components for service discovery, QoS negotiation, session management and logging.

Finally, more contributions of this work to GEMSS are rather hidden in various security-related developments. Generally, the entire security subsystem has been incorporated in the service infrastructure and, in particular, in its provisioning environment in order to integrate certain levels of protection as transparent to the user as possible.

In summary, manifold contributions of this work, particularly the service infrastructure and the QoS support, have been successfully applied and demonstrated within the GEMSS project, which serves as an essential proof of concept for these developments.

6.2 Aneurist - Integrated biomedical informatics for the management of cerebral aneurysms

The research project Aneurist funded by the European Commission has demonstrated similarly to the GEMSS project the benefits of using a complex Grid-based IT infrastructure on an even larger scale in the biomedical context. In particular, Aneurist and its developments are concerned with all processes linked to research, diagnosis and treatment of cerebral aneurysms. Although the focus of the project with respect to the medical context is on a one specific disease, the basic technologies are generic and transferable to other domains in healthcare [*Arbona et al.*, 2006].

The Aneurist project is a four year project commenced in 2006. The Aneurist consortium consists of 27 partner organizations in Europe and five collaborators from the USA, Japan and New Zealand. The partner organizations are from industry and academia including five clinical pilot centers supplying real patient data to the consortium for medical research purposes.

In contrast to GEMSS, which highlights complex computing, the medical data and its integration is emphasized in Aneurist. All medical data available in the Aneurist project is exposed through the IT infrastructure and utilized by four different application suites. The following capabilities are addressed by an individual application suite: personalized aneurysm rupture risk assessment (@neuRisk), design of smart implants to treat ruptured aneurysms (@neuEndo), knowledge discovery for linking genetics to disease (@neuLink), as well as integration of modeling, simulation and visualization of multimodal data (@neuFuse).

Subsequently, the major objectives of the Aneurist project, its scope and context as well as the contributions of this work are discussed in further detail.

Objectives

The Aneurist project addresses the general problem in healthcare that the process of disease diagnosis, treatment planning and development is compromised by the lack and/or fragmentation of relevant medical data. Although Aneurist is mainly a biomedical project, one major aspect to encounter this situation is utilizing state-of-the-art information technology.

A main objective of Aneurist is the design and development of a comprehensive IT system, which supports and consequently improves all processes linked to research, diagnosis and treatment development for complex and multi-factorial diseases, such as cerebral aneurysms. But the system is generic enough to be adapted to support the treatment of other diseases as well.

The Aneurist IT infrastructure consolidates heterogeneous data, computing and complex processing services in a distributed environment to be used across scientific and organisational boundaries. The infrastructure is created in line with evolving Grid and Web services standards and leverages existing developments from GEMSS and Fura[4] [*Arbona et al.*, 2007].

6.2.1 Scope and context

The main objective of the Aneurist project as concerned in the context of this work is the design and development of a generic IT infrastructure to support and improve the mentioned problems in healthcare. Therefore, four integrative application suites are selected to utilize and demonstrate the added value of the overall Aneurist system.

In the following these application suites are introduced with respect to their actual purpose and implementations as well as their targeted domain.

Aneurist application suites

The Aneurist application suites span from individual treatment and its planning to knowledge discovery based on population studies, but all share the common objective to considerably improve the understanding and management of cerebral aneurysms. In a more technical context, these applications have in common that they use the Aneurist platforms, which provide support for computationally demanding tasks such as complex modeling and simulation as well as access to health data distributed geographically in public and/or protected databases.

The Aneurist application suites include @neuFuse, @neuLink, @neuRisk, and @neuEndo, which rely on the Aneurist platforms comprising @neuCompute and @neuInfo. In the following these application suites and platforms will be outlined briefly:

@neuFuse provides an interactive open application framework for medical professionals and/or bioengineers to work with multimodal patient-specific data. This includes fusing diagnostic and modeling data into a coherent representation, visualizing different types of data using multiple display modalities and types, as well as simulating and processing based on the available data according to defined clinical processes. The underlying technology comprises compute-intensive image segmentation methods and multimodal registration algorithms, access to distributed sources of patient-specific information and advanced visualization methods.

@neuLink comprises a framework for knowledge discovery in the context of linking genetics to a certain disease such as cerebral aneurysms. For this purpose the application suite supports the identification of candidate genes associated with the disease phenotype as well as an integrated analysis of genetic epidemiology and clinical data. Therefore, it relies on the integration of and access to structured and

[4] Grid Systems Fura, `http://fura.sourceforge.net`

unstructured data from heterogeneous distributed data sources. The gained information is subject to advanced data and text mining in order to eventually identify candidate genes that are relevant in the context of managing cerebral aneurysms [*Friedrich et al.*, 2008].

@neuRisk enables personalized risk assessment in a decision support system which assists a clinician in assessing the rupture risk to determine if an aneurysm should be treated or not. This application relies on the collection of relevant data, including patient-specific data with @neuFuse and general data of similar patients and risk factors with @neuLink. Moreover, compute intensive methods are applied to calculate a personalized comprehensive risk associated to a specific aneurysm [*Dunlop et al.*, 2008].

@neuEndo supports the intervention planning and the design process with advanced computational tools towards the next generation of personalized smart flow-correcting implants to finally improve treatment of ruptured aneurysms. Within this application suite the planning and design of implantable devices is conducted by simulation of the structural, haemodynamic and biological response to intervention, considering patient specific characteristics. Hence according compute-intensive simulation applications as well as access to patient specific data is required.

The manifold application suites of Aneurist target different domains in the biomedical context, but as indicated in their individual outline, they depend on intensive computation and access to distributed, heterogeneous data. In order to address these requirements the Aneurist infrastructure comprises the platforms @neuCompute and @neuInfo to support compute intensive applications as well as access and integration of different data sources.

Aneurist platforms

Subsequently, the Aneurist platforms which constitute the essential components of the basic Aneurist infrastructure are outlined briefly:

@neuCompute provides an environment to set up compute services exposing native applications based on existing developments from GEMSS and Fura. Both software systems rely on standard Web services technologies to allow compliance with other Web services-based systems and to realize a service oriented Grid architecture. A major achievement of the GEMSS infrastructure as mentioned earlier is the Quality of Service support of complex HPC applications, while Fura comes with the benefit of improved support for parametric sweep-type applications relying on an agent-worker model.

@neuInfo supports accessing and integration of distributed heterogeneous data sources. The @neuInfo platform provides a generic framework that supports the provision and deployment of data services exposing a variety of health data sources. Internally, the platform is based on developments from OGSA data access and integration (OGSA-DAI)[5]and the Grid Data Mediation Service (GDMS) [*Wöhrer et al.*, 2005]. Furthermore, data services have the same interface as compute services in order to realize transparent access for the client to Aneurist platform services in general.

In the following both platforms and their provided services will be described in further detail in the context of the Aneurist architecture.

Architecture

A high-level layered view of the Aneurist architecture is shown in Figure 6.2. The architecture focuses on three layers - application suites, middleware and resources.

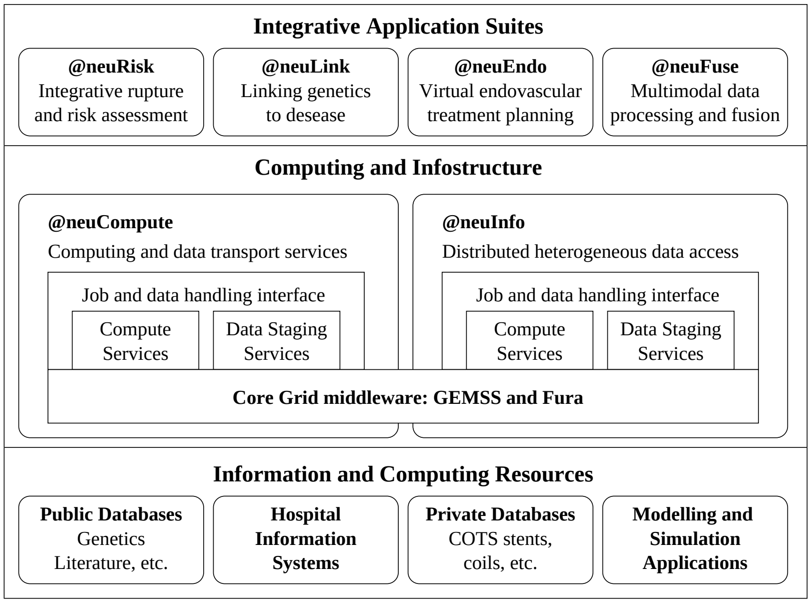

Integrative Application Suites
@neuRisk
Integrative rupture and risk assessment
@neuLink
Linking genetics to desease
@neuEndo
Virtual endovascular treatment planning
@neuFuse
Multimodal data processing and fusion
Computing and Infostructure
@neuCompute
Computing and data transport services
Job and data handling interface
Compute Services
Data Staging Services
@neuInfo
Distributed heterogeneous data access
Job and data handling interface
Compute Services
Data Staging Services
Core Grid middleware: GEMSS and Fura
Information and Computing Resources
Public Databases
Genetics Literature, etc.
Hospital Information Systems
Private Databases
COTS stents, coils, etc.
Modelling and Simulation Applications

Figure 6.2. Aneurist architecture

The top layer consists of the Aneurist application suites as outlined earlier, comprising multimodal data processing and image fusion with @neuFuse, linking genetics to diseases by @neuLink, integrative rupture and risk assessment with @neuRisk and virtual endovascular treatment planning by @neuEndo. Similar to other Grid architectures the middle and also middleware layer exposes various kinds of resources provided in the resource layer depicted at the bottom of Figure 6.2. The middleware basically enables clients (i.e. application suites) to transparently access resources without knowing the exact details of these resources (e.g. location). In the following the middleware and resource layers are discussed briefly.

The **middleware layer** comprises a service-oriented Grid middleware with data and compute services, exposing a variety of computation and information resources.

[5] OGSA-DAI, `http://www.ogsadai.org`

@neuInfo is basically concerned with providing abstractions to heterogeneous distributed data and @neuCompute provides computational Grid facilities to enable advanced modeling and simulation tasks. Both @neuInfo and @neuCompute are also referred to as @neuPlatforms, which provide the provisioning of Aneurist Grid services based on standard Web services technologies, i.e. these service are defined by the Web Services Description Language (WSDL) and securely accessed using SOAP.

The **resource layer** encompasses mainly computational and storage resources. Typically, computational resources embrace miscellaneous hardware (e.g. HPC facilities, such as PC clusters) offered by service providers in order to execute computationally demanding tasks, such as simulation or modeling algorithms. Storage resources are usually utilized by databases comprising simulation or patient data of various kinds including private and public data sources within and outside of Aneurist. Moreover, the clinical pilot centers offer patient data through a dedicated component of the Aneurist infrastructure named Biomedical Infostructure (BioIS). Internally, the BioIS will make use of clinical information systems (CIS) of the participating clinical pilot centers.

6.2.2 Contribution

The contributions to Aneurist gained from this work originated mainly from the GEMSS middleware. Even tough the developments that started in GEMSS have been extended and further improved, the main contributions of this work to GEMSS can be applied equally to Aneurist. Consequently, the well-proved service infrastructure from GEMSS with its key feature Quality of Service has been applied and demonstrated in Aneurist on an even larger scale as well.

The most considerable new capability of the Aneurist middleware in comparison to GEMSS is the incorporation of distributed heterogeneous data sources as data services relying on OGSA-DAI and GDMS. The contribution of this work in the context of data services was the adaptation of the existing application service infrastructure to accomplish the requirements and provision of data services as well.

Generally, the Aneurist system architecture comprises various innovative facilities besides the data and compute services and the most considerable issues have been published in [*Arbona et al.*, 2007]. Furthermore, the capabilities with respect to data access, integration and semantic mediation are far beyond the scope of this thesis, but can be found in [*Kumpf et al.*, 2007].

Chapter 7

Experimental Evaluation

This chapter comprises an experimental evaluation of the developed system with a particular focus on the Quality of Service support. A number of experiments have been performed which are presented in the following. The main experiments conducted address the micro QoS management and the macro QoS negotiation, both targeting an investigation of the system's behavior with respect to rationality and robustness.

Besides the major evaluation of the QoS support, a lot of different less considerable tests and investigations have been accomplished in the context of this work. These examinations are either performed to underpin the use of specific technologies and/or frameworks in advance to the actual implementation or to provide a proof of the general applicability of a certain software with respect to performance or security.

The most notable work related to evaluation has been made in [*Benkner et al.*, 2003a] to conclude the general use of the open source Web services framework Apache Axis due to performance and standards compliance, as well as in the evaluation section of [*Benkner et al.*, 2004b], which substantiates the applicability of Web services security mechanisms. A detailed presentation of these topics is beyond the scope of this chapter.

The experiments presented in this chapter have been published in [*Benkner et al.*, 2007] and [*Middleton et al.*, 2007] emphasizing the micro and macro QoS respectively. Therein the applicability and behavior of the QoS system is investigated with experiments at different scales. All experiments make use of a specific QoS-aware HPC application, which is presented in advance to the actual experiments. As a consequence, this chapter is structured as follows: First the underlying QoS-aware HPC application is presented, followed by the micro QoS management evaluation and finally the examination of the macro QoS negotiation.

7.1 SPECT application

All QoS experiments rely on a specific HPC application from the GEMSS project, which provides capabilities for advanced image reconstruction in the context of single photon emission computed tomography (SPECT). The SPECT application supports a fully 3D iterative image reconstruction algorithm for the whole image volume considering principal 3D effects of data acquisition. Internally a state of the art OS-EM (Ordered Subsets - Maximum Likelihood) algorithm is utilized, which is based on a stochastic model of Poisson-distributed generation and detection of photons. The work in the context of this application has been initially presented in [*Backfrieder et al.*, 2001] and refined consecutively in [*Benkner et al.*, 2002; *Backfrieder et al.*, 2002, 2003a, b].

Hybrid parallelism

Fully 3D iterative image reconstruction comes with the drawback of considerably increased demand of computing power compared to traditional 2D image reconstruction algorithms. Moreover, a sequential processing of a 3D iterative algorithm is not practical to be performed in a reasonable time and thus, a parallel approach has been utilized. The reconstruction kernel has been parallelized for an SMP clusters and achieved decent speedups as shown in Figure 7.1.

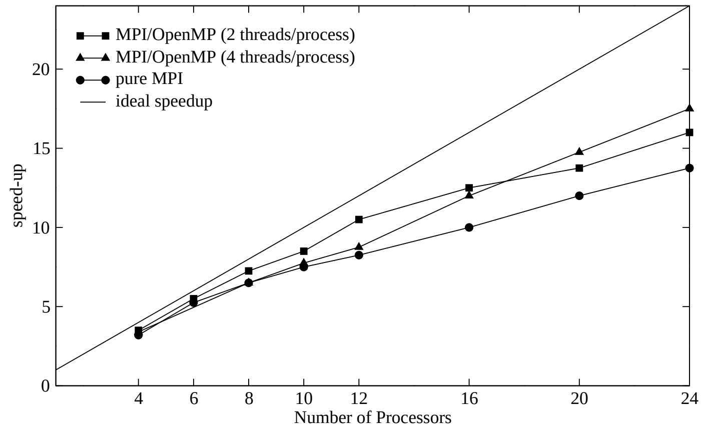


Figure 7.1. SPECT parallelization speedups [*Backfrieder et al.*, 2003a]

Please note that the SPECT parallelization strategy follows a hybrid approach utilizing MPI[1] and OpenMP[2]. Usually, MPI addresses communication between processes typically run on separate computing nodes while OpenMP realizes parallelism with threads and shared memory.

[1] Message passing interface, `http://www-unix.mcs.anl.gov/mpi/`

[2] Open multi processing, `http://openmp.org/wp/`

Application interface

The provision of this native HPC application as Grid service enables the use of advanced 3D image reconstruction software remotely within a clinical environment without relying on the operation and maintenance of according HPC hardware locally [*Benkner et al.*, 2003b]. Furthermore, this contributes to the long-term objective towards improving healthcare by utilizing novel IT infrastructures.

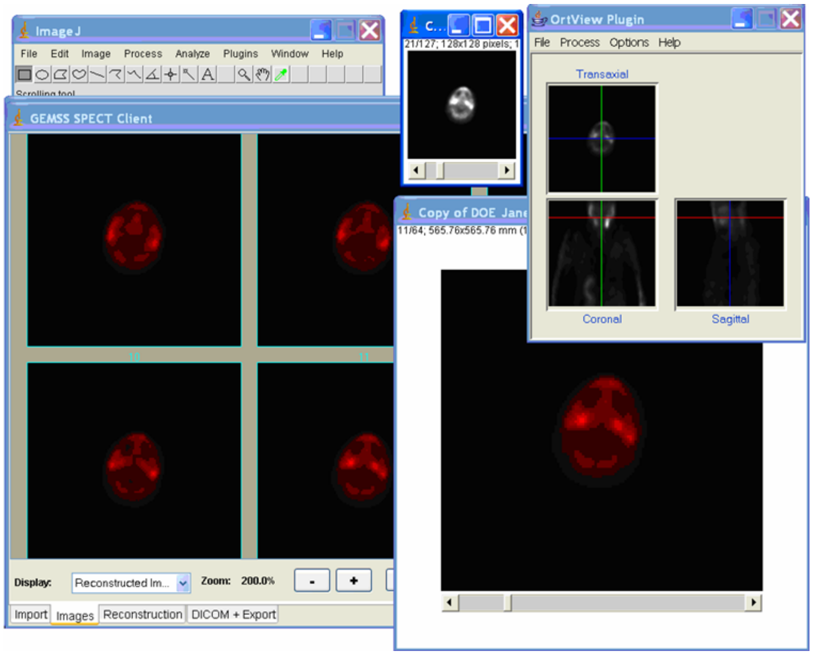

ImageJ
File Edit Image Process Analyze Plugins Window Help
Scrolling tool
GEMSS SPECT Client
Display: Reconstructed Im...
Zoom: 200.0%
-
+
Import
Images
Reconstruction
DICOM + Export
C...
21/127; 128x128 pixels; 1
Copy of DOE Jane
11/64; 565.76x565.76 mm (1
OrtView Plugin
File Process Options Help
Transaxial
Coronal
Sagittal

Figure 7.2. SPECT client interface

The screenshot depicted in Figure 7.2 shows the SPECT application interface from the client point of view (c.f. clinical environment).

Performance Modeling

Provisioning the SPECT application as QoS aware Grid service requires the availability of a performance model in order to predict the actual runtime of a SPECT job in advance to its execution. As discussed in Section 5.2.2 due to the very diverse characteristics of the native applications the QoS infrastructure does not prescribe the actual nature of a performance model. For the performance modeling of the SPECT application, an empirical approach is used to estimate the performance of the reconstruction kernel.

The actual code of the reconstruction kernel has been firstly structured in sequential and parallel blocks. Then example runs have been measured with respect to the required execution time down to each code block dependent on the input parameters used as well as the supplied number of CPUs. Given the required CPU time and the varying input parameters an according analytical model parameterized with the number of CPUs has been derived. The developed SPECT performance model estimated the total execution time for a specific SPECT job specified by its meta input parameters dependent on the supplied number of CPUs. Further on this SPECT performance model is used in all subsequent QoS experiments.

7.2 Micro QoS evaluation

The main objective of this experiment is to prove the rational behavior of micro QoS management. For this purpose services that expose the previously explained SPECT application are set up and stressed continuously by a number of clients over a certain period of time. The entire experiment is monitored and analyzed afterwards especially considering the individual components of the micro QoS management.

In the following the setup of the experiment, the used metrics and finally the results are presented and discussed.

7.2.1 System setup

The experimental setup comprises three service providers each exposing the SPECT application on a dedicated 16 CPU cluster. On the client-side ten different workstations are used to concurrently run the SPECT client and penetrate the available services. In order to cover a range of use-cases, these clients can select different SPECT jobs out of a pool of 24 jobs with varying job characteristics and input data, as listed in Table 7.1.

	Small	Medium	Large
Resolution	128	128–256	256
Projections	60	60–120	120
Slices	8–32	64–128	8–32
Iterations	5–25	5–25	5–25

Table 7.1. SPECT job characteristics

A SPECT job can be characterized by its resolution, the number of projections, slices and iterations. The resolution defines the horizontal and vertical number of pixels or dots per inch (DPI) of the projections (images) acquired from the CT or

MRI scanner. The concrete projections count is specified by the number of scans taken, which is usually a proportion of 360 degrees due to the CT/MRI scanner orbiting the compound. The slices determine the horizontal transsection to compute and finally the iterations define the numbers of successive increments the algorithm should execute. Furthermore, the jobs are grouped distinguishing small, medium and large jobs to reflect their computational requirements.

The characteristics of a specific SPECT job are supplied by the client during the QoS negotiation and then fed into the performance model as its main input. The SPECT performance model as described in the beginning of this section is parameterized with this meta-information about the concrete job as well as the number of CPUs. This implies that executing the performance model with varying numbers of processors determines the required and/or feasible number of processors to execute a certain SPECT job. The micro QoS management actually makes use of this possibility and eventually determines the number of processors that also meets the client's QoS constraints such as time and price. The concrete processing of the micro QoS management is subject to a configurable strategy as detailed in Section 5.3, but in the context of this experiment the prime time algorithm has been applied.

Nodes#	Small	Medium	Large
2	13	40	130
4	8	21	80
8	5	12	53
16	4	8	41

Table 7.2. Average SPECT job runtimes

Table 7.2 shows a list of average runtimes in minutes, that has been measured for all classes of SPECT jobs on 2, 4, 8, and 16 processors, respectively. The numbers also indicate a considerable well scaling behavior when increasing the number of processors the application is executed on.

Client tasks

The test infrastructure comprised three service providers and ten concurrently running clients as initially mentioned. Each client iteratively requests the execution of a random SPECT job subjected to a negotiated agreement until a total number of 400 jobs is exceeded. In particular, each client performs the following steps structured in three phases: environment setup, QoS negotiation, and job execution:

1. Environment setup

- Randomly pick up a job set (from the jobs presented in Table 7.1).
- Randomly define QoS constraints following one of three types of preferences: fast job execution, medium time job execution or just the execution with a long time constraint.

2. QoS negotiation

   - Query a single registry to obtain a list of service endpoints, which actually always returns all three available services.
   - Sequentially request an offer from a service provider randomly picked up from the list retrieved from the registry in order to equally distribute the requests of the clients to the available service providers.
   - Confirm the first matching offer to establish a QoS contract. If no offer is returned from any of the service providers, start over with a new environment setup.

3. Job execution (assuming a QoS contract has been established)

   - Upload input data and initiate the start operation, which actually just schedules the job according to the QoS contract.
   - Wait until the agreed start time and query the status iteratively until the application job runs
   - Similarly wait until the agreed finish time as well as the retrieved status indicates the job has been finished
   - Finally download the results, store the statistics of this job execution and start over with a new environment setup.

Each individual job run initiated by a client reflects user preferences with respect to QoS and the used job. More precisely, the job selection and the QoS preferences are determined in the environment setup of each client run. In order to cover a wide range of users, jobs and preferences, this experiment comprises three different test runs with clients randomly requesting small, medium and large jobs and also randomly selecting QoS constraints. The job size is characterized in Table 7.1 while the selected QoS constraints reflect three kinds of user groups: the first with a high priority for a fast job execution, one group with medium time constraints, and finally one group with no hard time constraints.

Given this setup, all ten clients have been started concurrently until a total number of 400 jobs has been exceeded. In the following the measured metrics are defined before the actual results are presented.

Metrics

The metrics used in the course of this experiment include robustness, throughput, utilization, and performance model accuracy. The robustness has been determined by the percentage of successfully run jobs and the throughput by the number of jobs per hour. The utilization has been averaged using mechanisms from the scheduler monitoring the utilization every 15 minutes. Finally, the performance model accuracy has been recorded by comparing the estimates against the actual run time.

In summary, the following metrics have been measured in this experiment:

- Robustness = % of jobs successfully run
- Throughput = Jobs run per hour
- Utilization = Average % node use reported by the cluster
- Performance model accuracy = (estimated runtime - actual runtime) / actual runtime

7.2.2 Results

The comprehensive results of this experiment are shown in Table 7.3 primarily distinguishing between small, medium and large jobs. The first line represents the total uptime of the test infrastructure to complete the number of requested jobs given in hours. The second part of the table presents the actual numbers of jobs totally requested, rejected, resulted in contracts or errors and finally succeeded. The bottom part comprises statistical information as stipulated by the defined metrics.

		Small jobs	Medium jobs	Large jobs
Uptime	hours	8,4	25,7	21,4
Jobs	requests	401	401	401
	rejected	0	1	319
	contracts	401	400	82
	errors	5	3	4
	successes	396	397	78
Stats	Robustness	98.75%	99.25%	95.53%
	Throughput (Jobs/h)	46.94	15.42	3.66
	Av. utilization	60.01%	68.82%	88.84%
	Av. PerfModel Acc.	97.28%	97.62%	97.57%

Table 7.3. Micro QoS experiment summary

7.2.3 Analysis

Generally, this experiment delivered excellent news with the system remaining stable under heavy stress conditions over a longer period of time. The infrastructure works considerably robust with handling most of the requests properly and transparently to the user. This includes coping with a number of errors related to general network timeouts or failed negotiations, which have been compensated by retries or balanced by other service providers respectively.

The very few unhandled issues that arose and eventually resulted in errors can be classified in two kinds of problems. The first problem was related to network and resource delays and consequently timeouts in a multithreaded environment when multiple clients tried to access the same resource. More precisely, this kind of malfunction appeared when multiple threads were attempting to write on a certain resource (e.g. resource model organizing the QoS offers and contracts), while other threads wanted to read from this resource. As writing threads have priority, delayed reading threads may finally cause network timeouts. A pragmatic solution would be the fine-tuning of the timeout settings.

The second problem was related to a security issue on the client side. As presented in Section 5.5 each client-to-service communication is encrypted using a security token that is exchanged in advance. Each security token has a short validity and must be renewed in the concourse of multiple invocations of service operations. This is usually the case even a couple of times in the long lasting job handling phase (c.f. setup). In the unlikely case that the security token expires during a specific invocation of a service operation (e.g. on a considerably long lasting up- or download), the client parses the actually regular response message from the service, but is under the impression that the message is compromised. A potential solution to this would be to accept expired security tokens on incoming SOAP responses if the associated outgoing SOAP request has been within the security token validity.

Besides the robustness of the system the throughput has been measured, which is obviously higher and consequently faster with smaller jobs compared to bigger jobs, which demanded more time to complete. Furthermore, the average utilization has been recorded, which fairly varies but correlates to the number of requested jobs and the number of actually started jobs. In case of small jobs all requests have been handled by the system and the total utilization stayed on a level below the high utilization of the medium and large jobs. Especially in the large job case a lot of requests have been rejected due to capacity reasons. In any way the throughput and utilization also underpins the rational behavior of the system.

Finally, the accuracy of the SPECT performance has been recorded, which has been considerably high, verifying that the SPECT application is very well suited to this reservation-based QoS approach. Furthermore, the statistical significance is proven by the standard deviation for these figures varying between 0.013 and 0.014 over the entire 1203 jobs.

Despite these very few problems the experiment proved a very satisfying behavior of the overall system and provided evidence of its rationality and robustness.

7.3 Macro QoS examination

The main objective of this section is to investigate the system with respect to its macro QoS negotiation capabilities by performing distinct tests. These tests should provide an insight on how the system actually works on the macro QoS level with a particular focus on the used pricing model. Due to the support of flexible pricing models by the QoS infrastructure a Grid marketplace for a specific medical application exposed via services can be simulated. The main implication in doing so is that service providers create offers for using their services based on different pricing models.

Two separate tests have been devised considering the effects of pricing in an isolated fashion as well as factoring in the actual execution time. In general it should be noted, that various limiting assumptions have been made to keep track of complexity while conducting these tests. Both tests are further on described with respect to their setup and the applied metrics. Finally, the results are presented and discussed.

7.3.1 System setup

Similar to the tests performed in the context of the micro QoS evaluation the experimental setup comprises three different service providers each exposing the SPECT application on a dedicated 16 CPU cluster. The client-side is slightly different by just using a single client, which continuously drives the macro QoS negotiation as described in Section 5.4 with all available service providers.

For the sake of simplicity, the client iteratively requests the same SPECT job execution. This particular SPECT job lasted for 45 minutes using 16 CPUs (i.e. the entire resources of one service provider) to finish and can be characterized as a large type job as specified in Table 7.1. Furthermore, the client only submits a total of nine jobs to keep track of the overall system within a reasonable time of 135 minutes. An observation with extending the total time of the job executions would not deliver any new insights and moreover, the uptime capabilities of the system being stressed with a lot of jobs has already been demonstrated in the micro QoS evaluation.

General SPECT job attributes as well as SPECT performance modeling and its implications have also been discussed and analyzed in the micro QoS evaluation and eventually proved to be applicable. Hence this functionality is just utilized in this experiment, even if certain capabilities such as how many CPUs should be used, are practically turned off due to the use of constantly 16 CPUs. The main reason for this limitation is to isolate the macro QoS negotiation in the best way.

Client tasks

Actually, two experiments are conducted for the macro QoS negotiation evaluation differing from each other just in the way the client assesses the offers returned from the services. More precisely, the client awards a specific service provider that wins an auction and the winner is determined by different assessment strategies. One is just considering the price and the other is factoring in the execution time as well. In particular, the client performs the following steps:

1. Client environment setup (static job input and QoS constraints)
2. Invitation of all available service providers to participate in an auction about the execution of the specified SPECT job
3. The auction is round-based driven by the client and as long as service providers improve their bids according to the following assessment, the auction continues and finally the best offer wins the auction.
   a. Assess cheapest offer as best offer.
   b. Assess all offers with equally considering price and time.
4. The winning offer of a single service provider is finally confirmed by the client.
5. Job execution is equally performed as with the test carried out for the micro QoS evaluation.

These steps are performed consecutively with the client expressing its preferences in the assessment of the service offers. The price-only assessment is achieved by specifying the QoS constraints with a large time window; reflecting start and end time constraints are irrelevant and analyze the pricing models and their impact on the system's behavior in an isolated fashion. In the second test the client sets up the environment with QoS constraints that consider time and price. This allows the service provider to schedule jobs where they see the best fit within the specified QoS constraints.

Pricing models

The service providers in this setup used different pricing models including two fixed pricing models and a dynamic pricing model considering the utilization. The following pricing models have been used:

1. Price model A: Fixed price = € 0.6 per CPU hour (medium)
2. Price model B: Variable price = € 0.4 per CPU hour + € 10 * <*system load*>

3. Price model C: Fixed price = € 0.8 per CPU hour (high)

The *system load* in the context of this test is a fractional value, which is increased by one third for each new job that is agreed to be executed by the service provider following the dynamic pricing model. This simplification can be made due to the total runtime for each test of 135 minutes, which corresponds to the execution of nine jobs, whereat each service provider is able to handle three 45 minutes jobs and consequently, each service provider's load is increased by one third with each job.

Given this setup the client was started and kept running until a total number of nine jobs was reached. In the following the measured metrics are defined before the actual results are presented.

Metrics

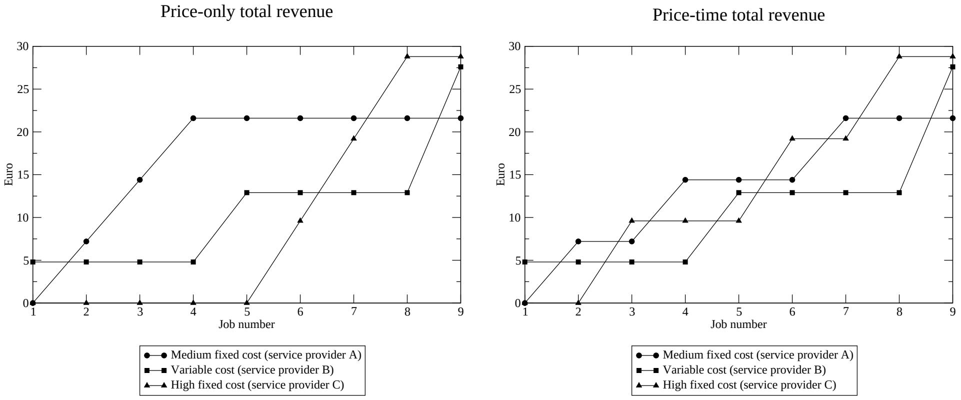
The main metric considered in this experiment is the service provider's revenue, which sums up the agreed prices from the QoS contracts established between the client and a particular service provider. The revenues of the service providers are contrasted with respect to the different assessment strategies of the client. Following the revenues, a mapping of the actual jobs to the service providers is also shown.

7.3.2 Results

The accumulated revenue for each service provider can be seen in Figure 7.3 distinguishing between different client offer assessments. Basically, both graphs show the increase of the revenues in Euro of each service provider with each new job. Furthermore, it can be seen how the different pricing models influence the revenues and, in particular, how the dynamic pricing model is affected by the system load.

The left graph shown in Figure 7.3(a) presents the revenues with totally isolated pricing, which means that the client always chooses the cheapest offer. This test aimed to demonstrate that the client behaves rationally and always chooses the lowest price first. As expected, the low price model grabs the early jobs, while given constant demand, the clients are forced to pay higher prices later on. Furthermore, it can be seen that the price of the variable model increases with the system load and becomes less attractive.

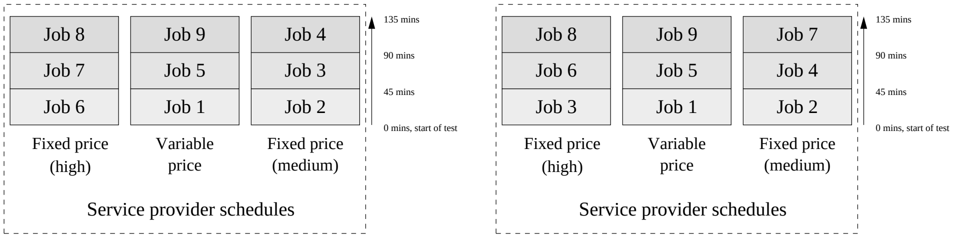
The second test is presented in Figure 7.3(b). In this test the client factors in the scheduled timeframe of the application execution equally weighted to the price in the context of the offer assessment. The main objective of this test was to examine potential differences to the first test with given resource limitations. The general expectation is to see a more balanced load between the service providers. The first jobs will be quick and cheap, but at a certain point the fast but expensive will outplay



(a) Offer assessment of price only (b) Offer assessment of price and time

Figure 7.3. Total revenue in Euro from each service provider [*Middleton et al.*, 2007]

the slow and cheap offers, because the execution time is now equally considered as the price in the client's offer assessment.



(a) Offer assessment of price only (b) Offer assessment of price and time

Figure 7.4. Service provider job schedules [*Middleton et al.*, 2007]

Finally, the job schedules for all service providers distinguishing again both client offer assessments are shown in Figure 7.4. This view of each service provider's reservation schedule can be concluded from Figure 7.3 as well, but it presents an explicit schedule depicting the order of the jobs in each service provider.

7.3.3 Analysis

The basic results of the performed macro QoS negotiation tests indicate that the system apparently follows rational behavior. The job offers with the lowest prices will always be awarded first as long as no other factors such as end time of the job are considered (c.f. price-only assessment). If the end time of a job is taken into account it may significantly degrade the assessment and make the price overall less important (c.f. price-time assessment).

The success a service provider achieves in terms of the best revenue is strongly dependent on the level of demand. Given a low level of demand service providers will be most successful with offers at a low price in order to obtain a sales volume at all. This can be underpinned by the situation depicted in Figure 7.3(b), where even after half of the jobs have been submitted, which is equivalent to a low demand, the high price service provider finds itself with no jobs and no revenue so far, since the cheaper providers are chosen first.

On the other hand, if the level of demand is high and clients have less choice due to lack of free resources with the cheap providers, they have to use the remaining service provider, which offer only high price jobs. This situation is shown by the second half of the submitted jobs in Figure 7.3(b). The clients are forced to choose the high price service providers and thus, charging such prices is sensible in a high demand situation.

The macro QoS negotiation only considers the price deals with a low and high demand condition as expected, which is also in line with the economic basics of supply and demand.

The second test depicted in Figure 7.3(b), which limited the resource allocation with hard time constraints, showed a more balanced load between the service providers. In this situation the clients actually acted similarly, but had to choose the high pricing service providers earlier in order to achieve their time constraints. Again, if a low demand for a certain time frame exists, low price providers will gain profits first, but with increasing demand also a high pricing policy is feasible.

The key issue to success in different market situations is flexibility. The optimal pricing model would be able to calculate the price based on the current demand, which can be concluded from the systems' utilization. The variable pricing model aims to simulate such an adaptive price, which allows the service provider to be awarded with some early cheap jobs to gain an initial utilization and exploit a potential increasing demand with high prices later on.

In summary, a lot more complex pricing models and investigations are imaginable, but for the sake of coping with the complexity of the system, these tests are kept rather simple, but even on this level the basic economic rules of supply and demand can be verified. Furthermore, this provides evidence for the proper and rational behavior of the macro QoS negotiation.

7.4 Summary

This chapter presented an experimental evaluation of the developed system. The Quality of Service support has been investigated by conducting different experiments to address the micro QoS management and the macro QoS negotiation. All experiments proved the robustness and rational behavior of the system. Moreover, from an overall viewpoint the use of the micro and macro QoS appears to complement each other well.

Chapter 8

Related Work

Manifold related research work has been presented in all the chapters so far, focusing explicitly on the individual detail at hand. Contrarily, this chapter surveys further general work mostly in terms of related projects that deal with similar aspects as this work does or reside in the connatural life science domain as the presented projects in Chapter 6. The main aim of this chapter is to extend the prospects to research activities and projects relevant to this thesis, which have not been mentioned or outlined so far.

This chapter is structured comprising the following parts, each highlighting relevant research and/or projects as well as briefly addressing their objectives:

- Grid infrastructures
- Grids and business
- Medical Grids
- Quality of Service

Subsequently, each area is being detailed.

Grid infrastructures

The increasing number of production level Grid infrastructures utilize heterogeneous middleware, such as Globus, gLite, or UNICORE. Due to many standardization efforts a common consensus has been achieved in the adoption of service-oriented architectures and, in particular, the use of Web services technologies to interface different middleware services. In the following the core infrastructures and their deployed middleware is briefly outlined.

The Globus toolkit[1] has been mentioned in Section 2.2 as an initial effort of assembling a set of tools towards the vision of Grid computing [*Foster et al.*, 2001]. The Globus toolkit has been promoted by the Open Grid Forum (OGF)[2], which was formerly known as Global Grid Forum (GGF). The OGF community also envisaged the trend towards service oriented architectures and supported the transition of the Globus toolkit to use Web services technologies recently. Globus also constitutes the basis for more recent developments such as the gLite middleware.

The gLite middleware is used in the Enabling Grids for E-sciencE project (EGEE)[3] [*Laure et al.*, 2006], which is Europe's leading research cyber-infrastructure project. The gLite middleware follows a service oriented architecture and exposes five major groups of services, including services for data and job management, security, information and monitoring. These groups of services are similar to the vision of OGSA as introduced in Section 2.2, which allows compliance and integration of other infrastructures built upon the same principles.

The UNICORE (Uniform Interface to Computing Resources)[4] middleware is also used in a number of Grid projects mentioned later in the context of medical Grid projects. UNICORE also adopted Web services technologies and provides distributed computing and data services as well as client access to these services in a seamless and secure way.

Grids and business

The business aspect in Grid computing has been underestimated with respect to the adoption of Grid solutions in industry. This has often been explained by the academic world with the lack of standardization or similar issues, mostly reduced to technical reasons, in comparison to the economic evolution of Web computing. This also might have strived the advent of Cloud computing and its encouragement by the major players in the IT industry at the end of the 2000s, which adopted a pay-as-you-go approach. In contrast, Grid computing more likely employed an academic model which foresees to contribute to a common pool of resources and then being authorized to use the resource pool freely on a fair use policy.

A lot of research collaboration and projects have implemented the academic resource sharing model. The large **US TeraGrid**[5] and the **EU DataGrid** [*Laure*, 2004] (with its successive already mentioned EGEE project) are just two major examples, which have their origin in the USA and EU respectively, but are globally distributed today. A common ground of most of these academic Grids is the use of

[1] Globus, `http://www.globus.org/`

[2] Open Grid Forum (OGF), `http://www.ogf.org/`

[3] EU EGEE, `http://www.eu-egee.org/`

[4] UNICORE, `http://www.unicore.eu/`

[5] US TeraGrid, `http://www.teragrid.org/`

the Globus toolkit[6] to share computing resources. On the other hand, several projects have proposed economy-based Grid systems on top of these systems

Contrarily to the academic model, [*Buyya*, 2002] proposed a **Grid Architecture for Computational Economy (GRACE)** with a distributed resource management and scheduling for Grid computing following business principles. The developed system provided management for resources controlled by the Nimrod-G resource broker to be allocated with parametric sweeping applications based on price, time and availability of resources as well as allowing to fix either cost or time while optimising the other. This work provided a major step forward by modeling resource allocations in the Grid considering economic aspects. However, GRACE relies on centralized and trusted components to collect information about the availability of the resources as well as to allocate them which basically conflicts with the vision of the Grid.

Further business-oriented Grid projects, which rather follow an application service provider (ASP) model, can be found with the GRIA and GRASP project. The industry driven **EU GRASP** project[7] [*Dimitrakos et al.*, 2003] aimed to explore the use of Grid services with respect to their service provision in enterprises. At the core infrastructure of the GRASP project Web services have been utilized and basic service level agreements have been investigated. The basic approach was to pool resources of service providers and manage them by a virtual organization (VO) in order to offer stronger service level guarantees or cheaper prices than a single service provider. However, the members of the virtual organization have to trust the VO and cannot easily compete against each other. As a consequence, the full benefits may not be seen by customers.

The **EU GRIA** project[8] [*Surridge et al.*, 2004], which initially was an acronym for Grid resources for industrial applications, aimed to make the Grid usable by industry. The basic approach of GRIA was to employ a business-to-business (B2B) service provision model that connects consumers and providers directly. This client-to-service link is also expressed in according service level agreements, which include the service capability and its price, but no concrete guarantees about execution time. The **EU GEMSS** project[9], which has been introduced in Section 6.1 reuses some developments regarding security, but the applications and their requirements in GEMSS differ substantially from the ones addressed in GRIA and thereof the business and QoS model is also different.

Medical domain

Medical applications are considered as the potential ”killer applications” for the Grid and thereof the life science domain gained an increasing importance to Grid

[6] Globus, `http://www.globus.org/`

[7] EU GRASP, `http://eu-grasp.net/`

[8] EU GRIA project, `http://www.gria.org/`

[9] EU GEMSS project, `http://www.gemss.de/`

computing in recent years. This situation is also reflected in the growing number of research projects which deal with certain aspects in the field. In the following some of these projects are outlined briefly.

The **EU BioGrid** project[10] [*Bala et al.*, 2002] aimed to develop a Grid infrastructure for biomolecular applications and databases utilizing HPC facilities. This enables chemists and biologists to advance data-intensive and computational demanding biotechnological procedures. The Grid infrastructure of the project is based on the UNICORE middleware.

Also based on UNICORE is the **EU OpenMolGrid** project[11] [*Romberg et al.*, 2007] which aimed to develop and utilize a Grid infrastructure to solve molecular design and engineering tasks in chemistry, pharmacy and bioinformatics. A particular emphasis in this context was the integration and access of distributed data sources. The **EU MammoGrid** project[12] [*Warren et al.*, 2007] was also mainly concerned with the data access and integration aspect in the Grid. Its emphasis was obtaining and providing information about breast cancer via an according Pan-European Grid data infrastructure.

The **EU BioinfoGrid** project[13] [*Milanesi*, 2007] constituted a specific support action (SSA) that fosters a collaboration of Bioinformatics with Grid infrastructures. Therefore applications from molecular biology have been enabled to run on the EGEE Grid infrastructure. The project focused on the investigation and evaluation of genomics, proteomics and molecular dynamics applications relying on EGEE Grid technology. A number of other projects are also based on the wide-spread EGEE cyber-infrastructure, most notable for the medical field is the **EU WISDOM** initiative[14] [*Jacq et al.*, 2007]. This project utilized the EGEE Grid infrastructure to demonstrate its capabilities in drug discovery for diseases like Malaria or avian influenza in in-silico experiments.

The **EU ViroLab** project[15] [*Gubala et al.*, 2007] aimed to realize a virtual laboratory for the study of infectious diseases such as HIV. Medical doctors may use the virtual laboratory to understand individual drug resistance and tailor personalized drug therapy for a specific patient, while virologists are supported in studying virus trends on accumulated patient data. The software used in the virtual laboratory is able to utilize different infrastructure services such as Web services or Globus-based Grids.

The **UK myGrid** project[16] [*Pettifer et al.*, 2007] is a large initiative generally concerned with the improvement of e-science by many kinds of tools and core infrastructures. With respect to a broader medical domain, myGrid provides support for

[10] EU BioGrid, `http://biogrid.icm.edu.pl/`

[11] EU OpenMolGrid, `http://www.openmolgrid.org/`

[12] EU MammoGrid, `http://www.cems.uwe.ac.uk/cccs/project.php?name=mammogrid`

[13] EU BioinfoGrid, `http://www.bioinfogrid.eu/`

[14] EU WISDOM initiative `http://wisdom.eu-egee.fr/`

[15] EU ViroLab, `http://www.virolab.org/`

[16] UK myGrid, `http://www.mygrid.org.uk/`

data intensive in silico experiments such as analysing protein sequences and structures. The core middleware of myGrid is based on Web services, which are accessed for example using the workflow design and enactment environment Taverna or the myExperiment research collaboration platform.

The **US cancer Biomedical Information Grid** (caBIG) project[17] [*Buetow*, 2005] provides a network for the cancer community to collaborate by sharing data and knowledge via the IT infrastructure caGrid. This open cyberinfrastructure strives to achieve computational and semantic interoperability by relying on a service oriented architecture and building on existing technologies such as Web services and OGSA-DAI, similar to the myGrid project.

The US **Biomedical Informatics Research Network** (BIRN)[18] [*Jovicich et al.*, 2005] promotes a large virtual community with the major objective to advance diagnosis and treatment of diseases. The initiative is driven by the US National Institute of Health providing a framework to foster the installation of an IT infrastructure, which notably is a Grid of super-computers. This enables distributed collaboration and computation on large-scale studies in biomedical sciences. The medical focus lies on brain imaging of neurological disorders and thus, many different diseases are addressed including depression, multiple sclerosis, brain cancer, Parkinson's disease, Tourette's disorder or ADHD.

Most of the projects presented so far rather focus on the data management aspect in the Grid, which is fairly similar to the further discussed AneurIST project in Section 6.2, but the computational feature constitutes also an important common ground, which highly relates to the work performed in this thesis. The GEMSS project as outlined in Section 6.1 and many other projects such as the Swiss BioOpera[19] [*Bausch et al.*, 2002] or the Japanese BioGrid[20] [*Akiyama et al.*, 2005] emphasize compute intensive applications and their Grid-enabling.

Quality of service

Related work in the field of Quality of Service has been investigated basically in Section 2.3 covering individual QoS aspects on the network-, application- and service-level. Contrarily to the presented rather traditional approaches and solutions specifically focusing on the detail at hand this section emphasizes similar comprehensive work as envisaged in this thesis. Quality of Service is closely related to business approaches due to the market potential of a certain level of a service in contrast to the common best effort situation in the Internet. As a consequence, projects highlighted in the business domain usually have QoS implications as well.

From the medical domain it can be seen that a number of projects focus rather

[17] caBIG, `http://cabig.cancer.gov/`

[18] US BIRN, `http://www.nbirn.net/`

[19] Swiss BioOpera, `http://www.iks.inf.ethz.ch/projects/projects/bioopera/`

[20] Japanese BioGrid, `www.biogrid.jp/`

on data aspects and the presented QoS approach has not yet been investigated with respect to data access and integration in the Grid. A pointer in this direction is the work presented in [*Braumandl et al.*, 2003], which focuses on QoS support for distributed query processing, suggesting QoS management at every stage of a query execution including planning, optimization and execution, in order to achieve significantly better results than in current systems based on a best-effort policy.

An introducing work of QoS in Grid computing has been presented in [*Menasce and Casalicchio*, 2004] dealing in particular with a QoS-based Web services architecture. The work proposes Web services build upon QoS-aware components, which support invocation via an according QoS negotiation protocol. The Web services concerned are rather plain Web services with no underlying native applications or data sources which e.g. require HPC hardware to be executed on, compared to the complex application-based Web services dealt with in this work. However, the basic requirement of advance reservation in this context, as also proposed in this work, is also discussed in [*Sulistio and Buyya*, 2004] and [*Menasce*, 2004]. The latter presented the effects of resource allocation on service level agreements. Again, the underlying applications are fundamentally different being typical enterprise applications as opposed to this work.

Similar to [*Menasce and Casalicchio*, 2004] the work proposed in [*Al-Ali et al.*, 2003] follows a QoS-aware component architecture for Grid computing, but again it rather emphasizes plain Web services. With respect to QoS-aware service discovery the UDDIe system [*ShaikhAli et al.*, 2003] introduced in Section 2.1 should be mentioned due to its support for QoS attributes. Similarly [*Ran*, 2003] proposes specific QoS metrics to be integrated in UDDI. Basically both systems support arbitrary QoS attributes, but do not specifically deal with response time and price guarantees, which have to be provided by the registered Grid services and their underlying applications.

In summary, the field of Quality of Service in Grid computing comprises a number of diverse and promising approaches, which this thesis also aims to contribute to.

Chapter 9

Conclusion

This chapter provides a brief wrap-up of this thesis comprising a short summary, conclusions and future directions. The summary presents the discussed work with respect to Grid computing and Quality of Service, while the concluding remarks discuss the work and its applicability. Finally, potential future directions are addressed mainly pointing towards the upcoming Cloud computing developments.

Summary and contribution

Grid Computing and Quality of Service brought up enormous challenges for research in recent years especially towards the creation of a new IT cyber-infrastructure. This PhD thesis aimed to contribute to this development by proposing a Grid infrastructure supporting Quality of Service. QoS-aware Grid computing envisaged in this work comprises on-demand provision of HPC applications as Grid services and negotiable QoS guarantees for multiple clients and furthermore follows standardized protocols and implementations therein.

This complex undertaking has been accomplished by presenting a comprehensive overview of relevant technology in Chapter 2, followed by the main contributions of this work, each in a separate chapter:

- Design and development of a service-oriented Grid environment (Chapter 3)
- Quality of Service models for capabilities, requests and negotiation (Chapter 4)
- Secure Quality of Service support infrastructure (Chapter 5)

Each topic comprises an according design and implementation, which is finally evaluated. This evaluation of the developments has been performed practically by applying the middleware in different research projects as presented in Chapter 6 and experimentally by conducting various experiments as outlined in Chapter 7.

Conclusion

The design and implementation of the Grid environment has proved to be forward-looking by adopting a service-oriented approach and realizing a pure Web services-based solution. This has been especially important due to the fact that the Grid software market was dominated by the initial versions of the Globus toolkit at the time of the development kick-off for the Grid environment. In the meantime the trend towards service orientation has been globally accepted.

The presented Grid middleware and in particular the QoS support has been successfully applied and demonstrated in EU projects in the medical and bio-medical domains. Using Grid technology in these contexts enabled medical practitioners and researchers to utilize Grid services in their clinical environment and eventually demonstrate new prospects for improving healthcare.

Furthermore, in the concourse of this work an experimental evaluation of the overall system has been performed. Various tests have been conducted ranging from a low level stress tests to high level negotiation tests providing insights to the system and the pleasant evidence for the system's rational behavior and high robustness when penetrated with a huge number of client requests.

Future directions

Even if many commercial Grid developments exist, Cloud computing is increasingly promoted by the industry. Companies like Amazon, Google or IBM investigated ways to make business with their enormous computing resources, which are only utilized upon peak demand and the results are the nowadays broadly used Amazon Elastic Compute Cloud (EC2), Google App Engine or IBM Enterprise Data Center. The concept of virtualization is also envisaged in the Grid. Clouds are typically accessible through (Web) services, i.e. the computing power is actually virtualized. Furthermore, Cloud computing adopts concepts of the service science, which targets to make software, platforms or even hardware available as a service to elevate virtualization to even higher layers.

However, Cloud computing is not intended to entirely replace Grid computing, it rather enriches the Grid with further capabilities and increases scalability and computing power, if Grids incorporate Clouds or Clouds incorporate Grids. Moreover, economic aspects are considered with the pay-as-you-go concept and as soon as the industry promotes a certain concept, a promising future for the next generations of cyberinfrastructures can be expected.

Bibliography

Akiyama, T., et al., Scientific grid activities in Cybermedia Center, Osaka University, *Cluster Computing and the Grid, IEEE International Symposium on*, *1*, 463–470, 2005.

Al-Ali, R., A. Shaikhali, O. Rana, and D. Walker, Supporting QoS-based discovery in service-oriented Grids, in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, 2003.

Al-Masri, E., and Q. H. Mahmoud, Investigating web services on the world wide web, in *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pp. 795–804, ACM, New York, NY, USA, 2008.

Allcock, B., The Globus Toolkit: Status and Plans, CrossGrid 2004, `http://grid.ucy.ac.cy/axgrids04/AxGrids/presentations/CrossGrids04.ppt`, 2004.

Alonso, G., F. Casati, H. Kuno, and V. Machiraju, *Web Services Concepts, Architectures and Applications*, Springer, 2004.

Andrieux, A., et al., Web Service Agreement Specification (WS-Agreement), `http://www.ogf.org/documents/GFD.107.pdf`, 2007.

Arbona, A., S. Benkner, G. Engelbrecht, J. Fingberg, M. Hofmann, K. Kumpf, G. Lonsdale, and A. Wöhrer, A Service-oriented Grid Infrastructure for Biomedical Data and Compute Services, in *Proceedings of the International Workshop on Network Tools and Applications in Biology*, Santa Margherita di Pula, Italy, 2006.

Arbona, A., S. Benkner, G. Engelbrecht, J. Fingberg, M. Hofmann, K. Kumpf, G. Lonsdale, and A. Wöhrer, A Service-oriented Grid Infrastructure for Biomedical Data and Compute Services, *IEEE Transactions on NanoBioscience*, *2*(6), 2007.

Ausubel, L., *Auction Theory for the New Economy - New Economy Handbook*, Academic Press, 2003.

Backfrieder, W., S. Benkner, and G. Engelbrecht, Web-Based Parallel ML-EM Reconstruction for SPECT on SMP Clusters, in *Proceedings of the International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences*, CSREA Press, Las Vegas, USA, 2001.

Backfrieder, W., M. Forster, S. Benkner, G. Engelbrecht, N. Terziev, and A. Dimitrov, Accurate Attenuation Correction for A Fully 3D Reconstruction Service, in *Proceedings of the International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences*, CSREA Press, Las Vegas, USA, 2002.

Backfrieder, W., M. Forster, S. Benkner, and G. Engelbrecht, Locally Variant VOR in Fully 3D SPECT within A Service Oriented Environment, in *Proceedings of the International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences*, CSREA Press, Las Vegas, USA, 2003a.

Backfrieder, W., M. Forster, G. Engelbrecht, and S. Benkner, Optimized design of VOR for 3D image reconstruction in SPECT in a service oriented parallel implementation, *Nuclear Medicine Technology*, *31*(2), 2003b.

Bala, P., et al., BioGRID - An European Grid for Molecular Biology, in *Proceedings of 11th International Symposium on High Performance Distributed Computing (HPDC)*, pp. 412–417, IEEE Computer Society, Edinburgh, Scotland, 2002.

Balenson, D., B. Kaliski, S. Kent, and J. Linn, Privacy Enhancement for Internet Electronic Mail: Part I to IV (RFC 1421, 1422, 1423, 1424), `http://tools.ietf.org/html/rfc1421|1422|1423|1424`, 1993.

Banks, T., Web Services Resource Framework (WSRF) Primer v1.2, `http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-02.pdf`, 2006.

Bausch, W., C. Pautasso, R. Schaeppi, and G. Alonso, BioOpera: Cluster-Aware Computing, in *CLUSTER '02: Proceedings of the IEEE International Conference on Cluster Computing*, p. 99, IEEE Computer Society, Washington, DC, USA, 2002.

Benkner, S., and G. Engelbrecht, Generic QoS Support for Application Web Services, in *International Symposium on Web Services and Applications*, Las Vegas, USA, 2005.

Benkner, S., and G. Engelbrecht, A Generic QoS Infrastructure for Grid Web Services, in *Proceedings of the International Conference on Internet and Web Applications and Services*, IEEE Computer Society Press, Guadeloupe, French Caribbean, 2006.

Benkner, S., A. Dimitrov, G. Engelbrecht, R. Schmidt, and N. Terziev, Medical Image Reconstruction in a Grid Environment, in *Proceedings, 2nd Cracow Grid Workshop*, Cracow, Poland, 2002.

Benkner, S., I. Brandic, A. Dimitrov, G. Engelbrecht, R. Schmidt, and N. Terziev, Performance of Java Web Services Implementations, in *Proceedings International Conference on Web Services*, CSREA Press, Las Vegas, USA, 2003a.

Benkner, S., A. Dimitrov, G. Engelbrecht, R. Schmidt, and N. Terziev, A Service-Oriented Framework for Parallel Medical Image Reconstruction, in *Proceedings International Conference on Computational Science, pp.612-621*, Springer Verlag, Melbourne, Australia, 2003b.

Benkner, S., G. Berti, G. Engelbrecht, J. Fingberg, G. Kohring, S. E. Middleton, and R. Schmidt, GEMSS: Grid-infrastructure for Medical Service Provision, in *In Proceedings of HealthGRID 2004*, Clermont-Ferrand, France, 2004a.

Benkner, S., I. Brandic, G. Engelbrecht, and R. Schmidt, VGE - A Service-Oriented Grid Environment for On-Demand Supercomputings, in *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (Grid 2004)*, IEEE, Pittsburgh, PA, USA, 2004b.

Benkner, S., G. Berti, G. Engelbrecht, J. Fingberg, G. Kohring, S. E. Middleton, and R. Schmidt, GEMSS: Grid Infrastructure for Medical Service Provision, *Methods of Information in Medicine*, *44*, 2005a.

Benkner, S., I. Brandic, G. Engelbrecht, S. E. Middleton, and R. Schmidt, Application-Level QoS Support for a Medical Grid Infrastructure, in *Life Sciences Grid Workshop, Grid Asia*, Singapore, 2005b.

Benkner, S., I. Brandic, G. Engelbrecht, and R. Schmidt, VGE - A QoS-Enabled Grid Computing Environment, in *Proceedings 1st Austrian Grid Symposium*, OCG Verlag, Schloss Hagenberg, Austria, 2005c.

Benkner, S., K. F. Doerner, R. Hartl, G. Kiechle, and M. Lucka, Commutication Strategies for Parallel Cooperative Ant Colony Optimization on Clusters and Grids, in *Proceedings PARA'04 Workshop on State-of-the-art in Scientific Computing*, edited by J. Dongarra, K. Madsen, and J. Wasniewski, pp. 3–12, Technical University of Denmark, Lyngby, 2005d.

Benkner, S., G. Engelbrecht, S. E. Middleton, and M. Surridge, Supporting SLA Negotiation for Grid-based Medical Simulation Services, in *Workshop on State-of-the-Art in Scientific and Parallel Computing*, Umea, Sweden, 2006.

Benkner, S., G. Engelbrecht, S. E. Middleton, I. Brandic, and R. Schmidt, End-to-End QoS Support for a Medical Grid Service Infrastructure, *New Generation Computing, Computing Paradigms and Computational Intelligence, Special Issue on Life Science Grid Computing, Ohmsha, Ltd. and Springer*, *25*(4), 2007.

Benkner, S., G. Engelbrecht, M. Köhler, and A. Wöhrer, Virtualizing Scientific Applications and Data Sources as Grid Services, in *Cyberinfrastructure Technologies and Applications*, edited by J. Cao, Nova Science Publishers, New York, USA, 2008.

Benkner, S., et al., Numerical Simulation for eHealth: Grid-enabled Medical Simulation Services, PARCO2003, Parallel Computing 2003, Dresden, Germany, in

Parallel Computing: Software Technology, Algorithms, Architectures and Applications, edited by G. Joubert, W. Nagel, F. Peters, and W. Walter, Advances in Parallel Computing Elsevier, The Netherlands, 2004c.

Berti, G., S. Benkner, J. W. Fenner, J. Fingberg, G. Lonsdale, S. E. Middleton, and M. Surridge, Medical Simulation Services via the Grid, in *Proceedings of 1st European HealthGRID Conference*, pp. 248–259, Lyon, France, 2003.

Blake, S., D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, An Architecture for Differentiated Services (RFC 2475), `http://tools.ietf.org/html/rfc2475`, 1998.

Braden, R., Requirements for Internet Hosts – Communication Layers (RFC 1122), `http://tools.ietf.org/html/rfc1122`, 1989.

Braden, R., L. Zhang, S. Berson, S. Herzog, and S. Jamin, Resource ReSerVation Protocol (RSVP) (RFC 2205), `http://tools.ietf.org/html/rfc2205`, 1997.

Brainard, J., A. Juels, R. Rivest, M. Szydlo, and M. Yung, Fourth Factor Authentication: Somebody You Know, in *In ACM CCS*, pp. 168 – 178, 2006.

Brandic, I., S. Benkner, G. Engelbrecht, and R. Schmidt, Towards Quality of Service Support for Grid Workflows, in *Proceedings of the European Grid Conference 2005 (EGC2005)*, Springer Verlag, Amsterdam, The Netherlands, 2005a.

Brandic, I., S. Benkner, G. Engelbrecht, and R. Schmidt, QoS Support for Time-Critical Grid Workflow Applications, in *Proceedings 1st IEEE International Conference on eScience and Grid Computing*, Melbourne, Australia, 2005b.

Brandic, I., S. Pllana, and S. Benkner, Specification, planning, and execution of QoS-aware Grid workflows within the Amadeus environment, *Concurr. Comput. : Pract. Exper.*, *20*(4), 331–345, 2008.

Braumandl, R., A. Kemper, and D. Kossmann, Quality of Serivce in an Information Economy, *ACM Transactions on Internet Technology*, *3*(4), 291–333, 2003.

Braverman, A. M., Father of the Grid, `http://magazine.uchicago.edu/0404/features/index.shtml`, 2007.

Buetow, K., Cyberinfrastructure: Empowering a "Third Way" in Biomedical Research, *Science*, *308*(5723), 821–824, 2005.

Buyya, R., Economic-based Distributed Resource Management and Scheduling for Grid Computing, Ph.D. thesis, Monash University, Melbourne, Australia, 2002.

Cao, J., and F. Zimmermann, Queue Scheduling and Advance Reservations with COSY, in *Proceedings of the International Parallel and Distributed Processing Symposium*, Santa Fe, New Mexico, USA, 2004.

Cha, S.-J., Y.-Y. Hwang, Y.-S. Chang, K.-O. Kim, and K.-C. Lee, The Performance Evaluations and Enhancements of GIS Web Services, in *Proceedings of the 2007 International Conference on Multimedia and Ubiquitous Engineering*, pp. 668–673, IEEE Computer Society, Washington, DC, USA, 2007.

Chinnici, R., M. Gudgin, J.-J. Moreau, and S. Weerawarana, Web Services Description Language (WSDL) Version 2.0, `http://www.w3.org/TR/wsdl20/`, 2007.

Christensen, E., F. Curbera, G. Meredith, and S. Weerawarana, Web Services Description Language (WSDL) Version 1.1, `http://www.w3.org/TR/wsdl`, 2001.

Clement, L., A. Hately, C. von Riegen, and T. Rogers, UDDI Version 3.0.2, `http://uddi.org/pubs/uddi_v3.htm`, 2004.

Cohen, F., Discover SOAP encoding's impact on Web service performance, `http://www.ibm.com/developerworks/webservices/library/ws-soapenc/`, 2003.

Cooper, D., S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile (RFC 5280), `http://tools.ietf.org/html/rfc5280`, 2008.

de Laat, C., G. Gross, L. Gommans, J. Vollbrecht, and D. Spence, Generic AAA Architecture (RFC 2903), `http://tools.ietf.org/html/rfc2903`, 2000.

D.Ferraiolo, and R. Kuhn, Role-based access controls, in *In Proceedings of 15th NIST-NCSC National Computer Security Conference*, pp. 554–563, 1992.

Diffie, W., and M. Hellman, Multi-user cryptographic techniques, *AFIPS Proceedings*, *45*, 109–112, 1976.

Dimitrakos, T., D. M. Randal, F. Yuan, M. Gaeta, G. Laria, P. Ritrovato, B. Serhan, S. Wesner, and K. Wulf, An Emerging Architecture Enabling Grid Based Application Service Provision, in *EDOC '03: Proceedings of the 7th International Conference on Enterprise Distributed Object Computing*, p. 240, IEEE Computer Society, Washington, DC, USA, 2003.

Du, Z., J. Huai, and Y. Liu, Ad-UDDI: An active and distributed service registry, in *VLDB International Workshop on Technologies for EServices, volume 3811 of LNCS*, pp. 58–71, Springer, 2006.

Dubuisson, O., *ASN.1 Reference Book: ASN.1 - Communication between heterogeneous systems*, Elsevier-Morgan Kaufmann, 2008.

Dunlop, R., et al., @neurIST - Chronic Disease Management through Integration of Heterogeneous Data and Computer-interpretable Guideline Services, in *Proceedings of Healthgrid 2008*, Chicago, USA, 2008.

E.800, Recommendation E.800: Terms and definitions related to quality of service and network performance including dependability, `http://www.itu.int/rec/T-REC-E.800-199408-S/en`, 1994.

Eronen, P., and H. Tschofenig, Pre-Shared Key Ciphersuites for Transport Layer Security (RFC 4279), `http://tools.ietf.org/html/rfc4279`, 2005.

Farrel, A., A. Ayyangar, and J. Vasseur, Inter-Domain MPLS and GMPLS Traffic Engineering – Resource Reservation Protocol-Traffic Engineering (RSVP-TE) Extensions (RFC 5151), `http://tools.ietf.org/html/rfc5151`, 2008.

Fielding, R. T., REST: Architectural Styles and the Design of Network-based Software Architectures, Doctoral dissertation, University of California, Irvine, 2000.

Foster, I., What is the Grid? A Three Point Checklist, `http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf`, 2002.

Foster, I., and C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

Foster, I., C. Kesselman, and S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations, *International Journal of Supercomputer Applications*, *15*, 2001, 2001.

Foster, I., C. Kesselman, J. Nick, and S. Tuecke, The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, , Global Grid Forum, 2002.

Foster, I., et al., The Open Grid Services Architecture, Version 1.5, `http://www.ogf.org/documents/GFD.80.pdf`, 2006.

Friedrich, C. M., H. Dach, T. Gattermayer, G. Engelbrecht, S. Benkner, and M. Hofmann-Apitius, @neuLink: A Service-oriented Application for Biomedical Knowledge Discovery, in *Proceedings of Healthgrid 2008*, Chicago, USA, 2008.

Garofalakis, J., Y. Panagis, and E. Sakkopoulos, Web Service Discovery mechanisms: looking for a needle in a haystack, in *In: International Workshop on Web Engineering. (2004*, 2004.

Goldreich, O., *Foundations of Cryptography, Volume 1: Basic Tools*, Cambridge University Press, 2001.

Gubala, T., et al., ViroLab Virtual Laboratory, in *Kracow Grid Workshop 2007 Workshop Proceedings*, pp. 35–40, 2007.

Gudgin, M., N. Mendelsohn, M. Nottingham, and H. Ruellan, XML-binary Optimized Packaging, `http://www.w3.org/TR/xop10/`, 2005.

Gudgin, M., M. Hadley, and T. Rogers, Web Services Addressing 1.0 - Core, `http://www.w3.org/TR/ws-addr-core/`, 2006.

Gudgin, M., M. Hadley, N. Mendelsohn, J.-J. Moreau, F. F. Nielsen, A. Karmarkar, and Y. Lafon, SOAP version 1.2, `http://www.w3.org/TR/soap/`, 2007.

Haas, H., and A. Brown, Web Services Glossary, W3C Working Group Note 11 February 2004, `http://www.w3.org/TR/ws-gloss/`, 2004.

Heiko, L., A. Keller, A. Dan, R. P. King, and R. Franck, Web Service Level Agreement (WSLA) Language Specification, , IBM Corporation, 2003.

Helger, P., *phloc-logging 1.2 - A generic Java Logging System*, Sourceforge, `http://sourceforge.net/projects/phloc-logging`, 2008.

Herveg, J., and Y. Poullet, Directive 95/46 and the use of GRID technologies in the healthcare sector : selected legal issues, in *Proceedings of the 1st European HealthGRID Conference*, pp. 229–236, Lyon, France, 2003.

Hwang, S.-Y., and B. Riddle, BRUW: Bandwidth Reservation for User Work, in *In Proceedings of the TERENA Networking Conference 2005*, Poznan, Poland, 2005.

Iacono, L. L., and H. Rajasekaran, Security Architecture for Distributed Medical Information Systems, in *GI Jahrestagung (1)*, pp. 110–116, 2008.

Jackson, D., Q. Snell, and M. Clement, Core Algorithms of the Maui Scheduler, in *Proceedings of the 7th Job Scheduling Stgrategies for Parallel Processing, ACM SIGMETRICS*, Cambridge, Massachusetts, USA, 2001.

Jacq, N., et al., Virtual screening on large scale grids, *Parallel Computing*, *33*(4–5), 289–301, 2007.

Joel, A. E. (Ed.), *Asynchronous Transfer Mode Switching*, IEEE, 1993.

Jones, D., J. Fenner, G. Berti, F. Kruggel, R. Mehrem, W. Backfrieder, B. Moore, and A. Geltmeier, The GEMSS Grid: An Evolving HPC Environment for Medical Applications, in *Proceedings of HealthGrid 2004*, Clermont-Ferrand, France, 2004.

Josefsson, S., The Base16, Base32, and Base64 Data Encodings (RFC 4648), `http://tools.ietf.org/html/rfc4648`, 2006.

Jovicich, J., M. F. Beg, S. Pieper, C. E. Priebe, M. I. Miller, R. L. Buckner, and B. Rosen, Biomedical Informatics Research Network: Integrating Multi-Site Neuroimaging Data Acquisition, Data Sharing and Brain Morphometric Processing., in *Proceedings of Compuer-based Medical Systems (CBMS) 2005*, pp. 288–293, IEEE Computer Society, 2005.

Kahn, D., *The Codebreakers - The Story of Secret Writing*, Macmillan Pub Co, 1967.

Kaliski, B., PKCS #7: Cryptographic Message Syntax Version 1.5, `http://tools.ietf.org/html/rfc2315`, 1998.

Kesselman, C., and I. Foster, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

Koch, O., W. Kreuzer, and A. Scrinzi, MCTDHF in Ultrafast Laser Dynamics, AURORA TR-2003-29, Inst. for Appl. Math. and Numer. Anal., Vienna Univ. of Technology, Austria, http://www.vcpc.univie.ac.at/aurora/publications/, 2003.

Kumpf, K., A. Wöhrer, S. Benkner, G. Engelbrecht, and J. Fingberg, A Semantic Mediation Architecture for a Clinical Data Grid, in *Grid Computing for Bioinformatics and Computational Biology*, edited by E.-G. Talbi and A. Zomaya, Wiley Series in Bioinformatics, pp. 267–298, John Wiley & Sons, 2007.

Laure, E. (Ed.), *The EU DataGrid Setting the Basis for Production Grids*, *Journal of Grid Computing*, vol. 2(4), Springer, 2004.

Laure, E., et al., Programming the Grid with gLite, *Methods in Science and Technology*, *12*(1), 33–45, 2006.

Leach, P., M. Mealling, and R. Salz, A Universally Unique IDentifier (UUID) URN Namespace (RFC 4122), http://tools.ietf.org/html/rfc4122, 2005.

Li, J., and A. Karp, Access control for the services oriented architecture, in *SWS '07: Proceedings of the 2007 ACM workshop on Secure web services*, pp. 9–17, ACM, New York, NY, USA, 2007.

Linn, J., Trusts Model and Management in Public Key Infrastructures, , RSA Laboratories, Bedford, MA, USA, 2000.

Loscocco, P., and S. Smalley, Meeting Critical Security Objectives with Security-Enhanced Linux, in *Proceedings of the 2001 Ottawa Linux Symposium*, 2001.

Luhn, H. P., A business intelligence system, *j-IBM-JRD*, *2*, 314–319, 1958.

Lysyanskaya, A., Signature Schemes and Applications to Cryptographic Protocol Design, Ph.D. thesis, MIT, 2002.

MacLaren, J., Advance Reservations: State of the Art, , Open Grid Forum, 2003.

McGough, S., L. Young, A. Afzal, S. Newhouse, and J. Darlington, Workflow enactment in ICENI, http://www.lesc.ic.ac.uk/events/ahm2004.jsp, 2004.

Meier, J., C. Farre, J. Taylor, P. Bansode, S. Gregersen, M. Sundararajan, and R. Boucher, *patterns & practices Improving Web Services Security Guide*, Microsoft, http://www.codeplex.com/WCFSecurityGuide, 2008.

Menasce, D., QoS-Aware Software Components, *In Internet Computing Online, Vol. 8, No. 2, pp.91-93*, 2004.

Menasce, D., and E. Casalicchio, QoS in Grid Computing, *IEEE Internet Computing*, *8*(4), 85–87, 2004.

Merdy, E. L., A representation of concepts defined by a WSDL 1.1 document, http://en.wikipedia.org/wiki/Web_Services_Description_Language, 2008.

Middleton, S., J. Herveg, F. Crazzolara, D. Marvin, and Y. Pullet, GEMSS: Privacy and Security for a Medical Grid, *Methods of Information in Medicine (2)*, 2005.

Middleton, S. E., M. Surridge, S. Benkner, and G. Engelbrecht, Quality of service negotiation for commercial medical Grid services, *Journal of Grid Computing, Springer Verlag, ISSN 1570-7873*, *5*(4), 429–447, 2007.

Milanesi, L., White paper: guidelines and recommendations for the scientific community based on the experience and the results gained from the BioinfoGRID project, `http://www.bioinfogrid.eu/Documentation/bioinfogrid_white_paper`, 2007.

Nelson, D., and A. DeKok, Common Remote Authentication Dial In User Service (RADIUS) Implementation Issues and Suggested Fixes (RFC 5080), `http://tools.ietf.org/html/rfc5080`, 2007.

Nissanoff, D., *FutureShop: How the New Auction Culture Will Revolutionize the Way We Buy, Sell and Get the Things We Really Want*, The Penguin Press, 2006.

Nudd, G. R., D. J. Kerbyson, E. Papaefstathiou, S. C. Perry, J. S. Harper, and D. V. Wilcox, Pace–A Toolset for the Performance Prediction of Parallel and Distributed Systems, *International Journal of High Performance Computing Applications*, *14*(3), 228–251, 2000.

Orange Book 1985, *Department of Defense: Trusted Computer System Evaluation Criteria*, US Department of Defense, 1985.

Parkhill, D. F., *The challenge of the computer utility*, Reading, Mass., Addison-Wesley Pub. Co., 1966.

Patil, A., B. Belter, A. Polyrakis, M. Przybylski, T. Rodwell, and M. Grammatikou, GEANT2 Advance Multi-domain Provisioning System, in *In Proceedings of the TERENA Networking Conference 2006*, Catania, Italy, 2006.

Pattnik, P., K. Ekanadham, and J. Jann, Autonomic computing and Grid, in *Grid Computing Making Global Infrastructure a reality*, edited by F. Berman, G. Fox, and A. J. G. Hey, pp. 351–362, John Wiley & Sons, 2003.

Pettifer, S., et al., myGrid and UTOPIA: An Integrated Approach to Enacting and Visualising in Silico Experiments in the Life Sciences, in *Data Integration in the Life Sciences*, pp. 59–70, Springer, 2007.

Pllana, S., and T. Fahringer, Performance Prophet: A Performance Modeling and Prediction Tool for Parallel and Distributed Programs, in *ICPP Workshops*, pp. 509–516, 2005.

Raja, M. N., H. F. Ahmad, H. Suguri, P. Bloodsworth, and N. Khalid, SOA compliant FIPA agent communication language, in *Proceeedings of the first International Conference of Applications of Digital Information and Web Technologies (ICADIWT) 2008*, pp. 470–477, 2008.

Ran, S., A model for web services discovery with QoS, *SIGecom Exch.*, *4*(1), 1–10, 2003.

Rivest, R., A.Shamir, and L. Adleman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, *Communications of the ACM*, *21*(2), 120–126, 1978.

Romberg, M., E. Benfenati, and W. Dubitzky, Open Computing Grid for Molecular Sciences, in *Grid Computing for Bioinformatics and Computational Biology*, edited by E.-G. Talbi and A. Zomaya, Wiley Series in Bioinformatics, pp. 1–22, John Wiley & Sons, 2007.

Ruckenbauer, M., I. Brandic, S. Benkner, W. Gansterer, O. Gervasi, M. Barbatti, and H. Lischka, Nonadiabatic Ab Initio Surface-Hopping Dynamics Calculation in a Grid Environment - First Experiences, in *Proceeedings of the 2007 International Conference on Computational Science and Its Applications (ICCSA 2007)*, *LNCS*, vol. 4705, pp. 281–294, Springer Verlag, Kuala Lumpur, Malaysia, 2007.

Saltzer, J., and M. Schroeder, The Protection of Information in Computer Systems, *Proceedings of the IEEE*, *63*(9), 1278–1308, 1975.

SAML, Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, `http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf`, 2005.

Sawaragi, Y., H. Nakayama, and T. Tanino, *Theory of multiobjective optimization*, Academic Press, Orlando, Floria, USA, 1985.

Schmidt, R., S. Benkner, I. Brandic, and G. Engelbrecht, Applying a Component Model to Grid Application Services, in *Tenth International Workshop on Component-Oriented Programming (WCOP 2005)*, Glasgow, Scotland, 2005a.

Schmidt, R., S. Benkner, I. Brandic, and G. Engelbrecht, Component based Applications Programming within a Service-Oriented Grid Environment, in *Workshop on Component Models and Frameworks in High Performance Computing (Compframe 2005)*, Atlanta, USA, 2005b.

Schmidt, R., S. Benkner, I. Brandic, and G. Engelbrecht, Component-Oriented Application Construction for a Web Service Based Grid, *Concurrency and Computation: Practice and Experience*, *19*(5), 637–650, 2007.

SGE, *Beginner's Guide to Sun Grid Engine 6.2 - Installation and Configuration - White paper*, Sun Microsystems, Inc., 2008.

ShaikhAli, A., O. Rana, R. Al-Ali, and D. Walker, UDDIe: an extended registry for Web services, *Applications and the Internet Workshops, 2003. Proceedings. 2003 Symposium on*, pp. 85–89, 2003.

Simpson, W., CHAP - Challenge Handshake Authentication Protocol (RFC 1994), `http://tools.ietf.org/html/rfc1994`, 1996.

Skeen, D., and M. Stonebraker, A Formal Model of Crash Recovery in a Distributed System, *Software Engineering, IEEE Transactions on*, *SE-9*(3), 219–228, 1983.

Smith, W., I. T. Foster, and V. E. Taylor, Predicting Application Run Times Using Historical Information, in *IPPS/SPDP '98: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 122–142, Springer-Verlag, London, UK, 1998.

Smith, W., I. Foster, and V. Taylor, Scheduling with Advanced Reservations, in *In Proceedings of International Parallel and Distributed Processing Symposium (IPDPS00)*, pp. 127–132, Cancun, Mexico, 2000.

Snell, Q., M. J. Clement, D. B. Jackson, and C. Gregory, The Performance Impact of Advance Reservation Meta-scheduling, in *IPDPS '00/JSSPP '00: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 137–153, Springer-Verlag, London, UK, 2000.

Sotomayor, B., The Globus Toolkit 3 Programmer's Tutorial - Key concepts: OGSA, OGSI, and GT3, `http://gdp.globus.org/gt3-tutorial/multiplehtml/ch01s01.html`, 2004.

Stockinger, H., Defining the Grid: a snapshot on the current view, *Journal of Supercomputing*, *42*(1), 3–17, 2007.

Sulistio, A., and R. Buyya, A Grid Simulation Infrastructure Supporting Advance Reservation, in *Proceedings of the 16th International Conference on Parallel and Distributed Computing and Systems 2004*, ACTA Press, MIT Cambridge, Boston, USA., 2004.

Surridge, M., S. Taylor, and D. Marvin, Grid Resources for Industrial Applications, in *Proceedings of 2004 IEEE International Conference on Web Services*, pp. 402–409, San Diego, USA, 2004.

Surridge, M., S. Taylor, D. D. Roure, and E. Zaluska, Experiences with GRIA - Industrial applications on a web services Grid, in *Proceedings of 1st IEEE Conference on e-Science and Grid Computing*, Melbourne, Australia, 2005.

Taylor, D., T. Wu, N. Mavrogiannopoulos, and T. Perrin, Using the Secure Remote Password (SRP) Protocol for TLS Authentication (RFC 5054), `http://tools.ietf.org/html/rfc5054`, 2007.

TLS, Transport Layer Security (TLS) Protocol Version 1.2 (RFC 5246), `http://tools.ietf.org/html/rfc5246`, 2008.

Tuecke, S., et al., The Open Grid Services Infrastructure, Version 1.0, `http://www.ggf.org/documents/GFD.15.pdf`, 2003.

Vickrey, W., Counter speculation, Auctions and Competitive Sealed Tenders, *Journal of Finance*, *1*(16), 8–37, 1961.

Warren, R., et al., MammoGrid – a prototype distributed mammographic database for Europe, *Clinical Radiology*, *62*(11), 1044–1051, 2007.

Weerawarana, S., WS-* vs. REST: Mashing up the Truth from Facts, Myths and Lies, http://wso2.org/files/myths-facts-lies-apacheconus07.pdf, 2007.

Wiki-PKI, PKI - Public Key Infrastructure, http://en.wikipedia.org/wiki/Public_key_infrastructure, 2008.

Wöhrer, A., P. Brezany, and A. M. Tjoa, Novel mediator architectures for Grid information systems, *FGCS*, *21*(1), 107–114, 2005.

Woodside, M., and D. Menasce, Application-Level QoS, *IEEE Internet Computing*, *10*(3), 13–15, 2006.

WS-Policy, Web Services Policy Framework (WS-Policy) and Web Services Policy Attachment (WS-PolicyAttachment), http://schemas.xmlsoap.org/ws/2004/09/policy/, 2004.

WS-SecureConversation, Web Services Secure Conversation Language (WS-SecureConversation), http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-secon/ws-secureconversation.pdf, 2005.

WS-Security, Web Services Security, SOAP Message Security 1.1 (WS-Security 2004), http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf, 2006.

WS-Trust, Web Services Trust 1.3, http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.pdf, 2005.

WSMgmt, Web Services for Management (WS-Management) Specification (DSP0226), http://www.dmtf.org/standards/wsman/, 2008.

X.200, Recommendation X.641: Information technology - Open Systems Interconnection - Basic Reference Model: The basic model, http://www.itu.int/rec/T-REC-X.200-199407-I/en, 1994.

X.641, Recommendation X.641: Information technology - Quality of service: framework, http://www.itu.int/rec/T-REC-X.641-199712-I/en/, 1998.

XACML, eXtensible Access Control Markup Language (XACML) Version 2.0, http://xml.coverpages.org/XACMLv20CD-CoreSpec.pdf, 2004.

Xenitellis, S., *The Open-source PKI Book: A guide to PKIs and Open-source Implementations - version 2.4.6*, Sourcefourge, http://ospkibook.sourceforge.net/, 2000.

Xiao, X., *Technical, Commercial and Regulatory Challenges of QoS: An Internet Service Model Perspective*, Morgan Kaufmann, 2008.

XML-Encryption, XML Encryption Syntax and Processing - W3C Recommendation, `http://www.w3.org/TR/xmlenc-core/`, 2002.

XML-Signatures, XML Signature Syntax and Processing (Second Edition) - W3C Recommendation, `http://www.w3.org/TR/xmldsig-core/`, 2008.

Zheng, W., *Internet QoS: Architectures and Mechanisms for Quality of Service*, Morgan Kaufmann, 2001.

Zimmermann, P., *The Official PGP User's Guide*, MIT Press, 1995.

CURRICULUM VITAE

Name: Gerhard Engelbrecht

Title: Mag.

Date and Place of Birth: 16 October 1976, Salzburg, Austria

Nationality: Austria

Affiliation:
Institute of Scientific Computing
University of Vienna
Nordbergstr. 15/C/3, A-1090 Vienna, Austria
Phone: +43-1-4277-39410
Fax: +43-1-4277-9394
E-mail: gerry@par.univie.ac.at

Education:

1996	High School Diploma (*Matura*)
2002	Industrial Computer Science at the Technical University of Vienna

Selected Project Experience:

2002 – 2007	Aurora Project 2, Prof. Zima
2002 – 2005	FP5 EU GEMSS Project
2006 – 2009	FP6 EU Aneurist Project

Publications:

More than 30 publications in refereed journals and conference proceedings.