Bachelor-Arbeit

Design and Implementation of a Security Gateway for Grid Services

Roland Gude
Fachhochschule Bonn-Rhein-Sieg

DLR

Fachhochschule Bonn-Rhein-Sieg

Fachbereich Informatik
Departemet Of Computer Sciences

Bachelor Thesis

Design and Implementation of a Security Gateway for Grid Services

von/by
Roland Gude

Erstprüfer:	**Prof. Dr. Rudolf Berrendorf**
Zweitprüfer:	**Prof. Dr. Ralf Thiele**

Eingereicht am: July 5, 2005

Declaration

I hereby declare, that the work presented in this thesis is solely my work and that to the best of my knowledge this work is original, except where indicated by references to other authors. No part of this work has been submitted for any other degree or diploma.

Abstract

Grid services will form the base for future computational Grids. Web Services, have been extended to build Grid services. Grid Services are defined in the Open Grid Service Architecture (OGSA) [22]. The Globus Alliance has released a Web Service Resource Framework, which is still under development and which is still missing vital parts. One of them is a Concept that allows Grid-Service Requests to securely traverse Firewalls, and its realization.

This Thesis aims at the development and realization of a detailed Concept for an Application Level Gateway for Grid services, based on an existing rough concept. This approach should enable a strict division between a local network and the Internet. The internet is considered as a untrusted site and the local network is considered as a trusted site. Grid resources are placed in the internet as well as in the local network. This means that the possibility to communicate through a firewall is essential. Some further protocols like Grid Resource Allocation and Management (GRAM) and the Grid File Transfer Protocol (GridFTP) must be able to traverse the network borders securely as well, while no further actions must be taken from the user side.

The German Federal Office for Information Security (BSI) proposes a *Firewall - Application Level Gateway (ALG) - Firewall* solution to the German Aerospace Center (DLR) where this Thesis is written, as a principle approach. In this approach, the local network is divided from the Internet with two firewalls. Between those firewalls is a demilitarized zone (DMZ), where computers may be placed, which can be accessed from the Internet and from the local network. An ALG which is placed in this DMZ should represent the local Grid nodes to the Internet and it should act as a client to the local nodes. All Grid service requests must be directed to the ALG instead of the protected Grid nodes. The ALG then checks and validates the requests on the application level (OSI layer 7). Requests that pose no security threat and fulfill certain criteria will then be forwarded to the local Grid nodes. The responses from the local Grid nodes are checked and validated by the ALG as well.

Contents

Chapter 1

Introduction

Grid-computing is seen as a technique that may enable multiple companies or research facilities to form joint ventures for certain computational tasks. It might also offer excellent ways to organize large amounts of data. The expectations for Grid computing are high. The ability to form Virtual Organizations between project partners is highly anticipated. For instance a car manufacturer could form such a Virtual Organization with one of its component suppliers in order to run simulations during the design process of a car with the structural data of the components the supplier produces. The car manufacturer has of course multiple component suppliers which are not partners, but competitors. The car manufacturer can now build multiple Virtual Organizations (one with each component supplier), but it must be assured that no component supplier has access to the structural data of its competitors. This example shows, that security is of a high priority for the practical use of Grid computing.

Today's Grid computing middleware already offers ways to form Virtual Organizations, but it does not yet deal with highly secured networks. All participants of a Virtual Organization will usually protect their data as much as they can. This includes restrictive firewalls which block most communication protocols. Today's Grid middleware can not deal with those restrictive firewalls and creation of Virtual Organizations is, due to this issue, not generally possible by now.

The DLR institution Simulations- and Softwaretechnik (SISTEC) is currently developing a software to manage and setup distributed simulations like those from the example above. A major feature of this software is, to run such simulations in a Grid. This feature is hardly useful, if the underlying Grid resources are not able to communicate with each other. That is why SISTEC decided to spend efforts on the research of this issue. This thesis is part of these research efforts.

The Global Grid Forum (GGF), a user, developer and vendor community which is trying to create global standards for Grid computing, has recently formed the Firewall-Issue research group (FI-RG), led by Leon Gommans from the Advanced Internet Research group at the Informatics Institute at University of Amsterdam. The FI-RG is doing research on problems between Grids and firewalls like those outlined in the scenario above. The concept described in this thesis as a possible solution for the problem has been presented by Thijs Metsch from the DLR on the GGF13. It has been perceived with interest and will probably play a large role in the work of the FI-RG. The author of this thesis and Thijs Metsch are members of the FI-RG.

This thesis describes an Application Level Gateway or security proxy as a solution for the scenario above. This concept has been suggested by the BSI as a general approach on firewall problems. The concept is compared to some other possible solutions and has been implemented as part of the thesis. The chapter 3 *Technical basis* will provide technical background information. It will explain what Grid services are and contains an introduction to firewall technology as well. The problem itself is further explained in chapter 2 *Problem Description* and the concept is described and compared in chapter 5 *Concept*. Chapter 4 *Similar Software products* is about two existing products which share a common ground with the software which has been developed as a part of this thesis. This software is explained in chapter 6 *Realization*.

Chapter 2

Problem Description

Some main goals of Grids are the abstraction from resource locations and the creation of virtual organizations (VO). Foster, Kesselmann and Tuecke define virtual organizations in the article *The Anatomy of the Grid*[8] as

> "[...] flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resources."

The resources of those VOs are usually distributed over many different locations and for security reasons, the different locations should be protected with firewalls (see figure 2.1). While those firewalls offer a vital protection for the resources, they interrupt communication between the Grid resources on the different locations. Even though Grid services are more ore less *firewall friendly*, because they are based on Web services, Grid service messages still can not traverse very restrictive firewalls. Due to the sensibility of the resources in industrial environments, firewalls which protect industrial networks will most likely be too restrictive for Grid services.

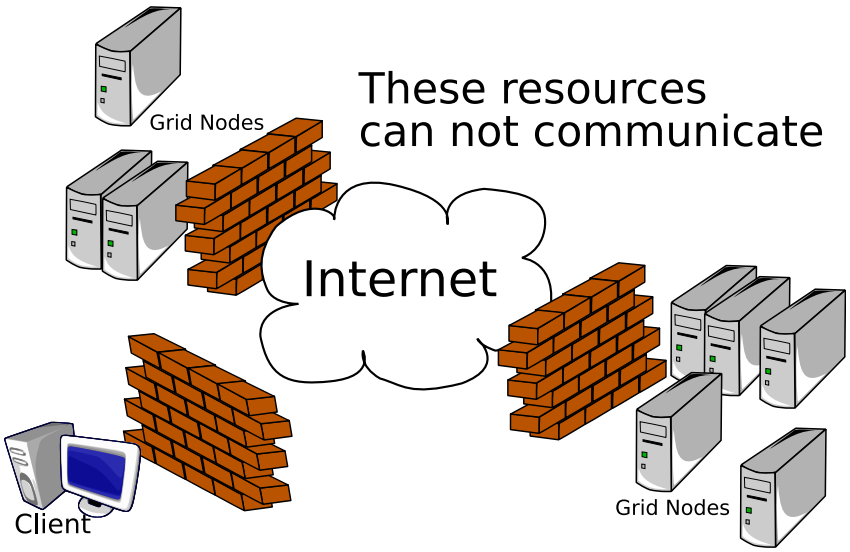

Figure 2.1: Virtual organization with firewall-protected local networks

There are several work-arounds for this problem, which can only be applied, while security for the resources is not of high priority. A common one, is to place all Grid resources in a DMZ which is

only protected by a lax firewall (Another far more restrictive firewall, divides the DMZ from the local sites network). Resources placed in this DMZ can be accessed from the internet and from the local network. The problem of this work-around is, that the Grid resources are not highly protected. Of course they can be accessed from the remote Grid resources, but any attacker has quite a chance to compromise them as well.

As said before, such work-arounds can not be used in Grids, whenever the Grid resources are somehow sensible. Which is always the case if industrial partners are involved and may even be the case when the Grid is just used for research. In such environments, Grid resources will never be placed inside the DMZ, but in the well protected local network. This leads to the conclusion, that a mechanism needs to be defined, which enables Grid service messages to traverse even restrictive firewalls while the security of the local site is not at risk.

The DLR institution SISTEC encountered this problem when they tried to aggregate their local Grid resources with those of the Fraunhofer-Institute for Algorithms and Scientific Computing (SCAI). Even though these Institutions cooperate very closely with each other, the firewalls posed obstacles which could not be overcome. Moving the Grid resources into the DMZ was not an option due to the licenses of some applications that where to run in the aggregated Grid, which only applied to machines in the local networks. Several concepts have been outlined to solve this problem. The Concept which seemed to be the only one that is generally applicable and secure enough, is the use of an Application Level Gateway, which is placed in the DMZ and transparently allows access to the Grid resources (see section 5.2 *Application Level Gateway*). Such an ALG would be accessible from both, the internet and the local network. It would be able to accept messages from the internet, check and validate them and forward them to their original receivers if they succeed validation. The answers would then be send to the ALG and forwarded to the original clients. If traffic from the DMZ into the local net is disallowed, the ALG must cache messages and another ALG, placed in the local network, must fetch the cached messages regularly. This concept is addressed in this Thesis and further elucidated in section 5.2 *Application Level Gateway*.

The example of the SISTEC/SCAI Grid shows, that the problem is not only a theoretical problem, but has quite some relevance for the practical usability of computational Grids. Even though both institutions work together very closely and often appear to project partners as a single Metainstitute, it was not possible to merge their Grid systems. A problem, almost every VO will probably encounter, when security policies have to be considered. Additionally to security policies, software licenses that are bound to certain machines can lead to the outlined problem.

Chapter 3

Technical basis

This Chapter describes some technical basics in order to understand the problem, and the solution which is proposed and developed in this Thesis, as well as the concept which lies behind that solution. The section 3.1 *Grid Services* explains what Grid services are. It therefor includes a short description of web services, which form the base of Grid services. The Simple Object Access Protocol is described, because it is the usual protocol Web services use. In section 3.2 *Firewalls* some basics about firewall technology and packet filtering are described.

3.1 Grid Services

During the evolution of Grid software, several approaches to the problems of grid computing have emerged. One of them is the concept of Grid services which is described in this section. Grid services are defined by the OGSA [9, 7, 5]. In short, OGSA Grid services are web services (see section 3.1.3 *Web Services*), which implement certain interfaces (also known as *portTypes*). In OGSA/Web Service Resource Framework (WSRF) based Grids, everything from computational resources over storage resources to networks and programs is represented as a service. Section 3.1.1 *The Open Grid Service Architecture* will give further insight into OGSA. According to [6], Grid service based Grids can be seen as the third generation of Grid systems. As stated in [6], these third-generation Grid systems are becoming more and more automated and autonomous. An attribute that is one of the goals of a service based Grid architecture.

3.1.1 The Open Grid Service Architecture

The Open Grid Service Architecture as proposed and outlined in [7] and defined in [9] is a service oriented Grid architecture, which addresses the need of Grid computing for standardization. It calls for capabilities which address certain key concerns of Grid computing, like authorization and membership of virtual organizations. It demands dynamic and heterogeneous environment support and interoperability in such environments. In addition it asks for the ability to integrate existing legacy systems into the Grid. The ability to virtualize resources, which means, that Grid services may be aggregated in order to create higher level Grid services, is also required. Common management capabilities are demanded in order to simplify the administration of such dynamic and heterogeneous environments as well as mechanisms for resource discovery and query. [9] states that a service based Grid architecture like OGSA must enable resource sharing between different organizations and still offer the possibility, to optimize resource allocation. Furthermore it must be possible to assure certain qualities of service if needed. And mechanisms for job submission, including the support for different job types as well as management of submitted jobs, scheduling and resource provisioning mus be contained. It must be capable of providing integrated data access and ensuring data consistence and persistence.

3.1.2 From OGSA to WSRF

In [7] Foster et al. explain and outline OGSA. They define a service, as a network-enabled entity which provides some capability. After a short discussion about how a Grid service could be accomplished, they identify Web services and the Web Service Definition Language (WSDL) as a good basis for Grid services. Some qualities of Web services and WSDL lead the OGSA Working Group (OGSA-WG) to this conclusion. These qualities are the ability to define service interfaces clearly and distinguished from the protocol bindings for service invocation. This service based approach allows distributed protocol bindings as well as locally optimized bindings for service invocation, which is a great advantage when it comes to Grid computing. The service implementation is also distinguished from interface definition and protocol binding. With basing OGSA on Web services and WSDL, OGSA adopts all these features.

However, this does not clarify, what the differences between Web services and Grid services are. While Web services and WSDL only provide means to define interfaces and invoke services (of course with the qualities mentioned above), Foster et al. came to the conclusion, that Grid services required well defined semantics as well. These semantics should assure, that all Grid services follow the same conventions for things like error notification and creation, as well as termination. They therefore proposed a set of well-defined standard interfaces, which address those conventions. This means, that a Web service, which implements those well-defined interfaces becomes a Grid service. A first specification of interfaces a Web service must implement to become a Grid service has been done in the OGSI Working Group (OGSI-WG) and published in [22]. As Web services evolved, the Open Grid Service Infrastructure (OGSI) specifications have been refactored into the WSRF [5]. That refactoring process and relations between OGSI and WSRF are explained in [4]. WSRF contains all functionality which has been specified in [22], but partitions it into five specifications and changes some syntax. Admittedly [5] is a draft document and [9] will most likely be subject to further work. This shows that the concrete Grid service is not yet clearly specified. Final interfaces may vary from those proposed in those documents, but the direction and the basic concept are clear.

3.1.3 Web Services

This section contains a short description of Web services and the underlying Simple Object Access Protocol (SOAP). Explaining the functionality and features of Web services is out of the scope of this thesis. This section only contains short summaries of technical basics. For further insight see [25] which contains detailed descriptions of protocol specifications and functionalities as well as architecture explanations.

Web services are software applications, which are identified with a Uniform Resource Identifier (URI). Each Web service implements a concrete interface which is defined with the WSDL [3]. The interfaces in WSDL are called portTypes. A client discovers the Web service and its description for example over a discovery service or a registry. The Web service description contains the information that is necessary to invoke the service. By obtaining the description, the client gains the ability to invoke the service [1]. This is done by sending a message to the Web service, which contains method name and parameters. The Web service responses that request with another Message which contains the results of the method invocation. See Figure 3.1

Web services commonly use SOAP for communication. Another protocol that may be used is eXtensible Markup Language - Remote Procedure Call (XML-RPC). SOAP is further described in section 3.1.4 *The Simple Object Access Protocol*.

It can be said, that Web services are for machines, what Web sites are for humans. A fitting example is the Google API [10], which enables applications to search the internet with the Google search engine (http://www.google.de), and enables them to use the search results. Without such a Web service, an application would have to parse the Google result page, which is quite a hard task.

[1] of course the client might have known the description without discovery

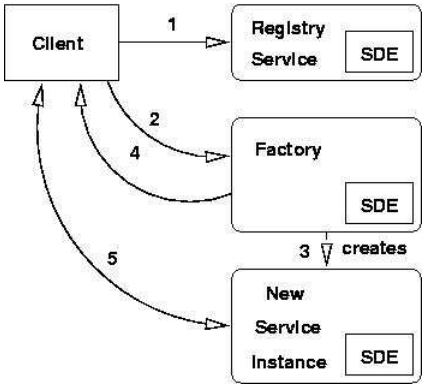

Figure 3.1: Interaction with Grid services [19]

Web services are often compared to the Common Object Request Broker Architecture (CORBA), where interfaces are defined in the Interface Definition Language (IDL) and may be implemented in any programming language. WSDL is for Web services, what IDL is for CORBA and Web services share some aspects with CORBA, but while CORBA follows a object oriented approach, Web services are, as the name says, service oriented.

3.1.4 The Simple Object Access Protocol

The SOAP [14] is a protocol for the exchange of structured and typed data, based on the eXtensible Markup Language (XML). It is a simple and lightweight message based protocol which can be used with any transport protocol (i.e. Hyper Text Transfer Protocol (HTTP)). A SOAP Message consists out of a mandatory envelope, which contains an optional header and a mandatory body element. Inside the header, optional header entries make it possible to provide additional information or metadata. The body contains body entries, which contain all essential information for the recipient.

SOAP can encapsulate remote procedure calls. Required entries for such a task are a URI which identifies the recipient, the name of the method that shall be invoked and its parameters. Applications which use SOAP for remote procedure calls, do not need to know about the implementation of the method. This is quite important for Web and Grid services.

Another of SOAPs features is an encoding mechanism for serialization of application-specific data types. As stated in [25], this mechanism might reduce development effort but may also lead to interoperability problems in heterogeneous systems.

3.2 Firewalls

The term firewall covers a large field of technologies. The original definition of a firewall is *the implementation of a security policy*. Today's firewalls vary vastly in functionality. In order to clear this up a bit, this section describes packet filter and content filters. But you should note that the concept described in chapter 5 *Concept* could also be referred to as a firewall, and it might as well be implemented in a single hardware device.

Firewalls are used to protect networks from attacks and to filter unwanted traffic. They consist out of filtering software, that screens incoming and outgoing network traffic. The traffic is filtered according to specific rules so packets will either be dropped or forwarded. There are different levels of filtering used in firewalls. The packet filter (section 3.2.1 *Packet Filter*) filters Internet Protocol (IP) packets one by one without knowledge of the content or complete data stream. Content filters are able to filter the content of several packets at once (section 3.2.2 *Content Filter*).

Usually, firewall architectures include a DMZ. The DMZ can be described as a location which is neither in the local network, nor in the external network, but next to the firewall. For the ease of illustration just imagine it as the location between two firewalls. For example, one firewall has a rule set, that allows most traffic, and one firewall has a rule set that is more restrictive and disallows a lot more connections. The place between these two firewalls is the DMZ. Resources that don't need a high level of protection but use certain protocols that would be blocked by the restrictive firewall can be placed in the DMZ for full functionality. This setup is illustrated in figure 3.2. Actually hardware firewall devices exist and their setup determines which resources should be treated as if they where in a DMZ.

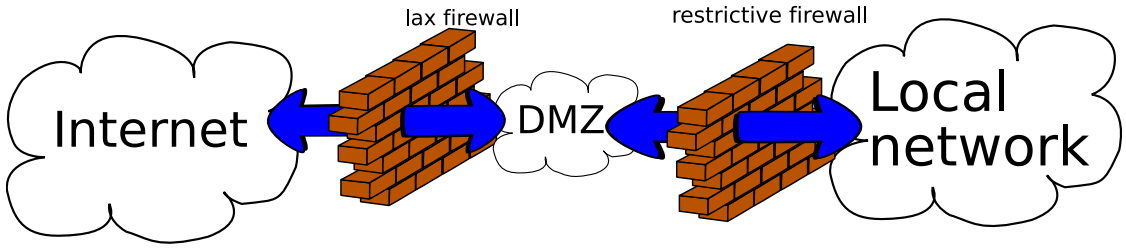

Figure 3.2: A local network, protected with two firewalls and DMZ between those firewalls.

3.2.1 Packet Filter

Packet filters are the lowest level of firewall technology. They filter IP packets, and make decisions based on rules and the contents of the IP-Header. This Header contains the IP-Address of sender and receiver, checksums and some other information (e.g. time to live, max hops). IP-packets wrap Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) packets. These protocols have additional header information which is usually used for filtering too. Packet filters have no knowledge of the protocol that is transmitted with TCP or UDP and for that reason can not filter the content. Packet filters can be used to disallow traffic from certain hosts and blocking of ports. Blocking of ports is sometimes referred to as blocking of protocols, which is not correct. Protocols are independent of the used port and can be used with any port, but they usually only use one specific port (e.g. HTTP usually uses port 80, but if a packet filter is set to block port 80, this will not disallow all HTTP traffic. A HTTP server listening on port 8080 or any other port would not be affected by that rule). For this reason, administrators tend to block as many ports as possible (e.g. all unused ports), in order to block all unwanted protocols. By opening a single port for an application, they enable every application to run over that port.

3.2.2 Content Filter

A content filter is able to filter the content of traffic that passes the firewall. It therefore collects complete data streams (which means, that it collects all packages which belong together). It needs knowledge about the protocol, which is transmitted and it must understand that protocol. Content filters can be used to remove certain parts of the content, e.g. JavaScript from a HTML document. These tasks are very complicated and usually not part of a firewall. If they are, they are restricted to filtering of a few protocols like HTTP or data formats like HTML. The content filtering is usually done by other special software (e.g. Virus scanners, SPAM-filters, Intrusion Detection Systems etc) which is employed after the use of a firewall, respective a packet filter. Figure 3.3 illustrates on which network layers of the TCP/IP modell the different filters work. You should note, that the presented concept and the content filters share some common ground and the concept might be integrated in firewalls as well.

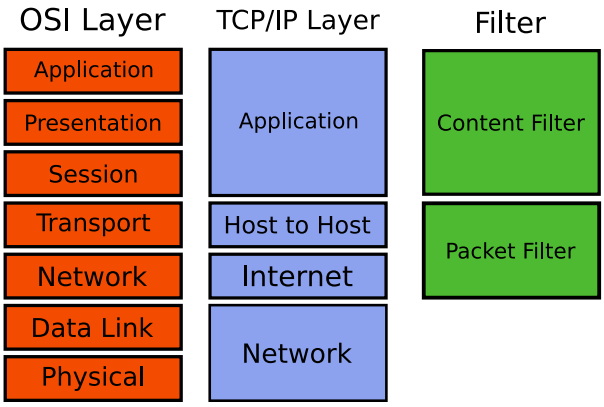

Figure 3.3: Network layers and corresponding filters.

Chapter 4

Similar Software products

This chapter will give a short overview over software products, which have similarities with the ALG, designed as part of this thesis. This comparison has already been done as a part of the requirement analysis for the ALG [12]. During the research for [12] two relevant software products have been identified. The first product is the *Xtradyne Web Service Domain Boundary Controller* and the second product is the *MultiNet iSecure Web Services Gateway*. Both products aim at similar problems as the ALG but do not target Grid services. The scope of both products only includes ordinary Web services.

The Xtradyne Web Service Domain Boundary Controller (WS-DBC) is an Application Level Gateway based approach to secure Web services transparently. This is exactly what the designed ALG should do for Grid services. WS-DBC features a SOAP content inspection and Security Assertion Markup Language (SAML) assertions. It is able to provide a secure gateway to Web services. The concept and the features of WS-DBC are described in [2, 23, 24]. According to those references, WS-DBC does not provide all features which are needed for Grid services. The following quote is taken from [12]:

> WS-DBC and Fireblade both share the same approach to secure Web/Grid Services. WS-DBC already features a lot of authentication methods like X.509 or SAML, but its scope is limited to Web Services, while Fireblade addresses Grid Services. The main difference is, that Grid Services often use protocols, that are not SOAP based, e.g. GridFTP or GASS. Those protocols can transfer any type of data between to nodes and will use SOAP messages only to initialise a data transfer. WS-DBC is not able to handle such data transfers. Additionally WS-DBC only validates SOAP-Messages against given schema files. It does not provide functionality to change message contents (e.g. for mapping of users from internal to external users).

The designed ALG does not provide all these functionalities that have been identified as missing features of WS-DBC, but instead it will make it easy to develop and add new features on demand (see Chapter 6 *Realization*).

The second software product which has been compared to the ALG is the *MultiNet iSecure Web Services Gateway* [15, 21]. Like WS-DBC and the solution proposed in this Thesis, it is based on an ALG architecture. The following statement is taken from [12]:

> The MultiNet iSecure Web Services Gateway is, like Xtradyne WS-DBC an ALG-based approach for securing Web Services. It addresses several security issues for Web Services, but it does neither provide end-to-end security like WS-DBC, nor is it able to handle protocols like GridFTP or GASS. It does not offer any authorization methods as well. MultiNet iSecure WebService Gateway is meant as a complement to Software that possesses these features.

The fact that at least two products identified similar problems for Web services and try to solve those problems with a similar concept shows, that the ALG-based approach goes into the right

direction. However both products lack certain features that are necessary in order to provide a security gateway for Grid services.

Chapter 5

Concept

Different concepts may provide a solution for the problem described in chapter 2 *Problem Description*. This chapter will give some insight into the chosen concepts and it will present some advantages over other concepts, which have also been considered.

The following concepts have been considered:

- SSH tunneling
- Virtual Private Network (VPN) plus Grid nodes in the DMZ
- Application Level Gateway

5.1 Comparison of different concepts

This section briefly describes the different concepts and compares them with each other. Every subsection features a table which shows the major advantages and disadvantages from the different concepts.

5.1.1 SSH tunneling based approach

The first concept, *SSH tunneling*, was the first solution which was considered, when SISTEC and SCAI encountered problems with firewalls while creating a VO. The idea behind this concept is, to create Secure Shell (SSH) tunnels for all necessary connection paths. This would require machines in the local networks of each participating institution, which would be starting points of such tunnels. Such an approach has been made by HP Laboratories Palo Alto in 2002 (see [11][1]) with Globus Toolkit 2. The following quote from [11] describes the concept briefly. See table 5.1 for an overview about this concept.

> The basic idea behind port tunneling is to open an encrypted connection to a remote host and forward all local connections to certain ports to referred ports at the remote host (or vise versa).

Advantages

SSH is a widely spread protocol which can be considered as stable. Even inexperienced system administrators will possess knowledge about SSH and know how to setup SSH-Tunnels. Additionally it is powerful and can enable any Grid communication between different nodes through SSH-Tunnels. There are no limitations on the protocols it supports, because it works on TCP/IP level. All traffic transmitted through those tunnels is encrypted and thus resistant against wiretapping.

[1] Some more concepts are discussed in [11], but considered significantly inferior to the Tunneling approach.

Advantages	Disadvantages
all features	complex setup
protection against wiretapping	complex configuration
stable	no reusable setup
no new software needed	no reusable configuration
	difficult automation

Table 5.1: Advantages and disadvantages of a SSH tunneling based approach

Advantages	Disadvantages
protection against wiretapping	limited to job submission
no new software needed	represented resources are invisible for users
not to complex configuration	difficult automation

Table 5.2: Advantages and disadvantages of a VPN based approach

Disadvantages

Even though the setup process for a single SSH-Tunnel is quite easy, the configuration and setup needed for this approach becomes a major drawback. At first the number of needed ports can be extremely high and it might be necessary to open a large number of tunnels. Because the tunnels may be accessed by anyone who has access to the start-point of the tunnel, it might be a security risk to leave the tunnels open when not needed. This means that they have to be opened whenever the Grid should be used. A problem might rise because that process can not easily be automated (Due to security policies of the participants). Every participating institution needs access to all other institutions in order to open the tunnels (which might be a security threat). This means that for each usage of the joined Grid, the users would have to talk to the responsible system administrators in order to initialize the setup. Another burden is the configuration of the Grid nodes. They would have to be configured in a way which makes them use the SSH-Tunnels instead of a direct connection to a certain machine.

5.1.2 VPN based approach

The second concept, using *VPN and Grid nodes in the DMZ* would require less configuration than SSH Tunneling. It features some similarity with the finally chosen concept. All participating Institutions would place one Grid node in their DMZ, and configure it in a way that communication from that Grid node into the local network would be possible. In order to fulfill services, it should invoke services from the other grid services. All those Grid nodes placed in the DMZ could than be connected with a VPN in order to disable access from unauthorized users. Although this solution seems to be much better than the SSH tunneling approach, it has its disadvantages as well.

Advantages

The setup for this concept would be significantly smaller than for the SSH-tunneling approach. Each institution should only have to configure one additional machine in the DMZ for Grid communication and VPN. Like SSH-Tunneling, this solution offers encryption for the traffic.

Disadvantages

The described setup would not enable all possible Grid communications. It would be limited to job submission. Services which require direct connections between several nodes might become impossible.

Advantages	Disadvantages
transparent communication	limited to supported protocols
easy setup	configuration might become hard
possibility of content filtering	eventually a bottleneck
possibility of content validation	

Table 5.3: Advantages and disadvantages of an ALG based approach

Furthermore, users would not be able to see the Grid resources which are represented by the nodes in the DMZ. This would disable the selection of concrete resources for a computation. Another drawback is, that the VPN can most likely not be opened automatically due to security policies of participating institutions (similar to the identified disadvantage of the SSH-Tunneling approach).

5.1.3 ALG based approach

An ALG is a system, which understands an application level protocol and filters network traffic on that protocols layer. It is placed between the actual communicating resources which instead of sending their data directly to their communication partner, send it to the ALG. The ALG acts as if it would be the original communication partner, accepts and filters the messages and relays them to their destination. The same happens with responses. This is similar to the content filter of a firewall (see section 3.2.2 *Content Filter*). Such a setup would enable transparent communication between all involved Grid nodes, which means that neither applications running in the Grid, nor the Grid nodes themselves would need significant configuration changes. However the installation and configuration of such an ALG might become difficult. A drawback for this concept is, that an ALG needs high level knowledge about all protocols used for Communication between Grid nodes. In theory, Grid nodes can communicate with any protocol. Certain Grid services might even introduce their own protocol for some communication. Grid nodes which are connected with an ALG have their communication limited to those protocols, the ALG understands and supports.

Advantages

Setting up an ALG is generally not a hard task (except for configuration, which might become difficult). Only one additional machine is needed in the DMZ and the nodes must be configured to talk to that machine instead of the real Grid nodes. This solution would be transparent and would work fully automated, which means that Administrators would not have to open ports on demands or setup tunnels each time a computation runs. Additionally an ALG based approach adds the ability to filter and validate transmitted data and metadata. This might increase the security of a Grid significantly.

Disadvantages

The greatest disadvantage of this concept is, that it might become a bottleneck for communication processes (all communication has to be done with the ALG). An ALG based setup would be limited to a subset of all possible communication protocols, because the support for each protocol must be implemented, too (unlike the SSH approach which works on a lower level and has no knowledge about communication protocols). Furthermore the configuration of the ALG might become a hard task.

5.2 Application Level Gateway

Neither the setup of SSH tunnels, nor the utilisation of additional Grid nodes in the DMZ provided a reusable and generally applicable solution for the problem described in chapter 2 *Problem Description*.

The third and most sophisticated concept leads to such a reusable solution. This is the chosen concept which introduces an *Application Level Gateway* and thereby enables a transparent communication between all grid nodes.

An ALG for Grid services must understand Grid service messages, which are SOAP messages (see chapter 3 *Technical basis*). But unlike a content filter, the ALG would not only have to filter those messages, but it would have to provide proxies for the requested services in many cases. This is due to the fact, that services are not restricted to SOAP messages for their communication. In order to support non-SOAP communication between the Grid nodes, the ALG would have to identify the initialization of such communications (which is done via SOAP) and it has to create a proxy for that communication on the fly. The SOAP messages which initialises the communication has to be changed, in order to force the usage of the created proxy. In order to create and provide those proxies, the non-SOAP protocols have to be supported by the ALG. If service requests would initialise communication over protocols the ALG does not understand, the ALG should not forward those messages and it should inform the client about that error. Obviously, this would require the ALG to have knowledge about the initialization process of all protocols. A requirement which cannot be fulfilled due to the high number of possible protocols. Instead of sending error messages whenever services are requested which may use unsupported protocols, those services should not even be exposed to the outside. The decision which services are going to be exposed to the outside has to be done by a system administrator and can not be done automatically.

Figure 5.1 shows how an ALG which is placed in the DMZ of a institution can enable Grid communication. Note that the DMZ is represented as the space between two firewalls as described in section 3.2 *Firewalls*.

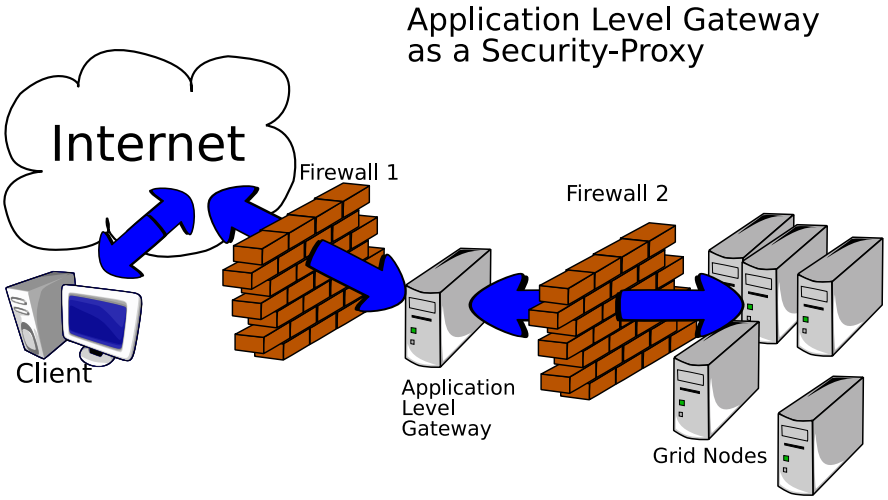

Figure 5.1: Application Level Gateway enabling a VO with firewall-protected local networks

As already stated in chapter 2 *Problem Description*, a firewall can deny incoming traffic even from the resources in the DMZ. In that case two ALGs are needed to enable Grid communication over a firewall. ALG one is placed in the DMZ (like in figure 5.1) and ALG two is placed in the local network. ALG one has to cache all incoming requests because it is not able to forward them through the firewall. Nevertheless it is accessible from ALG two, which may query ALG one for cached messages. ALG two than forwards the messages to the appropriate resources and sends the answers to ALG one. ALG one can then forward those answers to the original recipients. This setup is shown in figure 5.2 *Additional polling ALG for extremely restrictive firewall environments*. The following quote is taken from the Catalogue of Requirements [12].

> "An ALG is placed in the DMZ and reacts to requests from the Internet as if it was the requested resource. Therefor it acts as a client for the requested resource and forwards

the requests to that resource. After it has received an answer it will forward the answer to the original client.

The ALG must be able to understand the protocols which are used to determine which resource has been requested and must be configured to allow or disallow certain requests. Due to the high number of Grid communication protocols and the fast development in that area, an ALG for Grid Services must be highly extensible."

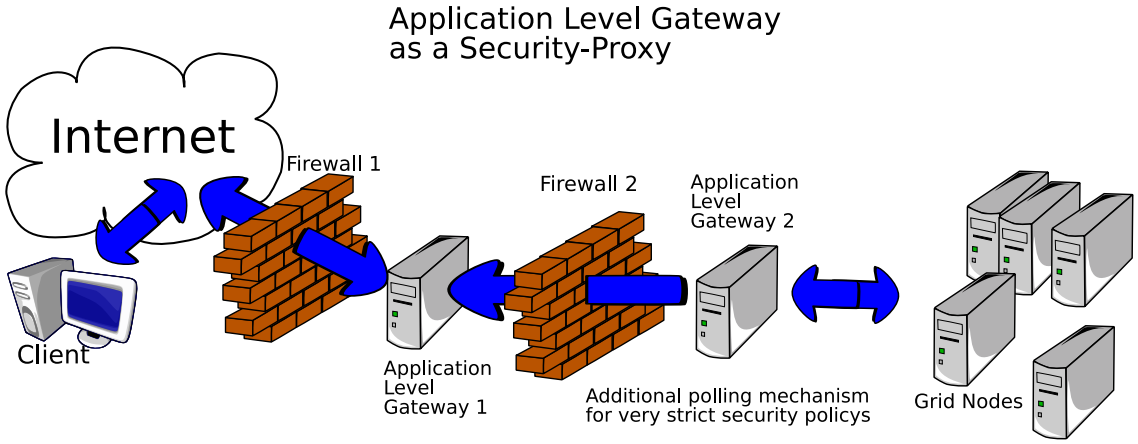

Figure 5.2: Additional polling ALG for extremely restrictive firewall environments

This concept has some advantages over the other concepts. It does not require any configuration except for the configuration of the ALG. None of the Grid nodes needs knowledge of the ALG. All types of service can be made possible with a sophisticated ALG. Even though this might be a hard task in many cases. Of course this concept has a disadvantage, too. It has a performance bottleneck. All requests to a firewall protected node have to be processed by the ALG, which creates latency and lowers the throughput.

5.3 Requirements

The requirements for this project are documented in the Catalogue of Requirements [12]. This section summarizes the requirements from that document which contains user profiles, prerequisites and duties, dependencies, functional requirements and demands on quality. The functional requirements are modelled as Use-Cases which are included and explained in section 5.3.4 *Use-Cases*.

5.3.1 User Profiles

As stated in [12], the typical user of the ALG is an experienced system administrator. Knowledge of firewalls and proxy-servers is required as well as knowledge about the institutions network structure and security policy. In order to enable Secure Socket Layer (SSL) encryption, the administrator must be able to acquire all certificates of involved Grid nodes. Further requirements on the user might arise through further extensions (units or plugins) of the ALG (e.g. a Supplier which uses CORBA to deliver messages will most likely require CORBA knowledge).

Those users who will just run computations on the Grid or developers of Grid applications should never have to deal with the ALG. This means that there are no requirements on the knowledge of those people.

5.3.2 Prerequisites, Duties and Dependencies

The ALG should integrate into existing environments easily and it has to provide high availability (because it provides the only way to access certain Grid resources). It should be able to do load balancing, because it is performance critical. Additionally, it must be very easy to add new features and support for new protocols to the ALG. And like every software product, the source code has to be well documented. The process of extending the ALG has to be documented well to. [2]

5.3.3 Demands on Quality

The demands on quality identified in [12] are the following:

> The ALG must run on normal out of the box computers (2005 - Pentium IV, 512M RAM). Load sharing must be possible.
>
> Downtimes must be reduced to a minimum. Availability of the ALG is of a high priority.
>
> The ALG must be configured by an easy to use Graphical User Interface to avoid miss configuration. Password protection for the Administration is an absolute must. All settings can only be made by the Administrator.
>
> Security is one of the major features of the Application Level Gateway. Security Policy must be to disallow everything from the beginning and allow certain specified connections later.

However those demands do not apply to the software version developed as part of this Thesis. This is due to the fact that it would not be possible to develop a Software addressing all identified requirements within the time which is available for a Bachelorthesis. Especially performance and GUI configuration have a low priority for a first implementation. They will be required in further advanced versions nevertheless.

5.3.4 Use-Cases

All Use-Cases in this subsection (Figures 5.3, 5.4, 5.5 and 5.6) are taken from the Catalogue of Requirements [12] and created by Roland Gude and Thijs Metsch, as part of the requirement analysis. The explanations are summaries of those from [12].

The first use-case diagram (figure 5.3) shows the basic functionality of the ALG. It shows how the different actors (Client, Administrator, Grid and Developer) interact with the ALG. The Client can be any source of a Grid service request, e.g. a Grid service from remote Grid nodes. Grid describes the local Grid nodes which should be protected by the ALG. Developers are those persons who actually work on the ALG (e.g. create units or plugins), but not the developers of Grid services. The Administrator is the maintainer of the ALG, who installs and activates units and plugins or defines the configuration. The *use handler unit* use-case is modelled in figure 5.5 and 5.4. The requests are accepted from the ALG (actually from the consumer), the deployed handler units inspect, validate and modify the MessageContext for that request (and possibly the request itself), and the ALG invokes the Grid service (using the supplier).

Figure 5.4 shows the interaction of the ALG's core with the different actors. Possible actors are cores from other ALG installations, Administrators or units. Administrators configure the core (e.g. the use-case order describes that they define the order in which the handler units should be used) and other cores just interact in order to achieve load balancing. The interaction between units and the core is a more complex. Whenever a unit has finished handling of a request, it passes the MessageContext back to the core. The core now selects the next unit (which is defined by the order of the units) and *sends* the MessageContext to that unit.

[2] A document providing a detailed description on the process of unit/plugin-development is currently under development ([13]). This thesis provides some information on the unit/plugin-development process in section 6.3 *Plugin Architecture*.

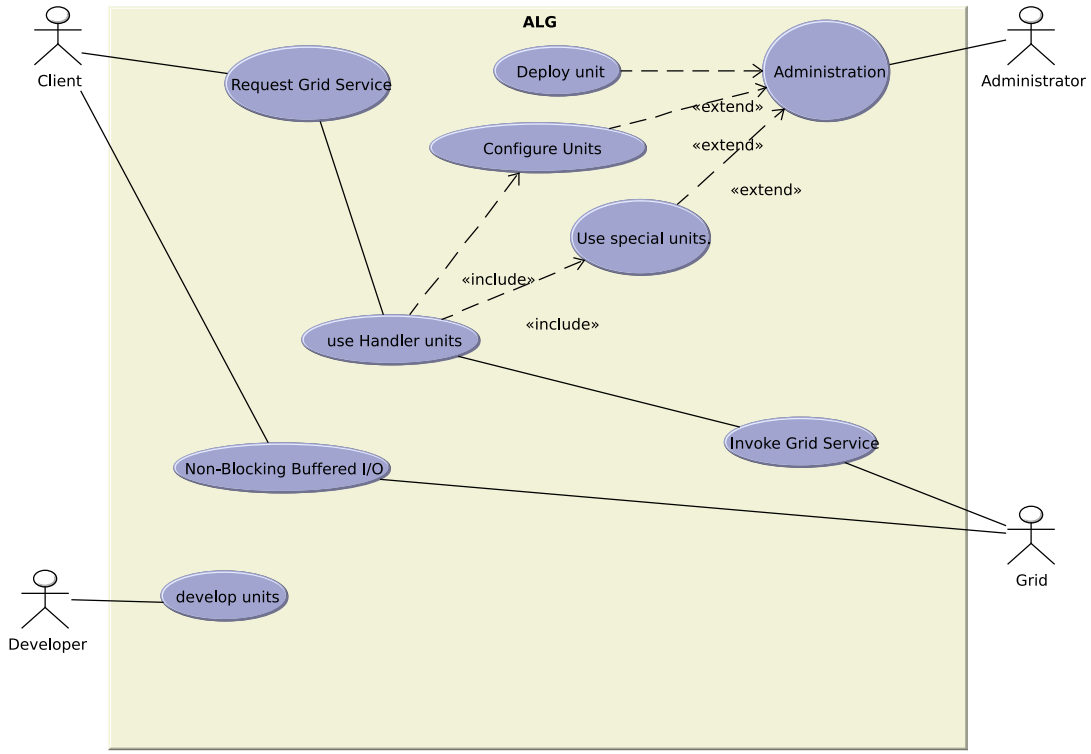

Figure 5.3: Use-Case: Basic functionalities of the ALG

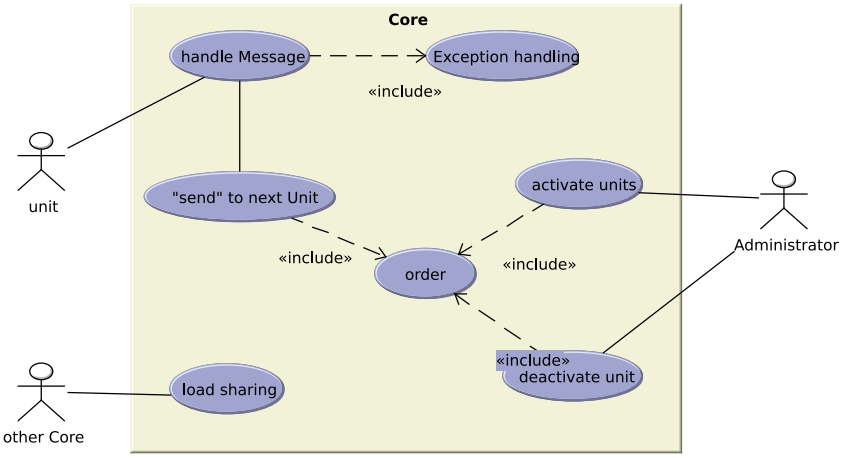

Figure 5.4: Use-Case: Functionalities of the ALG-Core

The unit use-case diagram (figure 5.5) shows how the ALG units works and interact with other actors. The actors in this context may either be the core of the ALG, an administrator or the plugins which extend the unit. The core asks for handling of a MessageContext. The unit then modifies the MessageContext (not necessarily the message) and utilizes appropriate plugins for further tasks. When all plugins have done their tasks, the MessageContexts history is updated and the Message is finalized, which means it is handed back to the core.

The last use-case diagram (figure 5.6) shows the interaction between units and their plugins from

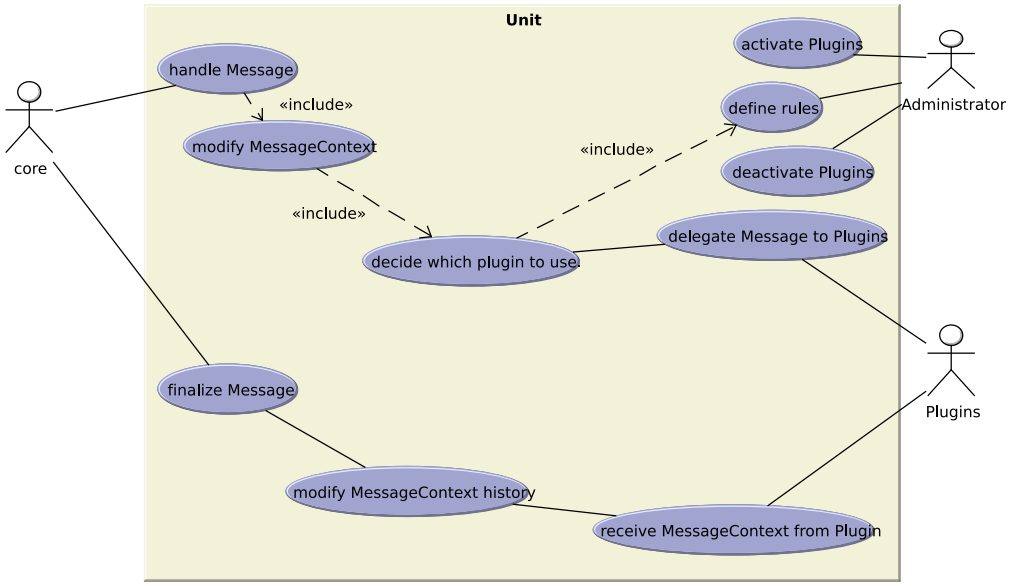

Figure 5.5: Use-Case: Functionalities of the ALG-Units

the perspective of the plugins. The Context receives a MessageContext from the unit it extends and handles that MessageContext. When this is done, it hands back the MessageContext to the unit. The plugin may be configured by an administrator of course.

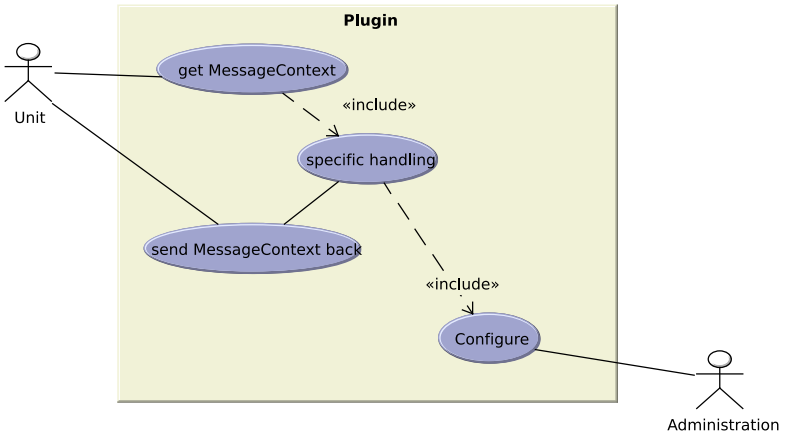

Figure 5.6: Use-Case: Functionalities of the ALG-Unit-Plugins

Chapter 6

Realization

This chapter contains information about the software development and the developed software, but it is neither a complete requirements analysis nor a concept and design manual. At DLR, both documents are being worked on currently, and this chapter will refer to them several times and even quote certain parts of the current drafts. These documents have not yet been published, so this Thesis contains the most detailed information about the developed solution, that is currently available. Additionally this chapter contains information about major software products that have been used for the realization of the ALG, namely AXIS, an open-source java based SOAP engine and the Java Plugin Framework (JPF) which has been employed to guarantee the demanded high grade of extensibility for this software.

6.1 Basic Software Architecture

The developed ALG has a highly extensible architecture, shown in figure 6.1. On top of an application core several units are deployed to handle the SOAP messages which reach the ALG. Two units have a special role in this architecture, namely the consumer unit and the supplier unit. The consumer unit is the part of the ALG where new SOAP requests arrive and responses are forwarded to their original destination. The supplier unit is the part which forwards checked messages to the original recipient and accepts the answers. The units between consumer and supplier are the handler units. They can be used for all other tasks that have to be done whenever a message arrives (e.g. validation of user names). While no handler units are required for the ALG to use (though that might not make to much sense), consumer and supplier unit are mandatory.

units may be further extended with plugins. Some possible plugins are marked with dotted lines in figure 6.1. The Consumer unit could be extended with several plugins. For example one which receives messages over HTTP, one that receives them over email or one which queries them from another ALGs cache. A unit which is intended to check whether a message is acceptable might contain plugins which check the user name as well as plugins which check whether the requested resource should be visible for that user. It might as well contain plugins which contain certain checks that are necessary in order to support certain Grid Protocols like GridFTP.

Messages are represented together with some status information as a MessageContext instance. units that have performed their tasks with a message, update the history of the associated MessageContext and hand the message over to the core. The core evaluates the MessageContext instance (e.g. checks the history and status information) and selects the next unit which should handle the message. The selected unit then performs its tasks and hands the message back to the core afterwards.

You can imagine the ALG as a bidirectional pipe of handling units, managed by a core. The internal communication between the core and the units is modeled in the UML sequence diagram figure 6.2. In figure 6.1 the consumer unit would be the entrance point of the pipe and the supplier

would be the pipes exit (and entrance point for the responses). A message arriving at an ALG with those units shown in figure 6.1 would pass the following stations in that order:

1. request received by the consumer unit.
2. consumer unit creates MessageContext from the request.
3. consumer hands the MessageContext to the core.
4. core checks status and history of the MessageContext and hands it to the next unit, which is the authorization unit.
5. authorization unit checks the MessageContext with the Virtual Organization Membership Service (VOMS) plugin.
6. authorization unit updates the MessageContext (history and status) and hands it to the core.
7. core checks status and history of the MessageContext and hands it to mapping and validation unit.
8. the MaV unit validates the request and maps from external to internal users, updates the MessageContext and hands it to the core.
9. core checks status and history of the MessageContext and hands it to the supplier unit.
10. the supplier unit delivers the request.
11. the supplier receives the response, updates the MessageContext and hands it back to the core.
12. core checks status and history of the MessageContext and hands it to the mapping and validation unit.
13. the MaV unit validates the request and maps from internal to external users, updates the MessageContext and hands it to the core.
14. core checks status and history of the MessageContext and hands it to the authorization unit.
15. authorization unit checks the MessageContext with the VOMS plugin.
16. core checks status and history of the MessageContext and hands it to the consumer unit.
17. the consumer delivers the response.

6.2 Used Software

This section describes briefly which major software products have been employed in order to design and realize the developed ALG. This is not a list of all used libraries, but contains only those software products that had significance when it came to the overall software design. The software has been developed in the Java programming language with the use of the Eclipse IDE and Subversion as a version management tool.

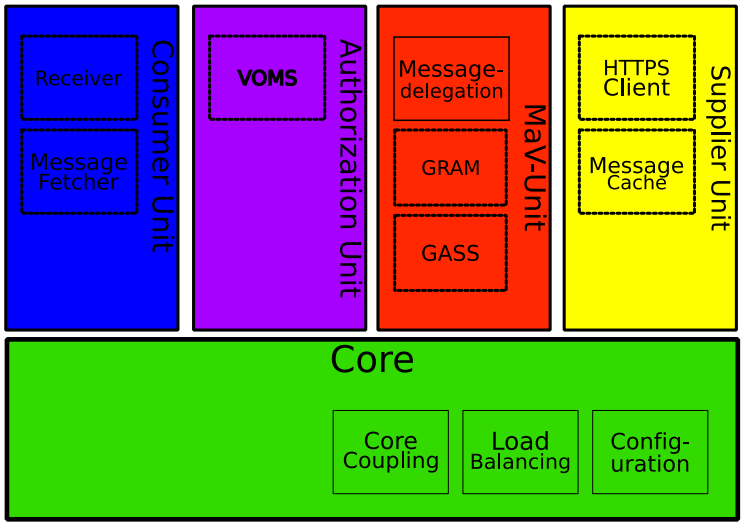

Figure 6.1: Basic Architecture of the developed ALG.

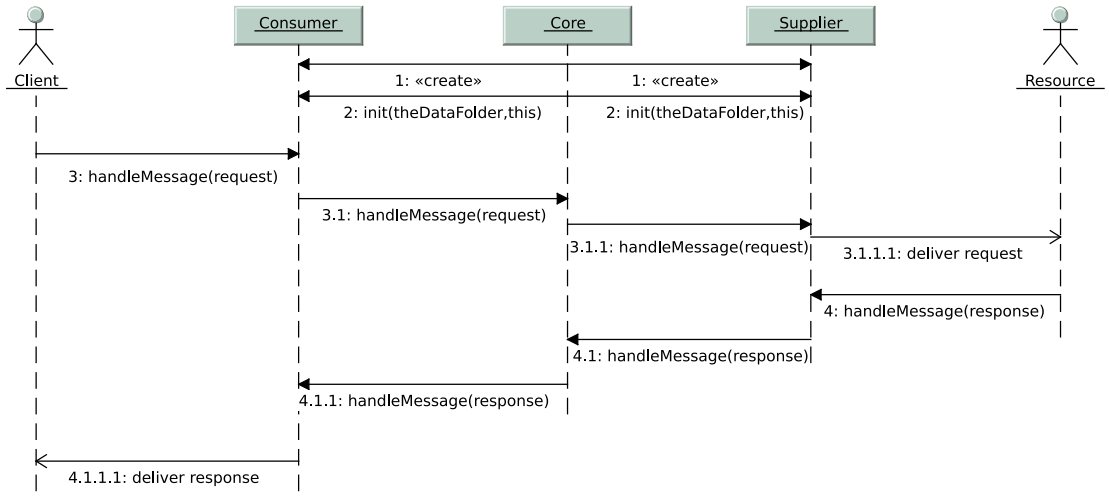

Figure 6.2: UML sequence diagram of the communication process with between a client, the ALG and a resource.

6.2.1 Axis

Axis [1] is an open-source java SOAP implementation. All messages, sent in Grid service context are such SOAP messages. Axis is used to represent those SOAP messages to the ALG. It is used for everything that has to do with SOAP in the ALG, like generation of error messages. The ALG can access the messages by utilizing the axis API. This renders XML parsing for message validation unnecessary.

The following quotation is from the Axis User Guide at [1].

> Axis is essentially a SOAP engine – a framework for constructing SOAP processors such as clients, servers, gateways, etc. The current version of Axis is written in Java, but a C++ implementation of the client side of Axis is being developed.

All SOAP messages are represented to the ALG using *de.dlr.fireblade.message.MessageContext* (see section 6.4.1 *MessageContext*. It uses the *javax.xml.soap.SOAPMessage* implementation from

Axis (*org.apache.axis.Message*) to do so. This enables the ALG to create and parse SOAP messages.

6.2.2 Java Plugin Framework

The Java Plugin Framework enables the development of plugin based architectures. It provides classes and interfaces to write plugins, as well as a central plugin manager and a plugin registry. It is able to discover plugins that are installed and to check whether the dependencies (upon other plugins) for a discovered plugin have been met.

This is what the project Web site [17] states about JPF:

> "The Framework implements the runtime engine that dynamically discovers and runs plugins. A plugin is a structured component that describes itself to the Framework using a manifest file. The Framework maintains a registry of available plugins and the function they provide (via extension points and extensions).
>
> A general goal of the Framework is that the application (and end user using it) should not pay a memory or performance penalty for plugins that are installed, but not used. A plugin can be installed and added to the registry (even when application is running), but the plugin will not be activated unless a functionality provided by the plugin has been requested according to the user's activity."

JPF is used to achieve the high grade of demanded extensibility. The base classes of the ALG (*de.dlr.fireblade.core.Core*, *de.dlr.fireblade.core.unit.Unit*, *de.dlr.fireblade.unit.plugin.Plugin*) are all derived from the JPF *org.jpf.plugin.Plugin* class. In order to extend the core, a new unit has to be written by subclassing *de.dlr.fireblade.core.unit.Unit* and its manifest file is needed as well. In order to extend a unit (like the consumer or the supplier, though the reference implementations do not define any extension points and due to this are not extensible), a new plugin must be written by subclassing *de.dlr.fireblade.unit.plugin.Plugin* and its manifest file must be created. The more detailed documentation [13], including coding standards is currently under development.

6.3 Plugin Architecture

In order to extend the ALG, a developer has two possibilities (besides modification of the source code of the ALG itself). He can either develop a unit or a plugin for the ALG. The difference between a unit and a plugin is simple. units extend the ALG, plugins extend a certain unit. A simple example would be the following. A developer who wants to add the functionality to check the user name included in a Grid request against valid user names would write a unit to do this task. For a first deployment he only needs to check if the user name is included in a certain file. In order to extend this functionality later, he defines a extension point for his unit and develops a plugin which attaches to that extension point and checks a given user name against a list it gets from a certain file. Some time later he realizes that checking the user name against user names in a file is not sufficient for his needs any more. He now creates a new plugin extending his unit, which checks a user name against a LDAP-Server (or something similar).

The developed implementation of the ALG includes some abstract classes which enable the creation of new units and Plugins.

- *de.dlr.fireblade.core.unit.Unit*
- *de.dlr.fireblade.core.unit.plugin.Plugin*

In order to create a new Unit, a developer has to subclass *de.dlr.fireblade.core.unit.Unit* and implement the abstract methods. Furthermore he needs to create a manifest file (see 6.2.2 *Java Plugin Framework*) for this Unit.

A unit might look like this:

```
import de.dlr.fireblade.unit.Unit;
import de.dlr.fireblade.unit.exceptions.UnitException;
import de.dlr.fireblade.core.CoreInterface;
import org.java.plugin.PluginDescriptor;
import org.java.plugin.PluginManager;

public class MyNewShinyUnit extends Unit {
    //A parameter name for some parameter which is read from the configuration file
    private final static String PARAM_PARAMETERNAME = "my.new.unit.param";

    //the default value for the parameter
    private final static String DEFAULT_PARAMETERNAME = "defaultvalue";


    //private String myVariable;

    //A Constructor
    public MyNewShinyUnit(PluginManager mgr ,PluginDescriptor desc) {
        super(mgr, desc);
    }

    //Initialisation of the unit, which must be called before it is usable
    public void init(File dataFolder, CoreInterface core) throws UnitException{
        super.init(dataFolder, core);

        //extracting the parameters from the configuration properties
        try {
            if (myProperties.getProperty(PARAM_PARAMETERNAME == null) {
                throw new NullPointerException();
            }
            myVariable = myProperties.getProperty(PARAM_PARAMETERNAME);
        } catch (NullPointerException e) {
            myVariable = DEFAULT_PARAMETERNAME;
        }
    }

    //implementing those abstract methods


    /**
     *
     * message handling that is done for every incoming message
     * before any plugins can work with the message
     *
     *  @param answer the message that should be handled
     * @throws UnitException thrown when something goes wrong.
     *
     */
    protected void doInitialRequestHandling(MessageContext request)
                                                        throws UnitException {
        //Some Implementation goes here
    }
```

```
/**
 *
 * hands an incoming Message (requests) to the appropriate
 *  plugins and let them do some handling
 *
 * @param request the message that should be handled
 * @throws UnitException thrown when something goes wrong.
 */
protected abstract void delegateRequestToPlugins(MessageContext request)
                                                        throws UnitException{
    //Some Implementation goes here
}

/**
 *
 * Final message handling that is done after all plugins did something with
 * the incoming message (request) and just before the message is send
 * back to the core
 *
 * @param request the message that should be handled
 * @throws UnitException thrown when something goes wrong.
 */
protected abstract void doFinalRequestHandling(MessageContext request)
                                                       throws UnitException {
    //Some Implementation goes here
}

/**
 *
 * message handling that is done for every returning message
 * before any plugins can work with the message
 *
 * @param answer the message that should be handled
 * @throws UnitException thrown when something goes wrong.
 *
 */
protected abstract void doInitialAnswerHandling(MessageContext answer)
                                                       throws UnitException {
    //Some Implementation goes here
}

/**
 *
 * hands an returning message (answer) to the appropriate
 *  plugins and let them do some handling
 *
 * @param answer the message that should be handled
 * @throws UnitException thrown when something  goes wrong.
 *
 */
protected abstract void delegateAnswerToPlugins(MessageContext answer)
```

```
                                                        throws UnitException {
        //Some Implementation goes here
    }

    /**
     *
     * Final message handling that is done after all plugins did something with
     * the returning message (answer) and just before the message is send
     * back to the core
     *
     * @param answer the message that should be handled
     * @throws UnitException thrown when something goes wrong.
     */
    protected abstract void doFinalAnswerHandling(MessageContext answer)
                                                        throws UnitException {
        //Some Implementation goes here
    }

    //Now there are some further methods which will not yet
    //be called anyway and might be removed from the software someday
        /**
     * Activate a deployed plugin
     *
     * @param plugin plugin that should be activated
     * @throws PluginException
     */
    protected abstract void activatePlugin(de.dlr.fireblade.unit.plugin.Plugin plugin)
                                                      throws PluginException {
        //Some Implementation goes here. Probably Empty.
    }


    /**
     * Deactivate a deployed plugin
     *
     * @param plugin plugin that should be deactivated
     * @throws PluginException
     */
    protected abstract void deactivatePlugin(de.dlr.fireblade.unit.plugin.Plugin plugin)
                                                      throws PluginException {
        //Some Implementation goes here. Probably Empty
    }



}
```

The manifest file must be named unit.xml and must be placed (together with the units classes) in the *units* folder of the ALG. A manifest file will look like this:

```
<?xml version="1.0" ?>
<!DOCTYPE plugin PUBLIC "-//JPF//Java Plug-in Manifest 0.2"
                                  "http://jpf.sourceforge.net/plugin_0_2.dtd">
```

```
<plugin id="myShinyNewUnit" version="0.0.1" class="mypackage.MyShinyNewUnit">
    <requires>
        <import plugin-id="de.dlr.fireblade.core"/>
    </requires>
    <runtime>
        <library id="myShinyLibrary" path="classes/" type="code">
            <doc caption="API documentation">
                <doc-ref path="api/index.html" caption="javadoc"/>
            </doc>
        </library>
    </runtime>
    <extension plugin-id="de.dlr.fireblade.core" point-id="Handler"
                                                 id="myShinyNewUnit">
        <parameter id="class" value="mypackage.MyShinyNewUnit"/>
        <parameter id="name" value="myShinyNewUnit"/>
        <parameter id="description" value="Does something really nice"/>
    </extension>
    <extension-point id="myPluginsMayConnectHere">
        <parameter-def id="class"/>
        <parameter-def id="name"/>
        <parameter-def id="description" multiplicity="one"/>
    </extension-point>
</plugin>
```

The unit described by this manifest file will be known to the ALG by its plugin id, which is *myShinyNewUnit* (defined in line2). It does only require the core of the ALG and provides its classes as a Library. The part surrounded by ¡extension¿ and ¡/extension¿ defines, to which extension point this unit connects (which part of the ALG is extended by the Unit). For units this must always be the core. The core offers three Extension points. All of them might be used by a Unit. units which connect to the extension point *Consumer* or *Supplier* must implement the corresponding interface. Ordinary units will extend the *Handler* extension point. This unit does not only extend the core, but it does also provide an extension point for Plugins.

Every unit has its own data folder, where it may store its configuration and other data it needs. This folder is a sub folder of the ALGs *data* folder is named with the value supplied as *plugin-id* in the units manifest file. For configuration of units, a file called *unit.properties* should be placed in that folder. This file is a standard Java-Properties file (Syntax: Parameter name=value). The properties set in this file will be available to the unit under the variable *myProperties*.

The creation of plugins is more or less the same as the creation of Units. The plugins are subclassed from *de.dlr.fireblade.core.unit.plugin.Plugin* and implement the abstract methods of that class. The manifest file differs slightly from the manifest file for Units. A plugin may not connect to any of the extension points of the core. It must connect to an extension point which is defined by a certain unit. A plugin may not define further extension points [1]

6.4 Implementation of the ALG

This section will provide some details about the software that has been developed as a part of this thesis. Three main parts of the ALG have been implemented. These are the core (section 6.4.2 *Core*), the consumer unit (section 6.4.3 *Consumer*) and the supplier unit (section 6.4.4 *Supplier*). Some other minor parts have been developed. These are the MessageContext (section 6.4.1 *MessageContext*) and some abstract classes and interfaces for units and plugins. Table 6.1 shows which

[1] Although this is technically possible, it is considered inconvenient.

classes are part of the final software. A more detailed documentation can be found in the JavaDocs. They are generated directly from the source code and its comments and provide the complete API.

Class name	abstract	interface	implements	function
Boot	no	no	—	Initialization
CoreInterface	no	yes	—	core Interface
Core	no	no	CoreInterface	core functionalities
CoreTest	no	no	—	JUnit Test for the core
CoreException	no	no	—	Exception thrown by core
MessageContext	no	no	—	message and context information
MessageType	no	no	—	distinction requests/answers
ValidationStatus	no	no	—	validation process monitoring
MessageHistory	no	no	—	handling process monitoring
Entry	no	no	—	MessageHistory entries
EntryType	no	no	—	Typing of history entries
AbstractUnitTest	yes	no	—	Abstract JUnit TestCase
Unit	yes	no	—	Base class for all units
Consumer	no	yes	—	Consumer Interface
UnitException	no	no	—	Exception thrown by units
Plugin	yes	no	—	Base class for all plugins
PluginException	no	no	—	Exception thrown by plugins
Supplier	no	yes	—	Supplier Interface
Receiver	no	no	Consumer	Consumer unit
ConsumerServlet	no	no	—	receive HTTP requests
BasicSupplier	no	no	—	deliver SOAP messages

Table 6.1: Main classes in the final software

6.4.1 MessageContext

The MessageContext class wraps the Axis *SOAPMessage* objects, in order to provide additional functionality. It is used to provide context information about messages. These context information is a messages type, like request or response, information about the validation process and its status and a history, where units can provide information about checks that have been passed or failed. MessageContext instances are created, when the consumer unit receives SOAP requests, and they are destroyed as soon as the request has been answered. That means, that request and response reside in the same MessageContext.

ValidationStatus

The ValidationStatus is used to distinguish between MessageContexts which passed all checks, failed at least one check or still require further checks. The following states exist:

- VALIDATION_IN_PROGRESS
- VALIDATION_SUCCESSFUL
- VALIDATION_FAILED
- VALIDATION_ERROR_MESSAGES.

While the first three status types are self explanatory, the last one needs some additional explanation. If a message has failed validation, the status is changed to VALIDATION_FAILED. The core will then

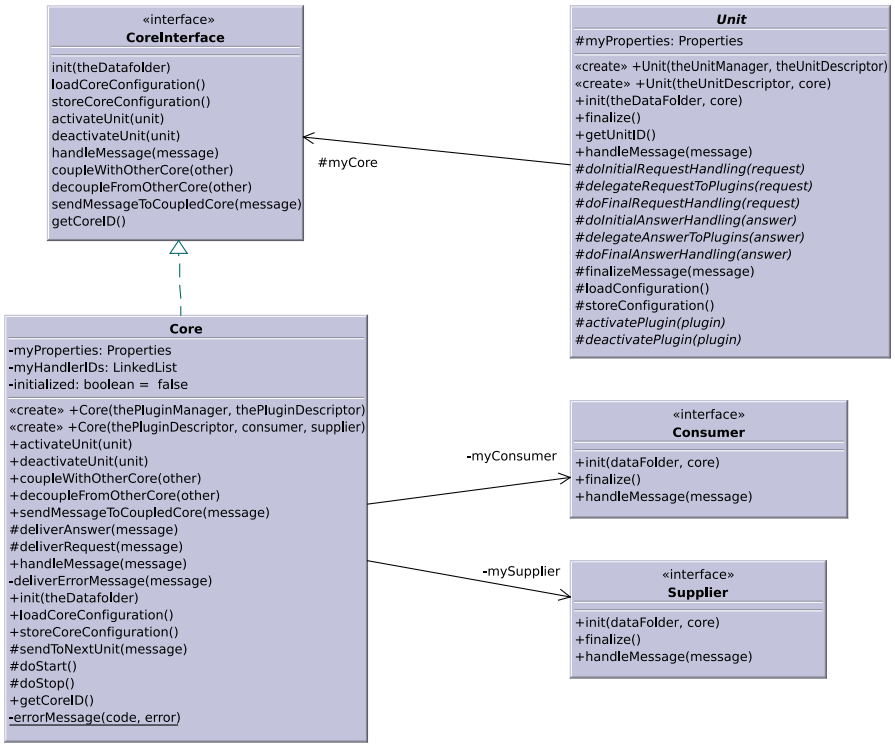

Figure 6.3: UML-Class-Diagram for the core.

evaluate the MessageHistory and generate an error message from the provided information. This error message is stored in the same MessageContext instance, as the original message, and the status is changed to VALIDATION_ERROR_MESSAGE. A message with this type does not require further checks (it has been created by the system) and will directly be delivered to the sender of the original message.

MessageHistory

A MessageHistory is a collection of events that occurred during the life of a MessageContext. The first history entry is the creation of the message context. Further entries will follow during the validation process. A history entry has a type and related phrase. The type of the entry determines what happened to the MessageContext and the phrase determines which unit is related to the event (It will usually be the unique ID of the unit or plugin which triggered the event). The existing entry types are:

- CHECKED: appended to the history whenever the MessageContext passes a check
- CHECKFAILED: appended whenever the MessageContext fails a check
- RECEIVED: appended whenever a MessageContext is Received from a Consumer unit
- SEND: appended whenever a MessageContext is delivered via a Supplier unit
- MODIFIED: appended whenever the message of a MessageContext has been modified
- REJECTED: appended whenever a MessageContext is rejected by the core
- ACCEPTED: appended when the MessageContext has passed all tests and is finally accepted by the core

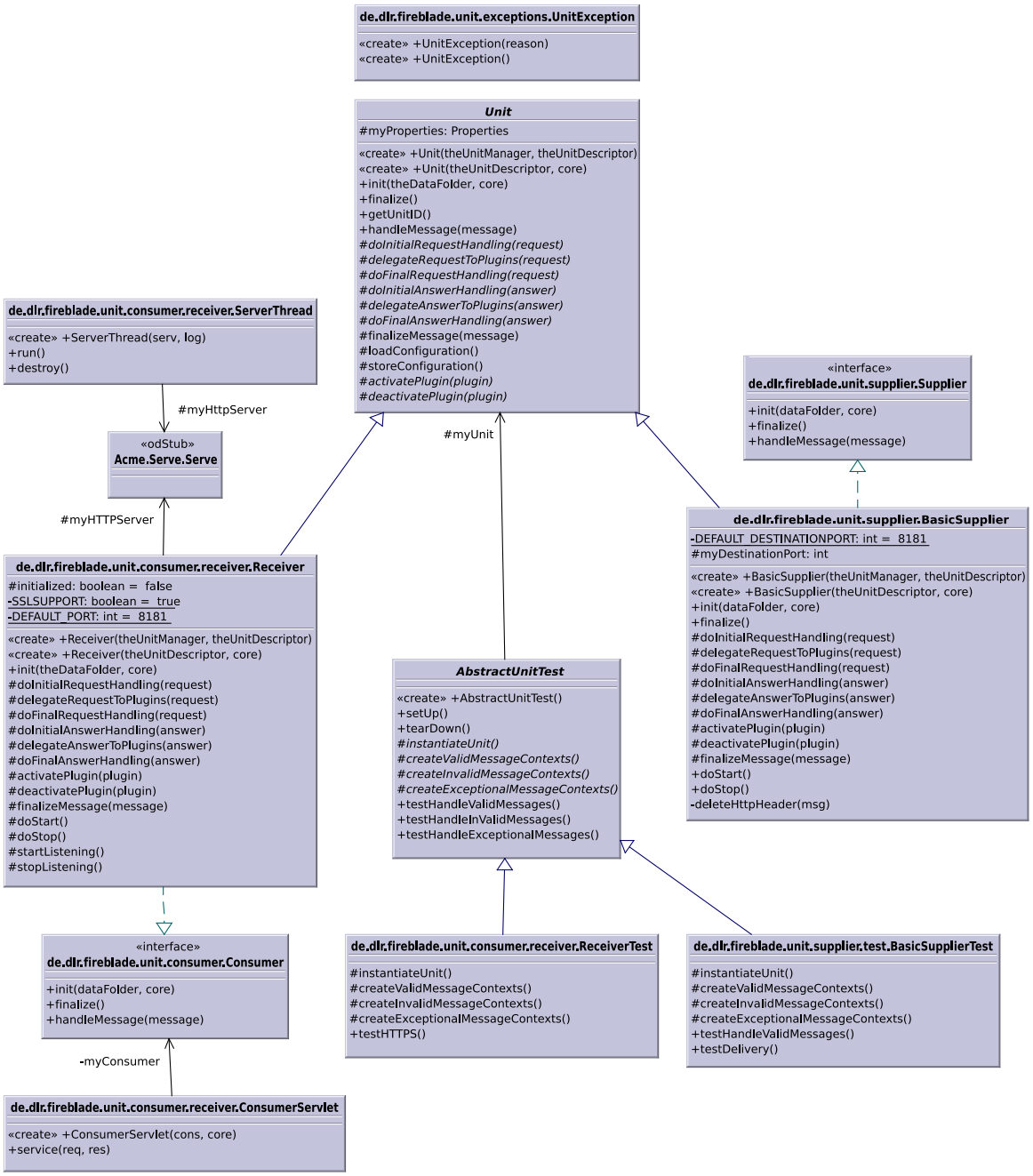

Figure 6.4: UML-Class-Diagram for Units.

- CONTINUED: appended whenever a MessageContext is forwarded to another unit for further checks
- CREATED: appended when the MessageContext is newly created
- ANSWERED: appended when the MessageContext has been delivered by the supplier and the supplier received an answer
- NOTYPE: An Entries of this Type is used whenever no other type fits

6.4.2 Core

The core is the main part of the ALG. It has knowledge about the deployed units and the order in which they should handle messages. It defines three JPF extension points. One for the consumer, one for the supplier and one for all other handler units. The cores main task is, to evaluate MessageContexts and to pass them to the units in the right order. Note that all units are JPF Plugins, as well as the core itself. The core uses the JPF plugin mechanisms to bring up units when they are needed. The core itself is started with a special boot class (*de.dlr.fireblade.Boot*). This class is in large parts the example Boot class from the JPF project page [17]. A java properties file *boot.properties* contains the JPF and the logging configuration. The boot class creates a JPF PluginManager using this configuration. It then instantiates the core and associates it with the PluginManager. This step makes the core aware of all deployed units. Figure 6.3 shows the relations between the core and units.

6.4.3 Consumer

A consumer is the part of the ALG, which accepts incoming messages. These messages may be accepted by many different transport protocols. The Consumer which has been developed for this thesis, accepts messages over HTTP over Secure Socket Layer (HTTPS). It consists out of several classes.

The actual unit is *de.dlr.fireblade.unit.consumer.receiver.Receiver*, a JPF plugin (like every unit). The core discovers and instantiates it as its consumer using the JPF PluginManager. When initialized, it starts an HTTP or HTTPS server. All connections which are made to that server a handled by *de.dlr.fireblade.unit.consumer.receiver.ConsumerServlet*, which is a Java servlet [20] implementing the *service()* method. When this servlet receives a message with HTTP POST, it tries to create a MessageContext from that message and forwards that context to the receiver class. The receiver class will then forward the MessageContext to the core (See figure 6.5).

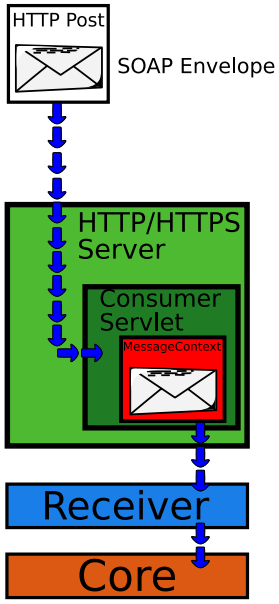

Figure 6.5: Path of an incoming SOAP message

This consumer does not provide many features yet. It can only be seen as a reference implementation on how to write a consumer. The supported transport protocols HTTP/HTTPS will probably be the most common transport protocols for SOAP messages in Grids anyway. Like every Unit, the consumer *de.dlr.fireblade.unit.consumer.Receiver* is theoretically extensible with JPF by. subclassing

de.dlr.fireblade.unit.plugin.Plugin and writing a manifest file which associates the plugin with the an extension point of the consumer (see [17]). But this consumer does neither define any extension points nor does it implement the methods which define when and where the plugins are used. See figure 6.4 for details about the consumers class hierarchy.

6.4.4 Supplier

The supplier is the message delivering part of the ALG. It forwards the Grid service requests to the appropriate Grid node. As already stated in 6.4.3 *Consumer*, it is possible to deliver these messages with different protocols. This supplier does only support HTTP and HTTPS for message delivery and consists of a single Java class (*de.dlr.fireblade.unit.consumer.supplier.BasicSupplier*).

The BasicSupplier is not able to determine the appropriate resource to which it should forward the SOAP message. Instead it forwards all SOAP messages to a single preconfigured resource. This is enough for really small Grids but would not be sufficient in larger ones (because every node would need its own ALG). Even though the BasicSupplier is limited like that, it provides a reference implementation for other suppliers (which would include all the necessary features). See figure 6.4 for details about the suppliers class hierarchy.

6.5 Tests

During the development process, software tests played a major role. In order to guarantee working software pieces, a Test-First approach has been chosen (which means that the TestCases are defined before the software is developed). Due to the plugin architecture and the use of JPF this posed a slight problem. JPF plugins have to communicate with a plugin manager, which is created upon initialisation of JPF. In order to test the different units and plugins of the ALG with standard JUnit TestCases [18], it would have been necessary to initialise the complete JPF for each TestCase. This problem has been solved by the utilization of JMock [16] as a testing library. JMock extends JUnit with *mock objects*, which are placeholders for unimplemented interfaces. It is possible to specify expectations on the number of method calls to those mock objects, which makes it possible to test the complete interaction between the tested and the unimplemented parts. For instance it was possible to test the core, which required a consumer and a supplier, even when consumer ans supplier had not been implemented. It was also possible to test the units (consumer and supplier) without instantiating a core (it was replaced by a mock object in the TestCases), so that failures in the core would not cause the TestCases for the supplier or the consumer to fail. The JPF could be decoupled from the TestCases as well by using mock objects for the plugin manager instead of starting up the complete JPF. JMock allowed a strict division between the different parts of the ALG during testing. This made failure discovery and debugging significantly easier.

When all parts had been implemented and succeeded in all JMock TestCases, the software has been tested in its whole. Therefore the setup shown in figure 6.6 has been used. The ALG has been installed on a PC running Linux and an iptables firewall. Machine A and machine B where not able to reach each other directly, but both could reach the ALG. When requests have been created from machine A and send to the ALG, machine B received those requests and answered them to the ALG. The ALG then forwarded the response back to machine A. This test scenario shows, that the developed software is actually able to fulfill its task of enabling communication of resources which are divided by a firewall. The way the message took through the system can be seen in the created log file:

```
2005-06-30 17:32:17,580 [main] INFO  de.dlr.fireblade.Boot logging system initialized
2005-06-30 17:32:17,892 [main] INFO  de.dlr.fireblade.Boot integrity check done: errors - 0, warnings - 0
2005-06-30 17:32:18,586 [main] INFO  de.dlr.fireblade.unit.consumer.receiver.Receiver Starting ACME HTTPS Server
2005-06-30 17:32:18,587 [main] INFO  de.dlr.fireblade.unit.consumer.receiver.Receiver configuring ACME for SSL support
2005-06-30 17:32:18,636 [Thread-1] INFO  de.dlr.fireblade.unit.consumer.receiver.Receiver ACME HTTPS server is running.
2005-06-30 17:32:18,689 [main] INFO  de.dlr.fireblade.unit.supplier.BasicSupplier Activating Unit
2005-06-30 17:32:32,693 [Request handler] INFO  de.dlr.fireblade.unit.consumer.receiver.Receiver handling message defaultCore/1
2005-06-30 17:32:32,693 [Request handler] INFO  de.dlr.fireblade.unit.consumer.receiver.Receiver finalizing message defaultCore/1
2005-06-30 17:32:32,694 [Request handler] INFO  de.dlr.fireblade.core.Core handling message defaultCore/1
2005-06-30 17:32:32,694 [Request handler] INFO  de.dlr.fireblade.unit.supplier.BasicSupplier handling message defaultCore/1
2005-06-30 17:32:33,006 [Request handler] INFO  de.dlr.fireblade.unit.supplier.BasicSupplier transferring SOAP message with HTTP GET
2005-06-30 17:32:33,101 [Request handler] INFO  de.dlr.fireblade.unit.supplier.BasicSupplier message defaultCore/1 delivered.
2005-06-30 17:32:33,159 [Request handler] INFO  de.dlr.fireblade.unit.supplier.BasicSupplier message defaultCore/1 has been answered
2005-06-30 17:32:33,160 [Request handler] INFO  de.dlr.fireblade.core.Core handling message defaultCore/1
2005-06-30 17:32:33,160 [Request handler] INFO  de.dlr.fireblade.unit.consumer.receiver.Receiver handling message defaultCore/1
```

```
2005-06-30 17:32:33,446 [Request handler] INFO  de.dlr.fireblade.unit.consumer.receiver.Receiver handling message defaultCore/2
2005-06-30 17:32:33,447 [Request handler] INFO  de.dlr.fireblade.unit.consumer.receiver.Receiver finalizing message defaultCore/2
2005-06-30 17:32:33,447 [Request handler] INFO  de.dlr.fireblade.core.Core handling message defaultCore/2
2005-06-30 17:32:33,447 [Request handler] INFO  de.dlr.fireblade.unit.supplier.BasicSupplier handling message defaultCore/2
2005-06-30 17:32:33,458 [Request handler] INFO  de.dlr.fireblade.unit.supplier.BasicSupplier Transferring SOAP message with HTTP GET
2005-06-30 17:32:33,531 [Request handler] INFO  de.dlr.fireblade.unit.supplier.BasicSupplier message defaultCore/2 delivered.
2005-06-30 17:32:33,582 [Request handler] INFO  de.dlr.fireblade.unit.supplier.BasicSupplier message defaultCore/2 has been answered
2005-06-30 17:32:33,582 [Request handler] INFO  de.dlr.fireblade.core.Core handling message defaultCore/2
2005-06-30 17:32:33,584 [Request handler] INFO  de.dlr.fireblade.unit.consumer.receiver.Receiver handling message defaultCore/2
2005-06-30 17:33:38,481 [Request handler] INFO  de.dlr.fireblade.unit.consumer.receiver.Receiver handling message defaultCore/3
2005-06-30 17:33:38,482 [Request handler] INFO  de.dlr.fireblade.unit.consumer.receiver.Receiver finalizing message defaultCore/3
2005-06-30 17:33:38,482 [Request handler] INFO  de.dlr.fireblade.core.Core handling message defaultCore/3
2005-06-30 17:33:38,482 [Request handler] INFO  de.dlr.fireblade.unit.supplier.BasicSupplier handling message defaultCore/3
2005-06-30 17:33:38,510 [Request handler] INFO  de.dlr.fireblade.unit.supplier.BasicSupplier transferring SOAP message with HTTP POST
2005-06-30 17:33:38,603 [Request handler] INFO  de.dlr.fireblade.unit.supplier.BasicSupplier message defaultCore/3 delivered.
2005-06-30 17:33:38,611 [Request handler] INFO  de.dlr.fireblade.unit.supplier.BasicSupplier message defaultCore/3 has been answered
2005-06-30 17:33:38,611 [Request handler] INFO  de.dlr.fireblade.core.Core handling message defaultCore/3
2005-06-30 17:33:38,611 [Request handler] INFO  de.dlr.fireblade.unit.consumer.receiver.Receiver handling message defaultCore/3
2005-06-30 17:34:43,760 [Request handler] INFO  de.dlr.fireblade.unit.consumer.receiver.Receiver handling message defaultCore/4
2005-06-30 17:34:43,761 [Request handler] INFO  de.dlr.fireblade.unit.consumer.receiver.Receiver finalizing message defaultCore/4
2005-06-30 17:34:43,761 [Request handler] INFO  de.dlr.fireblade.core.Core handling message defaultCore/4
2005-06-30 17:34:43,761 [Request handler] INFO  de.dlr.fireblade.unit.supplier.BasicSupplier handling message defaultCore/4
2005-06-30 17:34:43,773 [Request handler] INFO  de.dlr.fireblade.unit.supplier.BasicSupplier transferring SOAP message with HTTP POST
2005-06-30 17:34:43,828 [Request handler] INFO  de.dlr.fireblade.unit.supplier.BasicSupplier message defaultCore/4 delivered.
2005-06-30 17:34:43,872 [Request handler] INFO  de.dlr.fireblade.unit.supplier.BasicSupplier message defaultCore/4 has been answered
2005-06-30 17:34:43,872 [Request handler] INFO  de.dlr.fireblade.core.Core handling message defaultCore/4
2005-06-30 17:34:43,872 [Request handler] INFO  de.dlr.fireblade.unit.consumer.receiver.Receiver handling message defaultCore/4
2005-06-30 17:35:14,776 [Request handler] INFO  de.dlr.fireblade.unit.consumer.receiver.Receiver handling message defaultCore/5
2005-06-30 17:35:14,777 [Request handler] INFO  de.dlr.fireblade.unit.consumer.receiver.Receiver finalizing message defaultCore/5
2005-06-30 17:35:14,777 [Request handler] INFO  de.dlr.fireblade.core.Core handling message defaultCore/5
2005-06-30 17:35:14,777 [Request handler] INFO  de.dlr.fireblade.unit.supplier.BasicSupplier handling message defaultCore/5
2005-06-30 17:35:14,815 [Request handler] INFO  de.dlr.fireblade.unit.supplier.BasicSupplier transferring SOAP message with HTTP POST
2005-06-30 17:35:14,871 [Request handler] INFO  de.dlr.fireblade.unit.supplier.BasicSupplier message defaultCore/5 delivered.
2005-06-30 17:35:14,917 [Request handler] INFO  de.dlr.fireblade.unit.supplier.BasicSupplier message defaultCore/5 has been answered
2005-06-30 17:35:14,918 [Request handler] INFO  de.dlr.fireblade.core.Core handling message defaultCore/5
2005-06-30 17:35:14,918 [Request handler] INFO  de.dlr.fireblade.unit.consumer.receiver.Receiver handling message defaultCore/5
```

As you can see, there have been 5 different messages from machine A to machine B (with message ids *defaultCore/1* to *defaultCore/5*). Some of them have been HTTP Get requests and some have been HTTP Post requests. All messages took the intended path (machine A → consumer → core → supplier → machine B → supplier → core → consumer → machine B). Compare with this snippet of the log file:

```
2005-06-30 17:32:32,693 [Request handler] INFO  de.dlr.fireblade.unit.consumer.receiver.Receiver handling message defaultCore/1
2005-06-30 17:32:32,693 [Request handler] INFO  de.dlr.fireblade.unit.consumer.receiver.Receiver finalizing message defaultCore/1
2005-06-30 17:32:32,694 [Request handler] INFO  de.dlr.fireblade.core.Core handling message defaultCore/1
2005-06-30 17:32:32,694 [Request handler] INFO  de.dlr.fireblade.unit.supplier.BasicSupplier handling message defaultCore/1
2005-06-30 17:32:33,006 [Request handler] INFO  de.dlr.fireblade.unit.supplier.BasicSupplier transferring SOAP message with HTTP GET
2005-06-30 17:32:33,101 [Request handler] INFO  de.dlr.fireblade.unit.supplier.BasicSupplier message defaultCore/1 delivered.
2005-06-30 17:32:33,159 [Request handler] INFO  de.dlr.fireblade.unit.supplier.BasicSupplier message defaultCore/1 has been answered
2005-06-30 17:32:33,160 [Request handler] INFO  de.dlr.fireblade.core.Core handling message defaultCore/1
2005-06-30 17:32:33,160 [Request handler] INFO  de.dlr.fireblade.unit.consumer.receiver.Receiver handling message defaultCore/1
```

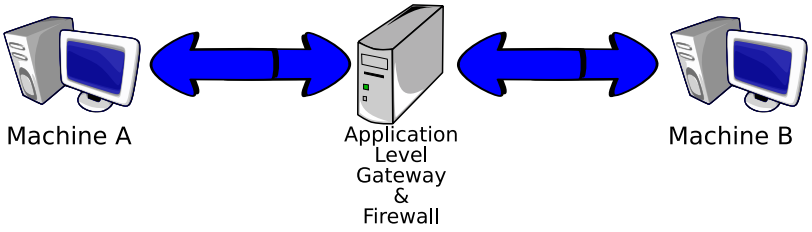

Figure 6.6: Test scenario for the ALG

6.6 Installation and Configuration

In order to use the ALG it is of course necessary to install and configure it correctly. The installation from the sources is really simple. At first the binaries are needed. If you possess the source code, you can build those binaries with the command

```
~/path/to/source# ant all clean
```

this will create a zip file in the current folder containing the binaries. Simply unzip that file to the folder where you want the ALG to reside (on a machine that is placed in the DMZ).

```
/path/to/source# mv fireblade-<someversionstring>.zip /path/to/my/alg/
cd /path/to/my/alg
unzip fireblade-<someversionstring>.zip
```

Actually the software is now able to run, but not very well configured. The core and all units have their own configuration file. In order to change the configuration of the core, create a file called *core.properties* in the *data* folder of your installation (the place where you extracted the binaries to). Put the properties that you would like to be set for the core in that file. See table 6.2 for possible properties.

```
cd /path/to/my/alg/data
vi core.properties
      // configure the core
```

Furthermore you have to create the configuration files for consumer and supplier. See table 6.3 and 6.4 for the properties that are available to configure the developed consumer and supplier.

```
cd /path/to/my/alg/data
mkdir <plugin-id of the consumer>
cd <plugin-id of the consumer>
vi core.properties
      // configure the consumer
cd ..
mkdir <plugin-id of the supplier>
cd <plugin-id of the supplier>
vi core.properties
      // configure the supplier
```

Now that you have configured core, consumer and supplier, you have to deploy keystore files for both of them. Those keystore files can be crated with the *keytool* command. See the man page of that command (*man keytool*) for details about this. because the consumer does not support client authorization yet, it is only needed to setup a certificate for the consumer and get it signed by a Certificate Authority (CA). Since the supplier needs to authenticate the Grid resource which it should talk to, it is needed to import that resources certificate into the keystore of the supplier.

The machine where the ALG is installed must additionally be configured in a way that enables access to the ALG from the internet and enables the ALG to access the local network. This can for instance be accomplished by opening the listening port of the consumer and by employing a second network interface. The open port enables machines from the internet to access the ALG and the second network interface could be used in order to access local machines.

Property	**Function**	**Default value**
fireblade.core.id	identification string for the core	defaultCore
fireblade.unit.consumer	plugin id of the consumer unit that should be used	de.dlr.fireblade.consumer.basic
fireblade.unit.supplier	plugin id of the supplier unit that should be used	de.dlr.fireblade.supplier.basic
fireblade.unit.handlers	list of plugin ids of the handler units that should be used in the given order	*empty*

Table 6.2: Properties for configuration of the core

6.7 Limitations

Due to the short development time of about two months, the realization has some limitations compared to the concept. It lacks features like load balancing and core coupling as well as content filtering

Property	Function	Default value
fireblade.consumer.port	port number where the consumer should listen for incoming requests	8181
fireblade.consumer.keystorefile	name of the keystore file where the SSL certificates are stored	ServerKeyStore
fireblade.consumer.keystorepass	password for that keystorefile	getaccess
fireblade.consumer.keystoretype	type of the keystorefile (see *man keytool*)	JKS

Table 6.3: Properties for configuration of the consumer

Property	Function	Default value
fireblade.supplier.destination.host	host name of the target resource which should receive the requests	localhost
fireblade.supplier.destination.port	port number where the target resource is listening for requests	8181
fireblade.supplier.keystore.file	name of the keystore file where the SSL certificates are stored	clientKeyStore
fireblade.supplier.keystore.password	password for that keystorefile	getaccess
fireblade.supplier.keystore.type	type of the keystorefile (see *man keytool*)	JKS
fireblade.supplier.secure.protocol	SSL encryption protocol	TLS
fireblade.supplier.secure.algorithm	SSL encryption algorithm	SunX509

Table 6.4: Properties for configuration of the supplier

and validation. By now it does not support any Grid communication but plain SOAP messages. Neither the Consumer, nor the Supplier support HTTPS client authentication. The harshest limitation is given by the supplier. It does not support multiple resources yet (see 6.4.4 *Supplier*). This renders the realization irrelevant for practical use. Even when the supplier would be changed or extended in order to enable multiple resources per ALG, it would probably pose a massive bottleneck because no efforts have been taken in order to optimize it. The lacking load balancing mechanisms add to this limitation. Furthermore the setup configuration is rather complicated. There are no tools which help an Administrator with this task by now. It can be seen as a framework with a reference implementation of the main features which can be the basis for further work, but it is far from a stable and fully-featured product.

The actual overhead which is created by this solution is hard to estimate. It largely depends on the number of units which check the SOAP messages. In theory it is possible to deploy a infinite number of units. This means that the overhead can become infinitly large. Additionaly it depends on the transport protocol used by supplier and consumer. A supplier which works with HTTP would be faster than one working with HTTPS or one working via email. None of the developed classes have been optimized for speed (latency and throughput) yet. Since Grid applications are usually *emberassing parallel* (or latency tolerant), A optimization for throughput might be mormeaningful than one optimizing latency. It can be assumed, that those applications which require high throughput or low latenncy would communicate over specialized protocols. The units which add support for those protocols should then be optimized for the according as well.

Chapter 7

Summary

The problem that today's Grids have with communication over firewall protected network borders has been described in this thesis, which also presented and compared several concepts for solving this *firewall issue*. The chosen concept, of utilizing an Application Level Gateway which creates proxies for Grid services has been implemented in large parts. This has been done with respect to the large number of possibly required protocols, what led to a highly extensible plugin architecture. Even though this implementation is not yet capable of solving the problem completely, it already demonstrates some advantages over other concepts, like its transparency.

By extensively testing the ALG during development, it was possible to create software which enabled communication between two machines which where separated by a firewall.

Not only the concept, but also the implementation have been discussed. While the developed architecture offers a transparent and secure solution for the issue, it might become a problem itself, when it comes to performance and especially to latency tolerance. Additionally the developed ALG supports has significantly inferior features to those, the other presented concepts might provide. However these features will most likely be implemented soon. Recently another bachelor thesis which deals with this issue has been started at SIStEC. During that thesis, Thijs Metsch will implement the missing content validation which has been identified as a major security advantage of the ALG concept over the other concepts. Additionally the author of this thesis will continue to work on this implementation in the future, and it will play a role in the future research of the FI-RG at the GGF.

List of Tables

List of Figures

Nomenclature

ALG Application Level Gateway

BSI German Federal Office for Information Security

CA Certificate Authority

CORBA Common Object Request Broker Architecture

DLR German Aerospace Center

DMZ demilitarized zone

FI-RG Firewall-Issue research group

GGF Global Grid Forum

GRAM Grid Resource Allocation and Management

GridFTP Grid File Transfer Protocol

HTTP Hyper Text Transfer Protocol

HTTPS HTTP over Secure Socket Layer

IDL Interface Definition Language

IP Internet Protocol

JPF Java Plugin Framework

OGSA Open Grid Service Architecture

OGSA-WG OGSA Working Group

OGSI Open Grid Service Infrastructure

OGSI-WG OGSI Working Group

SAML Security Assertion Markup Language

SCAI Fraunhofer-Institute for Algorithms and Scientific Computing

SISTEC Simulations- and Softwaretechnik

SOAP Simple Object Access Protocol

SSH Secure Shell

SSL Secure Socket Layer

TCP Transmission Control Protocol

UDP User Datagram Protocol

URI Uniform Resource Identifier

VO virtual organizations

VOMS Virtual Organization Membership Service

VPN Virtual Private Network

WS-DBC Xtradyne Web Service Domain Boundary Controller

WSDL Web Service Definition Language

WSRF Web Service Resource Framework

XML eXtensible Markup Language

XML-RPC eXtensible Markup Language - Remote Procedure Call

Bibliography

[1] Axis - project homepage. http://ws.apache.org/axis/.

[2] Gerald Brose. Securing web services with soap security proxies. http://www.xtradyne.com/documents/whitepapers/Xtradyne-WebServices_Security_Proxies.pdf.

[3] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web services description language (wsdl) 1.1, March 2001. http://www.w3.org/TR/wsdl.

[4] Karl Czajkowski, Don Ferguson, Ian Foster, Jeffrey Frey, Steve Graham, Tom Maguire, David Snelling, and Steve Tuecke. From open grid services infrastructure to wsresource framework: Refactoring & evolution, version 1.1, March 2004. http://www.globus.org/wsrf/specs/ogsi_to_wsrf_1.0.pdf.

[5] Karl Czajkowski, Donald F. Ferguson, Ian Foster, Jeffrey Frey, Steve Graham, Igor Sedukhin, David Snelling, Steve Tuecke, and William Vambenepe. The ws-resource framework, version 1.0, March 2004. http://www.oasis-open.org/committees/download.php/6796/ws-wsrf.pdf.

[6] David De Roure, Mark A. Baker, Nicholas R. Jennings, and Nigel R. Shadbolt. *Grid Computing - Making the Global Infrastructure a Reality*, chapter 3, pages 65 – 100. John Wiley & Sons, Ltd, 2003.

[7] Ian Foster, Carl Kesselmann, Jeffrey M. Nick, and Steven Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. Presented at GGF4, February 2002. http://www.globus.org/alliance/publications/papers/ogsa.pdf.

[8] Ian Foster, Carl Kesselmann, and Steven Tuecke. The anatomy of the grid. *International J. Supercomputer Applications*, 15(3), 2001.

[9] Ian Foster, Hiro Kishimoto, Andreas Savva, Dave Berry, Abdeslem Djaoui, Andrew Grimshaw, Bill Horn, Fred Maciel, Frank Siebenlist, Ravi Subramaniam, Jem Treadwell, and Jeffrin J. Von Reich. Open grid services architecture (ogsa) version 1.0, January 2005. http://www.gridforum.org/documents/GWD-I-E/GFD-I.030.pdf.

[10] Google web api. http://www.google.com/apis/.

[11] Sven Graupner and Carsten Reimann. Globus grid and firewalls: Issues and solutions in a utility data center environment, October 2002.

[12] Roland Gude and Thijs Metsch. Fireblade catalogue of requirements. DLR-SISTEC internal document, April 2005.

[13] Roland Gude and Thijs Metsch. Fireblade concept and design manual. DLR-SISTEC working document, June 2005.

[14] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, and Henrik Frystyk Nielsen. Soap version 1.2, June 2003. http://www.w3.org/TR/soap12.

[15] isecure web services gateway - data sheet.
http://www.multinetsecurity.com/images/pdf/isecureweb_services_data_sheet.pdf.

[16] jmock - a lightweight mock object library for java. http://www.jmock.org/.

[17] Java plugin framework - project homepage. http://jpf.sourceforge.net.

[18] Junit - testing resources for extreme programming. http://www.junit.org/.

[19] Thijs Metsch. Entwicklung von grid services für ein computational grid. seminar paper, June 2005.

[20] Sun developer network - java servlet technology. http://java.sun.com/products/servlet/.

[21] Yury Strashnoy. The need for web application security.
http://www.multinetsecurity.com/images/pdf/need_for_web_app_security.pdf.

[22] Steven Tuecke, Karl Czajkowski, Ian Foster, Jeffry Frey, Steve Graham, Carl Kesselman, Tom Maquire, Thomas Sandholm, David Snelling, and Peter Vanderbilt. Open grid services infrastructure (ogsi) version 1.0, June 2003. http://www.ggf.org/ogsi-wg.

[23] Protecting web services with the xml/soap security gateway.
http://www.xtradyne.com/documents/whitepapers/Xtradyne-WS-DBC-WhitePaper.pdf.

[24] Web services domain boundary controller 2.1 - product data sheet.
http://www.xtradyne.com/documents/datasheets/Xtradyne_WS-DBC_ProductDataSheet.pdf.

[25] Olaf Zimmermann, Mark Tomilson, and Stefan Peuser. *Perspectives on Web Services - Applying SOAP, WSDL and UDDI to Real-World Projects*. Springer Verlag Berlin Heidelberg New York, 2003.