Diplomarbeit

Content-validation of Messages and Policy assurances for a Security-Proxy supporting Grid Services

Thijs Metsch
DLR Simulations- und Softwaretechnik, BA Mannheim

DLR

Content-validation of Messages and Policy assurances for a Security-Proxy supporting Grid Services

Implementation and design of an Application Level Gateway

Diplomarbeit

für die Prüfung zum
Diplom-Ingenieur (Berufsakademie)

der Fachrichtung Informationstechnik
an der Berufsakademie Mannheim

von

Thijs Metsch

September 2005

Bearbeitungszeitraum	3 Monate
Kurs	TIT02BNS
Ausbildungsfirma	Deutsches Zentrum für Luft- und Raumfahrt - DLR e.V. Simulations- und Softwaretechnik Linder Höhe, 51147 Köln
Gutachter der Ausbildungsfirma	Dipl.-Math Andreas Schreiber
Gutachter der Studienakademie	Prof. Dr. Rainer Colgen

Abstract

Title	Content-validation of Messages and Policy assurances for a Security-Proxy supporting Grid Services
Name	Thijs Metsch
Matr.-nr.	197534
Firm	Deutsches Zentrum für Luft- und Raumfahrt
	Division for Simulation- and Softwaretechnology

Today Grid computing is an important technology that allows scientists and engineers to solve large and complex problems, to work in complex and heterogeneous environments, and to cooperate in various new ways. In Grid environments integrate distributed computing resources, networks, scientific instruments, data archives and databases, and visualization environments. The virtualization of these resources results in resource environments allowing the dynamic generation of Virtual Organizations (VO) to increase the productivity and quality of scientific work.

This work describes a security concept for securing Grid Services in a firewalled environment. The main aspect of this concept is a security gateway which performs content based checks on incoming Grid requests. This is an application level gateway and it checks SOAP messages of Grid Service requests and decides on the application level (OSI level 7) whether the message should pass the gateway or be blocked.

In combination with packet filtering, provided by usual firewall solutions, and encrypted data transfer methods, this allows a shared secured use of Grid resources, separated by security gateways. This can be accomplished without changing the respective Grid middleware and without increasing security risks to an unacceptable level (e. g., by opening network ports).

The work on this topic led to the conclusion that with an appropriate concept, modern services based distributed environments can be secured. This concept includes the use of firewalls and security proxies.

Kurzfassung

Titel	Content-validation of Messages and Policy assurances for a Security-Proxy supporting Grid Services
Name	Thijs Metsch
Matr.-nr.	197534
Firma	Deutsches Zentrum für Luft- und Raumfahrt
	Division for Simulation- and Softwaretechnology

Ingenieure und Wissenschafter können mittels Grid computing komplexe Probleme lösen. Ein Grid stellt dafür eine heterogene, aber mit vielen Funktionen ausgestattete Umgebung zur Verfügung. Verschiedenste Ressourcen werden in einem Grid integriert. Dies können Rechen-Ressourcen, Datenbanken oder Ressourcen anderer Art sein. Die Virtualisierung dieser Ressourcen führt zu der Bildung von Virtuellen Organisationen (VOs).
Diese Arbeit beschreibt ein Sicherheitskonzept für Grid Services in Virtuellen Organisationen. Firewalls trennen die reellen Organisationen und verhindern die reibungslose Zusammenarbeit innerhalb der VOs. Das Hauptaugenmerk dieser Arbeit liegt auf der Entwicklung eines Application Level Gateways. Dieses Gateway verifiziert den Inhalt von Transaktionen zwischen einem Client und einem Server. Dies geschieht auf Applikations-Schicht (OSI Level7). Das Gateway entscheidet ob eine Anfrage durchgelassen wird oder abgelehnt wird.
In Kombination mit Packet Filtern (Firewall) und verschlüsselten Verbindungen bildet das Gateway eine Methode um Grid Services abzusichern. So können die verteilten Resourcen innerhalb einer VO auf einem sicheren Weg genutzt werden. Vorhandene Grid middleware Lösungen müssen bei diesem Konzept nicht modifiziert werden.
Das Ergebnis dieser Arbeit zeigt, dass es mit einem passenden Sicherheitskonzept möglich ist verteilte Anwendungen innerhalb eines Grid sicher laufen zu lassen. Dieses Konzept umschliet sowohl Packet Filter als auch den Gateway.

Disclaimer

I hereby declare that I have written the bachelor thesis on my own, and used no other than the stated sources and aids.

Cologne, September 2005

Contents

1. Introduction

Today Grid computing [1] is an important technology that allows scientists and engineers to solve large and complex problems, to work in complex and heterogeneous environments, and to cooperate in various new ways. Grid environments integrate distributed computing resources, networks, scientific instruments, data archives and databases, and visualization environments. The virtualization of these resources results in resource environments allowing the dynamic generation of Virtual Organizations (VOs) [2] to increase the productivity and quality of scientific work.
In commerce Grid computing becomes even more important. Large computer vendors bring the idea of "business of demand" to the market. This includes the flexible use of distributed resources to cut IT costs and react to changes of the market.
Current Grid infrastructure solutions are based on Web Services. The Open Grid Service Architecture (OGSA) [3] and the Web Service Resource Framework (WSRF) [4] try to standardize this approach. These standards are still under development, but nevertheless Grid Computing is already being used in various projects and applications.
When creating VOs, not only access has to be granted to each other's resources, but means of communication have got to be established between them. In modern Grids this will mostly be communication for the invocation of services. But non-blocking I/O traffic is also part of this scenario. File transfers would be an example for this.
In these Grid-based resource environments security is an important topic, because all integrated resources belong to different owners and administrative domains, with different usage and security rules. On the other hand, users from the industry have high security requirements. Security mechanisms are already included in various levels and components of existing Grid infrastructures, which provide authentication and authorization of users, secure transfer of data, and information.
Most companies (and so their sites) use modern firewall technologies to protect their data and so their intellectual property. In most cases this includes several firewalls and security zones. These zones match different levels of security. Communication, which is needed by means of Grid computing becomes nearly impossible.
This bachelor thesis describes a security concept to realize these communications in a secure manner. Part of this concept will be a security proxy (or Application Level Gateway (ALG)) to validate traffic. The design and implementation of this proxy are part of this work.

1.1. Security issues for Grid based applications

Following use case shows that current security activities are not sufficient to create VOs. Several firewall issues arise during the setup of the system described below.
The workflow management system TENT (see figure 1 for a screenshot) has been developed at the German Aerospace Center over the last years. It allows engineers to easily setup and maintain workflows. Workflows are components coupled together to form a process chain. Components can be numerical or functional units within a work flow, e. g. computational fluid dynamic (CFD) codes, graphical editors for visualization, or pre-/post-processors.

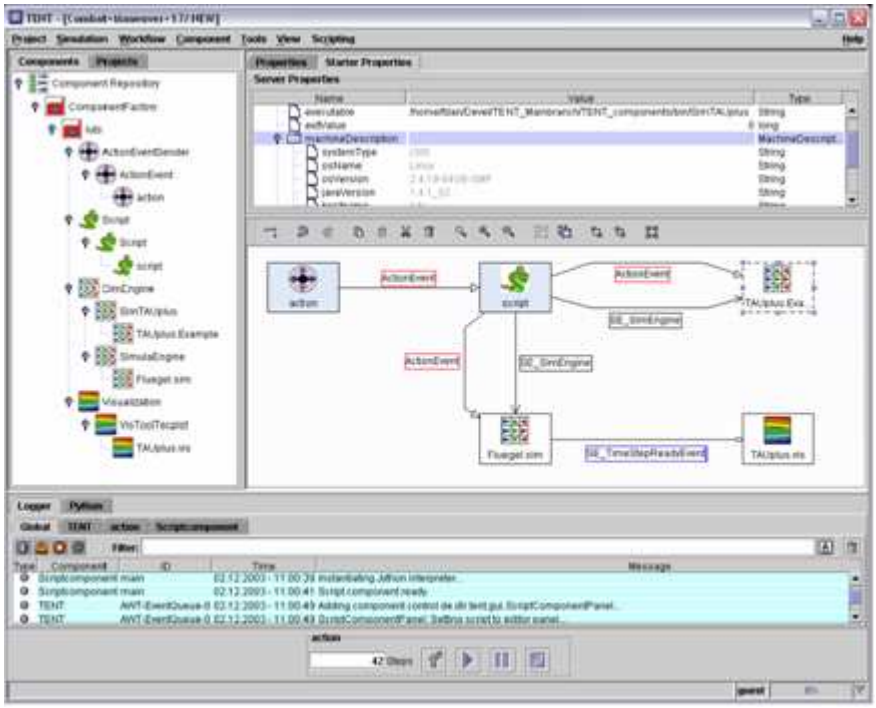

Figure 1: TENT user interface.

This process chain can be used to perform fluid dynamics, structural mechanics, and thermodynamic computations. Components are the smallest elements in a workflow and can be localized on distributed resources. Coupling of resources can be achieved by means of Grid computing. Computational resources can be placed on different remote hosts. By creating Virtual Organizations (VOs) it is possible to use all these resources as one. The following functionalities are necessary to use TENT within Grids:

- Access to all resources of a Grid. Authentication and authorization mechanisms need to be provided.
- Data transfers between the resources of a Grid must be possible. Data transfers can either be Reliable File Transfers (RFT) or status messages (MPI based messaging) passed along.

- Execution of CFD codes on Grid resources. Job Managers and their queues should be accessible to the TENT system.
- For all these communications TENT uses service based communication.

The creation of VOs becomes obligatory when applications (and its matching licenses) and resources are located on either sides of a firewall. The creation of VOs can extend beyond the borders of companies. The location of the Grid resources therefore is no longer bound to geographical positions. Figure 2 gives a closer picture of the borders of a company. Firewalls form the borders of the local site. But applications of resources like high performance cluster are not located within the local (and easily accessible) network.

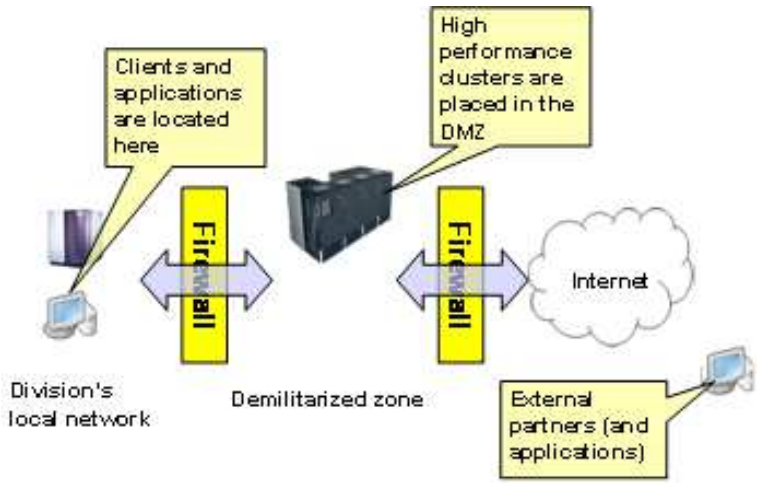

Figure 2: Possible firewall borders.

Due to the fact that most companies and organizations use firewalls following problems arise:

- Several firewalls have to be passed (internally and externally). The administrators of these firewalls are not always directly available.
- Firewalls should be opened for several TCP and UDP ports so Grid middleware solutions can communicate. Some port ranges are unknown during set-up. They will be initialized by the Grid middleware itself. So port ranges have to de defined.
- Data transfers have to be allowed beyond the borders of a local site. This includes the transmission of data packets and status information.
- VPNs have to be initialized at the borders of a site. To increase security the traffic connections should be secured against wire tapping.

Security policies disallow the opening of firewalls in almost every case. Strict control of the incoming and outgoing traffic becomes a major issue. A lot of politics have to be dealt with when establishing connections beyond company's borders.

1.2. Problem specification

As result of this use case following requirements and considerations can be defined:

- The resources in a Grid are in local networks protected by firewalls. The security policy of the location does not allow to establish connections through the firewall.
- The connections through a firewall, protecting the Grid resources, must be tunneled. But connections and request need to be filtered and checked before.
- The security policy of the location needs a strict separation of the Internet and the location's local network.
- No extra steps are necessary from the user's side. The solution must be transparent for the user.

1.3. Possible solutions

Several approaches to the problem described above are layed out in the following. A short overview of the advantages and disadvantages of these approaches are summarized.

1.3.1. SSH Tunnel

One of the easiest possible solutions to tunnel firewalls is the setup of SSH tunnels. The basic idea is to open encrypted connections to a remote host. These tunnel connections originate from a local host and reach to a remote destination. Direct connections then can be established on certain ports from the designated remote to the local host.
This solution has several disadvantages. All ports must be known prior to open enough tunneled connections. This constellation is difficult for an automated setup. A possible security risk is the direct access to the remote host through the tunnel. An attacker has the possibility to use an open connection and the data transferred on it is not verified. A large number of tunnels is needed for anything more than a very simple setup.

1.3.2. Demilitarized Zone (DMZ)

A common solution for the problem described here is the localization of the Grid resources in a demilitarized zone (DMZ), which is protected by a firewall. The DMZ separates the Internet from the local network with firewalls, but hosts placed within it can be accessed from both sides.
However this solution forces companies to expose their resources to possible misuse from the Internet. It would be more secure to place the Grid resources inside

the fully protected local network and to define mechanisms to access these resources securely. Sometimes it is impossible to place certain resources in a DMZ.

1.3.3. Application Level Gateways

Basically the ALG can be described as a system representing a service to the outside and functions as a client to the Inside. The figure 3 describes the setup.

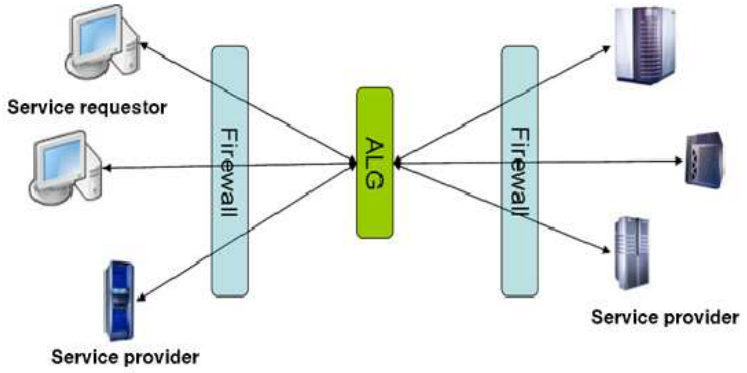

Figure 3: General setup of an ALG based security concept.

Being placed between two firewalls the ALG can be accessed from both sides - either the untrusted (Internet) or the local trusted intranet. Some ports in the firewalls have to be opened. The security holes which are now open must be closed with the help of the ALG. The ALG will take care of all traffic and disallows any direct connections between the Internet and the intranet.
An ALG is placed in the DMZ and reacts to requests from the Internet as if it was the requested resource. Therefore it acts as a client for the requested resource and forwards the requests to that resource. After the ALG received an response it will forward the response 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 communication protocols, an ALG must be highly extendable and flexible.
The Firewall which separates the DMZ from local network may be configured in a way that it allows incoming traffic only from the ALG. This would increase security for the Grid resources significantly. However some administrators will not even allow incoming traffic from a known system. If this is the case, traffic has to be fetched from inside.

1.4. Existing Solution - Xtradyne Web Service Domain Boundary Controller

The Xtradyne Web Service Boundary Controller (WS-DBC) [5] is an ALG-based approach to secure Web services. Web services form the basis of Grid services therefore this approach is an outrider to the ALG described here.

WS-DBC acts as a transparent security proxy for Web services. Simple Object Access Protocol (SOAP) messages are accepted from the WS-DBC which then identifies and authenticates the Client and inspects the content of the SOAP message.
Based on that content inspection and the client authorization WS-DBC allows or denies access to a requested Web service. It therefore adds Security Assertion Markup Language (SAML) assertions to the SOAP message header and forwards that header to the requested Web service. WS-DBC thereby provides end-to-end security for Web services on the application Level.
WS-DBC and the ALG described here 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 the solution described here addresses Grid services.
The main difference is: Grid services often use protocols that are not SOAP based, i.e. GridFTP or GASS. Those protocols can transfer any type of data between two nodes and will use SOAP messages only to initialize a data transfer. WS-DBC is not able to handle such data transfers. Additionally WS-DBC only validates SOAP messages against given schema files. A stricter validation of the content of Messages is needed. Extensions for very specific Services can not be deployed within WS-DBC. WS-DBC does not provide the functionality to change message contents (i.e. for mapping of users). The ALG described here adds this support. Together with existing software like WS-Security it provides a more flexible and stricter approach.

2. Requirements for an Application Level Gateway

The solution used here will be able to act as a complete Grid to any requests from the Internet. It will forward these requests to the real resources. Doing this it will collect the answers as well and deliver them to the original clients.

2.1. Setup

The two major parts of the proposed security infrastructure are the packet filter and the ALG.
The main task of the packet filter is to block connections based on the information provided in the header of TCP/IP packets. Requests having an unknown source are dropped immediately. If more than one ALG is available, the packet filters may perform load balancing. More important is the protection against attacks. Combined with an intrusion detection system (IDS), the packet filter can avoid denial of service (DoS) attacks.
In addition the ALG inspects the traffic on the application level (OSI Level 7). For this it intercepts the traffic between the source and the destination. Several ALGs can be chained if needed. The following tasks have to be accomplished by the ALG:

- Relay all connections in two directions (two-way proxy).
- User level control of access to the Grid.
- Verification of incoming request. Modifications of the requests should also be possible.
- Logging and accounting of all actions.

2.2. Security threats in Networking Environments

When planning security concepts for service based environments, it is helpful to be aware of possible security threats. When designing a concept suitable countermeasures can be included. The following list describes security threats and risks.

Spoofing This security threat refers to a situation where a user acts as if he is somebody else. Man-in-the-middle attacks are typical spoofing actions.

Data manipulation Operations executed on data that is not to be changed form a potential risk. This does not only include the manipulation of files, but also the data transferred in a connection.

Uncontrolled use of resources For economic and security reasons, the use of resources should be limited to those authorized to access them. The uncontrolled, and therefore unaccounted, use of resources would also mean that operations could be performed which ar not premitted.

Denial of service attacks This attacks is caused by flooding a network with request. Due to the limited throughput services may be "flooded" and can no longer be accessed by users.

Wiretapping of data By monitoring data connections it is possible to gather information on the system and its users. This information can be used for an attack. Password sniffing is a popular method to gain access to a system.

Unauthorized enhancement of privileges Users in most cases have limited access to resources. When users gain administrator privileges they could manipulate data or extend the access to resources.

These security threats can arise on different locations of the setup. Figure 4 demonstrates this. For more information on security threats and their countermeasures refer to [6].

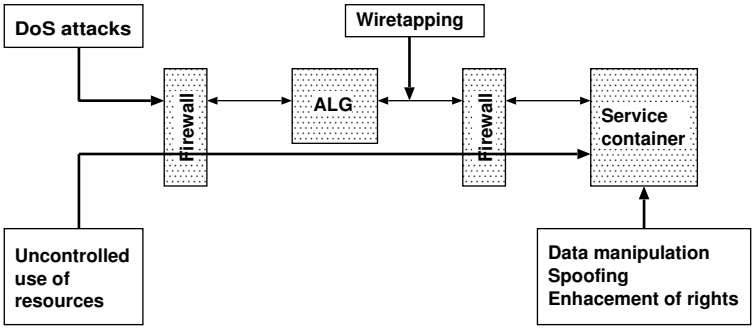

Figure 4: Some security threats and their occurrences.

2.3. Service Orientated Architecture

The Service Orientated Architecture (SOA) became more important over the last years. Currently developed Grid middleware solutions are based upon services. Already existing Grid middleware solutions are extended to support this architecture. This happens due to the fact that new standards emerge out of the Grid community.
Several standards have been defined for Grid based middleware solutions. The following three are of importance when using a service based approach.

2.3.1. Open Grid Services Architecture

The Open Grid Services Architecture (OGSA) [7] has been developed by the Global Grid Forum [8]. It defines a common and open architecture for grid based applications. This is, among other things, done by the a set of service interfaces definitions.
A list of services was specified which should be present in all kinds of Grids. This services include index services, job management service, and VO management services. All these services interact constantly with each other.
Many techniques could be used to realize this requirements. For example when looking at the OGSA definitions a Grid could be realized with CORBA, RMI, or even RPC. But Web services where chosen because of their close relation to the Internet. But to realize a Grid with a services orientated architecture web services are not sufficient.
OGSA requires stateful services. This means that information can be hold across several running instances. Web services do not keep their state information. But for most Grid services this is needed. Therefore Web services should be extended in their functionality.

2.3.2. Open Grid Services Infrastructure

The OGSA only describes what Grid services should have in comparison to Web services. The Open Grid Service Infrastructure (OGSI) gives a more detailed definition of Grid services.
OGSI was also initialized by the Global Grid Forum. It describes a more detailed

and technical view on the services needed for Grids.

2.3.3. Web Service Resource Framework

Due to the fact that the OGSI was not accepted by the Web service community a new standard was developed. The Web Service Resource Framework (WSRF). This standard, just like the OGSI, specifies how Grid Services should be designed. The following figure 5 demonstrates this in more detail.

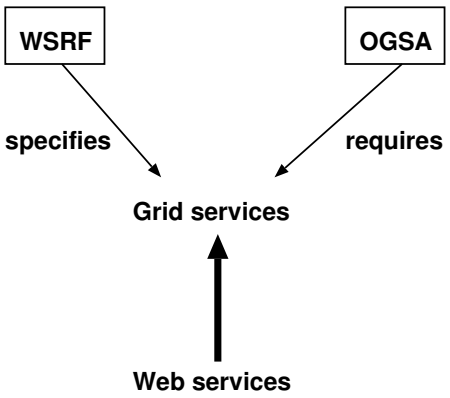

Figure 5: OGSA and WSRF interaction.

While the OGSA requires Grid Services the WSRF specifies them. The WSRF specifications where developed by the Globus Allicance and IBM. Therefore this standard will hopefully be more excepted within the Web service community.
WSRF described how stateful services can be modeled by the help of resources. These resources are deployed with the services and hold the state information. Currently the WSRF standard is being submitted to the OASIS [9] standardization consortium.

2.3.4. Services

Web services (and so Grid services) can be used to realize interactions over an network. Interfaces, being defined by open standards, allow interoperability between platforms and programming languages. The SOA is very similar to the inter process communication.
The interface of a service is described in Web Service Description Language (WSDL). This description of a service gives an overview of the methods a service provides. With the help of this WSDL description stubs are generated which handle the communication between a client and a service or between two services. The interface can be deployed within a Service Broker. So clients can request information about this service.
These stubs realize the communication with SOAP messages. These SOAP messages are send to the stub on the other side. This can be done by the help of several protocols. Mostly this would be HTTP.

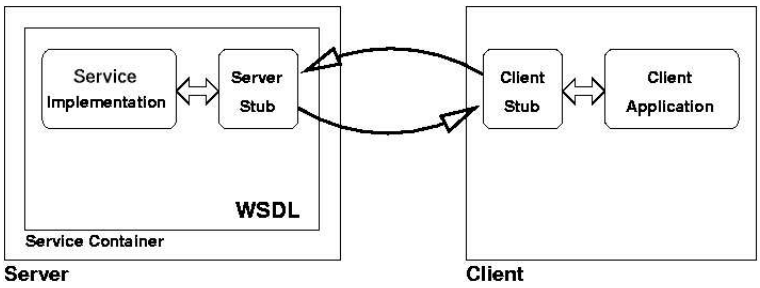

Figure 6: Client and service with their stubs

The service can be implemented in several programming languages. After implementation a service is deployed in the service provider. In most cases this is a service container. Figure 6 demonstrates the use of stubs.
The following steps are necessary when invoking a service.

1. Request the service interface from a service broker. This will return the location of a service and which methods can be called.
2. The client request the service provider to start an instance of the requested service. This is done with the help of SOAP messages.
3. The client invokes a method of the started service. This is done by SOAP messages as well.
4. Further SOAP or Non-SOAP based communication can follow.
5. The client is done. The service provider will destroy the instance of the service.

Figure 7 shows this process chain.

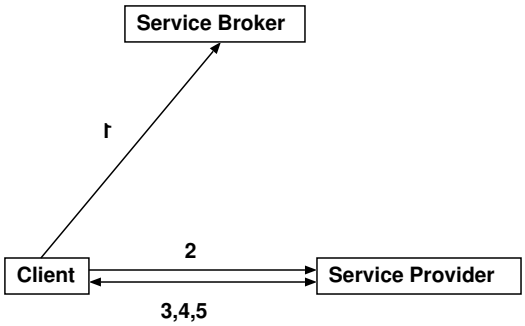

Figure 7: Invoking web services.

2.3.5. SOAP Based Communication

In current Grid infrastructures (e. g. based on the Globus Toolkit), most of the communication is service invocation. That means, the communication is usually based on the Simple Object Access Protocol (SOAP) [10].

SOAP has been specified by the WWW Consortium (W3C) in 1999. It is a definition of the XML based information exchange between peers in a decentralized environment. SOAP messages can be transferred using various protocols, as HTTP or SMTP.
Based on one-way communication, SOAP messages can hold almost any type of data. More complex communication can be accomplished by request and response messages. Messages are sent from a SOAP sender to a SOAP receiver. The sender and the receiver are SOAP nodes.
The data, encapsulated in XML documents, is sent to the SOAP receiver in a so called SOAP envelope. A SOAP message can be passed along several senders and receivers. The path the message follows is called the SOAP path. This chaining of SOAP nodes to a SOAP path allows the use of an ALG.
The SOAP envelope contains a header and a body. The header includes information on the transaction itself. This can be information about the sender and the receiver. Information not depending on the application can also be located here. For Grid Services these may be certificates for authentication.
The SOAP body contains application specific information. This entry is mandatory for a SOAP envelope. The main end-to-end information is contained in it.
Due to the fact that SOAP encapsulates data in XML documents it is easy to validate and modify the data. XML schemas and available parsers deal with these tasks.
SOAP messages can be easily tunneled through firewalls. In most time firewalls ruleset don not have to be changed. But SOAP mechanisms do not provide methods for auditing SOAP messages.

2.3.6. Handling Non SOAP Based Communication

In addition to SOAP based communication further non blocking I/O transfer is needed. Usually this is needed and used for high performance file transfers.
The advantage is that this traffic is still initialized by service based requests. If this type of request is intercepted by the ALG special steps have to be taken.
For most file transfers initialized by service requests a server and a client are started. Between these two participants the files are being transfered. While no direct communication between a client and a resource are allowed, the ALG has to start a server and client as well. The communication path would look like: *Resource (Server) – (Client) ALG (Server) – (Client) Requestor.*
This way no direct connection need to be established. Nevertheless the files can be transfered.

2.4. General requisitions to the Application Level Gateway

Several requirements emerge from the environment where the ALG is going to be deployed. These needs should be part of the use case analyses. User profile descriptions and demands on quality are described in this section.

2.4.1. Scope of Application

This ALG will probably be used in Grids of any size. Most likely it will be used as an interface for combining Grids from different organizations to a VO. This ALG should provide the only way to access Grid resources from another network. It therefore should be able to handle a large amount of requests.
This solution will be used to secure Grid resources therefore it should be secure itself. Administrators must have a secure way to configure the ALG. The ALG itself should be able to use protocols like HTTPS for transport of Grid service requests. However it does not need to deal with security on TCP/IP level, i.e. resistance against Denial of Service (DoS) attacks.
This can be done far more effective by the package filters which divide the DMZ from the Internet. The ALG must provide security on the Application Level. It therefore must understand all protocols that are used for the communication inside a Grid. It must be able to validate requests, identify the types, and authenticate clients and Grid resources. Grid communication often includes file transfers. This ALG must be able to identify file transfers as such and it must be able to scan transferred files for viruses and Trojan horses.

2.4.2. Requirements

Many kinds of Grid and Web services can be deployed in appropriate containers. This can include job managers, authentication services, index services and resource specific services. This wide range of working fields disallows the general inspection of data traffics. Specialized verifications must be possible. This leads to a plug-in based architecture for the ALG. The integration for special services can be realized within plug- ins.
The burden to extend the ALG with new features must be low. This is next to the fast growing number of services types a reason for the plug-in architecture. The need for strict coding conventions and good documentations are obligatory. Because of the platform independence the ALG should be implemented with the Java programming language. Sun Microsystems provides several different runtime environments for different platforms. Another major reason for Java is the high amount of available libraries which support Web services. These libraries support the easy development of plug-ins in the ALG.
The solution described here must be completely transparent. This means that already existing services and their corresponding clients should not be changed. Even third party services, where no source-code is available, should be supported. The user of a client and its services should not realize the ALG is placed in his communication path. Extra authentication and other steps are prohibited.
This ALG can only be part of a security concept. The host on which the system runs should be equipped with a state of the art operating system. Possible security holes should be closed (e.g. unused ports). The ALG can only provide maximum security when other arrangements are made. This includes the use of packet filters (firewalls), virus scanners, VPNs, and intrusion detection systems (IDS).

2.4.3. Demands on quality

Following demands on the quality where specified to match the above prerequisites:

- The ALG system must run on state of the art computers (Pentium IV, 1024MB of Memory and java runtime environment). Due to the wide range of operating systems which are currently in use it should be portable to different systems.
- An actual Java runtime environment should be used to run the ALG system. The actual version of the runtime version is 1.5.
- Load sharing should be possible in later distributions of the ALG. This is to allow load balancing and a faster handling of messages. For example data transfers should be initialized on a special host.
- Configuration and deployment of new plug-ins must be fool-proof.
- Security is one of the main features of the ALG. It is very important to realize this requirement.
- Downtimes of the ALG should be reduced to an absolute minimum. This requires stable and tested software. Availability is of a high priority. The ALG represents the Grid to the user and therefore downtimes can have massive economic consequences.

2.4.4. User Profile

Typical users of the ALG will be experienced system administrators. His skills cover knowledge of security, web services, and firewalls. The company's local site security policy should be known. Direct access to the network infrastructure is obligatory to deploy the ALG in the DMZ.
General task for this user is the setup and maintenance of the system. Administration should only be possible by one user to avoid configuration problems. Access to the configuration files therefore is only permitted to the administrator. This user should also be able to deploy security updates and newly designed plug-ins to the ALG. Developers implement new plug-ins for the ALG system. These plug-ins should then be deployed by the administrator. Security updates include new revisions of the ALG base system.

2.5. Operative Requirements

In order to solve the task of protecting Grid services on the application level a rich functionality is required. The various tasks of the software have been modeled with the Unified Modeling Language (UML). Several Use Case diagrams model different parts of the ALG system. These use cases show what functionality is needed in which part of the Software. This section describes those use case diagrams.

2.5.1. Use case description of the ALG system

The following figure 8 shows the general use case diagram for the ALG system. The use cases described here form the absolute minimum for a security proxy supporting Grid services.

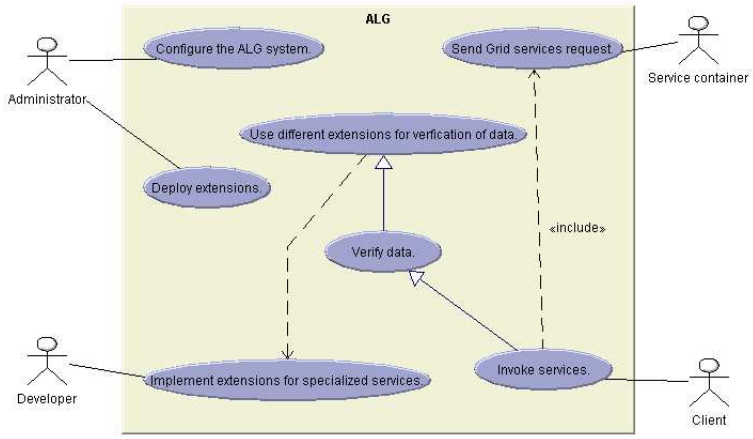

Figure 8: Use case diagram for the ALG system.

Following actors are present within an ALG. Their main tasks are described in the next table.

Nr	Name	Description
1	Administrator	The administrator of the ALG system. Sets up the ALG and maintains it.
2	Developer	Developer writing extensions for the ALG
3	Client	Can either be a service client or another service invoking a service.
4	Service container	Destination of all requests. The actual services are deployed here.

Each use case can be described in more detail. Every use case includes several steps which are necessary to complete the task. In the following table all use cases in this diagram are described.

Nr	Name	Necessary steps	Description
1	Configure the ALG system.	1. Read configuration 2. Validate configuration 3. Get configuration properties	The administrator has to provide the configuration files.

Nr	Name	Necessary steps	Description
2	Deploy extensions.	1. Read configuration 2. Start extension 3. Register extensions within the system	
3	Invoke service	1. Read request from socket 2. Validate the request 3. Reject or accept request 4. Send request to destination 5. Send response from destination to client	Includes the transmission of the request to the destination. It is build upon the verification of the request.
4	Verify data	1. Evaluate the message 2. Validate message 3. Special validation by extensions	Specialized validation is realized within separate extensions.
5	Use different extensions for verification of data	1. Do a specialized verification 2. Accept or reject data	All deployed extensions are called to verify the current request.
6	Implement extensions for specialized services	1. Extend existing extensions 2. Write special verification	
7	Send service request	1. Invoke container 2. Send request 3. Retrieve response	

2.5.2. Use case description of an extension

As already described a very flexible design is needed for this proxy. So extension have to be provided for each service the ALG should support. Figure 9 shows the use case for these extension.
Following actors are part of this use case diagram.

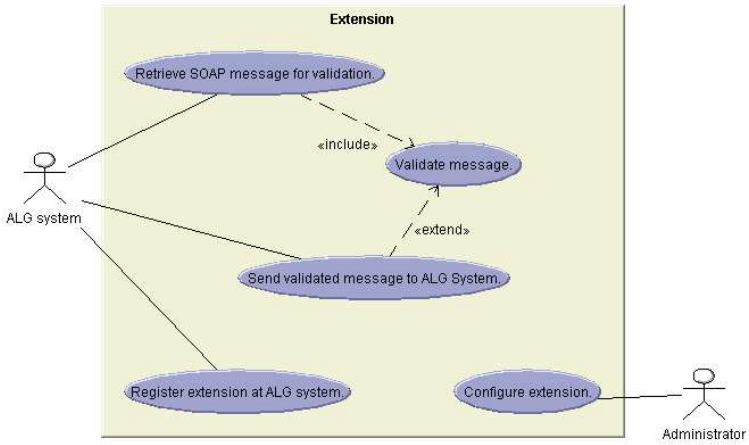

Figure 9: Use case diagram of an extension.

Nr	Name	Description
1	ALG System	The ALG system where this extension is registered.
2	Administrator	The person who maintains the ALG system.

Every necessary step for a use case is described in the following table.

Nr	Name	Necessary steps	Description
1	Register extension at ALG system	1. Discover ALG system 2. Register self 3. Send key	The key is an identifier. When a message matches this key it is send to this extension.
2	Retrieve SOAP message for validation	1. Get SOAP message 2. Validate message	Registered extensions will automatically be used if a message fits their key.
3	Validate message	1. Validate the message 2. Accept or reject the message	
4	Send validated message to ALG system	1. Set status of the message (accepted/rejected) 2. Send message to ALG system	

Nr	Name	Necessary steps	Description
5	Configure extension	1. Deploy configuration files for the extension 2. Validate configuration 3. Read properties	Administrator provides a configuration file for this extension if needed.

3. Implementation of an Application Level Gateway for Grid Services

The service based architecture of future Grids allow the easy auditing of traffic. Without standards (see section 2.3) it would be nearly impossible to design a proxy for Grid services. The next sections describe the design and implementation of an ALG, based upon the requirements of the previous section.

3.1. System architecture

To match the requirements describe in the previous section a system design with plug- ins is chosen for the basic system architecture of the ALG. The plug-in framework used here is a rewrite of the Eclipse plug-in framework. This framework was written to allow the development of plug-ins without the use of the complete Eclipse framework. The design and definition of plug-ins is almost identical.
The extensions and extension points of a plug-in define the interconnections between plug-ins. Plug-ins can dock to an extension point to interact with another plug-in. So an extension describes the use of other extension-points ("to extend a plug-in") while an extension-point itself defines a new port where a plug-in can dock into. Figure 10 shows the basic approach with extension-points and extensions.

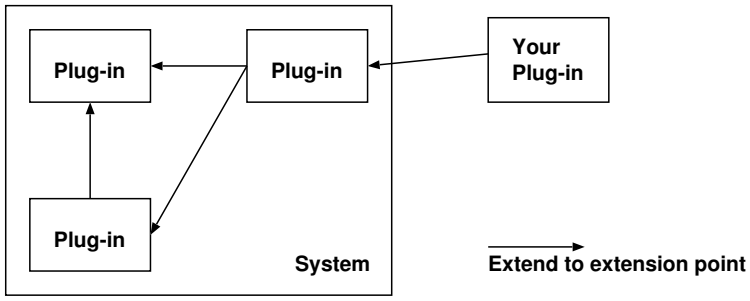

Figure 10: Overview of a plug-in based application.

The plug-in manifest file is a XML file which describes the extensions and extension- points of a plug-in. This file is distributed along with the imple-

mentation of the plug-in. This whole system of plug-ins has to be "booted" by another Boot class. It only initializes the whole system and sets the properties.

3.2. Terminology

To explain the system architecture in detail some terms and definitions have to be defined. Basically all parts of the software described here are plug-ins. To prevent confusion when discussing the plug-in framework some extra terms are defined for the ALG.

Core This plug-in provides extension points for all other modules. It provides basic functionalities which are needed by all modules.

Module A module is basically a major plug-in. It provides functionalities (and so extension-points) for other plug-ins to use. This abstraction allows a cleaner and more efficient system design. With the help of modules similar plug-ins can be grouped together (Within the Eclipse framework a module would more or less meet a feature.).

Plug-in Plug-ins are extensions for modules. They are developed to enhance the support for certain services.

Supplier The supplier module is used to send messages. It is the direct counterpart of the consumer unit.

Consumer The consumer unit accepts incoming connections. It is the first module to pass for an incoming message.

3.3. System design

Figure 11 shows the overall system design of the ALG. Based upon a core system several modules are deployed. The two special modules consumer and supplier form the outer bounds of the system. They handle the incoming and outgoing

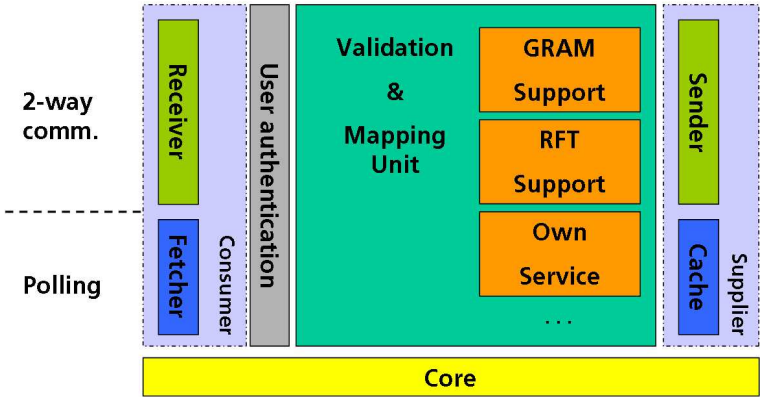

Figure 11: System design of the ALG.

messages. The core system receives and sends all messages from one module to the next one. No direct communication between the modules is allowed. This allows the coupling of several cores (e.g. for load sharing) in later distributions of the ALG.
Basically all messages have to pass the ALG from left to right as seen on the picture. The answers to the request are passed along in the other direction. Messages are delivered to a module by calling the *handleMessage()* from the core. The modules deliver their messages back to the core by calling their method *finalizeMessage()*. This method then calls the *handleMessage()* method which is implemented in the core.

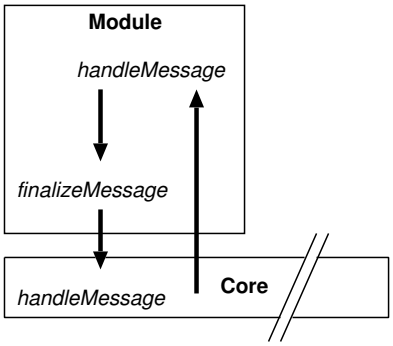

Figure 12: Call hierarchy.

Figure 12 shows this call hierarchy. The first step taken by the ALG is the user authentication. This module is only implemented rudimental during this work. First defined goal for this system is a prototype implementation. User authentication can be realized by the services container as well. For later distributions the authentication should be the first step. Rejections based upon fault authentication can than be handle much earlier.
Next the mapping module parses the incoming messages and validates them. This module has an extension point where plug-ins can dock onto. Within this plug-ins more specific validation of the message content can be implemented. Beside validation this module also modifies the messages. Several tags within the message content have to be mapped. Mapping means that external information is replaced by local (and therefore more secure information). After the messages are successfully validated the supplier passes the request to the destination.
The responses to the request do not have to pass all the steps a request has made. After the supplier receives the response it is passed to the mapping unit and sends to the consumer.

3.4. System implementation

All plug-ins described in the previous section are implemented during this work. Special attention was given to the quality of the code during implementation. This is ensured by the methods described in section 4.
First of all the core and a matching boot loader had to be written. The boot

loader reads the command line arguments and starts up the core. The core itself starts all the modules which are deployed.

3.4.1. The boot loader

Main task of the boot loader is the initialization and setup of the plug-in framework. Therefore it reads the configuration file to get a list of directories in which the modules and plug-ins are deployed. Now it scans within these directories for plug-ins which can be registered to the plug-in framework. They are stored in a Map data structure (moduleLocations).

```
PluginManager pluginManager =
        PluginManager.createStandardManager(moduleLocations);
pluginManager.getRegistry()
        .checkIntegrity(pluginManager.getPathResolver());
```

The *pluginManager* now holds all available plug-ins. Next step is the activation of the core plug-in.

```
Plugin corePlugin = pluginManager.getPlugin(coreID);
corePlugin.getClass().getMethod("startUp", new Class[]
        {}).invoke(corePlugin);
```

By calling *getPlugin()* the plug-in is activated. Next step is to call the *startUp()* method in the core. The last line of code in the code sample above accomplishes this.

3.4.2. The internal message

Within the ALG system messages have to be passed along the available modules. The internal message (*InternalMessage.java*) stores more information than SOAP message itself. The SOAP message is encapsulated within an *InternalMessage*. Within this data structure the SOAP message and additional information is stored.
For example the Type of the message is stored. This can either be a request or an response message. It also keeps a complete history of which module handled this message. This information is useful to look where the validation of the message failed. If the message fails for validation a flag is set so the core knows what to do next with the message.

3.4.3. The core

The plug-in manifest file describing the Core keeps all information needed by the plug- in framework. First of all a unique identifier has to be defined.

```
<plugin id="de.dlr.fireblade.core" version="0.0.1"
        class="de.dlr.fireblade.core.CoreImpl">
```

Within this tag information about the implementation is provided. The concrete class where the source code is implemented is defined. Next extension-points have to be defined so modules can dock onto the core. The definition of the extension point is quite simple:

```
<extension-point id="module">
    <parameter-def id="class"/>
    <parameter-def id="name"/>
    <parameter-def id="description" multiplicity="any"/>
</extension-point>
```

With this definition it is possible to let several modules attach to the core. The multiplicity lets the developer say how many times this extension-point can be used. While this is a very general description of an extension-point, it is possible to use a more strict way where only one specified plug-in is allowed to extend this extension point. The implementation of the core is realized in a class called *CoreImpl.java* which extends *net.java.plugin.Plugin* and implements the core interface. Figure 13 shows the matching UML diagram.

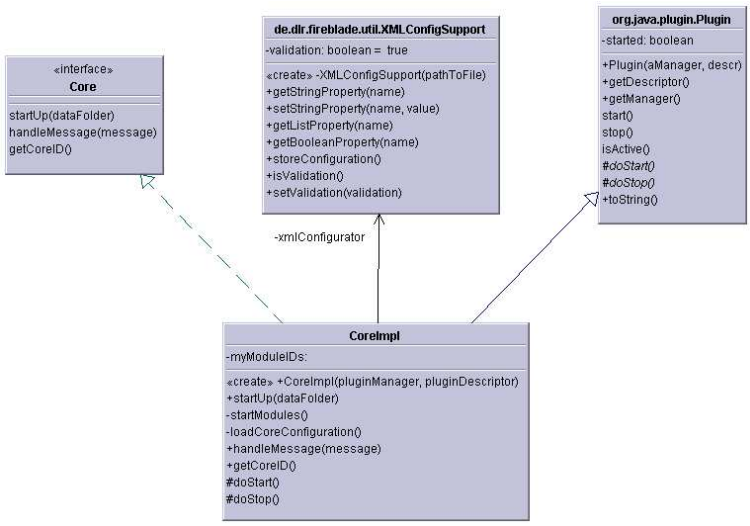

Figure 13: UML class diagram for the core.

By extending from *Plugin* it becomes clear that *CoreImpl* is a plug-in itself. Therefore a special constructor and methods for starting and stopping the plug-in have to be implemented. The core Interface ensures that basic methods needed by the modules are implemented.
After the boot loader started the core it calls the method *startUp()* which is defined by the core Interface as well. Within this method the properties of the core are loaded from a XML configuration file. This is done by a utility class written to support XML configuration files (Instead of java's default properties file). A list with available modules is defined in the configuration. The core starts this modules and calls their *startUp()* methods as well.

All available modules are already registered to the plug-in framework by the boot loader. The core only activates the plug-ins. So for every available module it runs the following lines of code.

```
Plugin module = getManager().getPlugin(currentModule);
Module thisModule = (Module) module;
thisModule.startUp(this)
```

The *currentModule* variable holds the id of the current module. After the initlizaion of the module the method *startUp* is called.
After start-up of all modules the ALG is ready to handle incoming requests. As described the Core realizes the communication between the modules. Therefore the method *handleMessage()* is most important for all the modules attached to the core. This method is called whenever a module finished validating a message. The core now decides which modules should handle the message next.
Several possibilities are available:

- The message was rejected by the module. As consequence an error message has to be generated.
- The message was verified by a module. Next unit has to be determined.

Depending on the direction of the message the next unit has to be determined. The direction of the message is either a request coming from the consumer or an response coming from the supplier.
The error messages are generated within the core. Directly after the generation of the error message the transport direction is changed. A request message becomes an answer message and vis versa. The error message is a standard AxisFault message.

```
errorEnvelope = new SOAPEnvelope();
errorEnvelope.getBody().addFault();
errorEnvelope.getBody().getFault()
        .setFaultString("Gateway rejected request!");
```

This *errorEnvelope* replaces the message in the *InternalMessage*. So the client retrieves a standard error message.

3.4.4. A basic module

To allow easy development of modules for the ALG an abstract class m̃phModule.java is implemented. It defines several methods which have to be implemented by the developer. But it provides methods for general purpose as well.
The class *Module.java* extends from *net.java.plugin.Plugin* so all methods needed for the plug-in framework are already implemented.

The most important method for a module is the *handleMessage()* method. It is called whenever the core wants to forward a message to a module. When the module is finished with a message it is then send back to the core by calling the *finalizeMessage()* method. This method is provided within this abstract class. By overwriting the *startUp()* method the developer can add more calls which should be made during the start of a module.

3.4.5. The consumer module

First step again is the definition of a manifest file. Within this file the unique identifier and the class which represents this plug-in is defined.

```
<plugin id="de.dlr.fireblade.consumer" version="0.0.1"
class="de.dlr.fireblade.module.consumer.Consumer">
```

While the core has an extension point, the consumer extends this. The definition of this extension is done by the following entry in the manifest file:

```
<extension plugin-id="de.dlr.fireblade.core" point-id="module"
        id="ConsumerModule">
    <parameter id="class"
        value="de.dlr.fireblade.module.consumer.Consumer"/>
    <parameter
        id="name" value="ConsumerModule"/>
    <parameter id="description"
        value="Handles incoming requests."/>
</extension>
```

So the consumer module can dock onto the core. Now the core can call the *handleMessage()* method in *Consumer.java*. And the consumer module can call the *handleMessage()* method of the core by calling *finalizeMessage()* in *Module.java*. The consumer is the first step in the process chain of the ALG. So it has to accept incoming data traffic.
Grid services, like web services, use HTTP POST to send their request. The consumer has to listen to a socket so clients can connect. After processing the incoming data, an *InternalMessage* has to be created which is then passed along to the core. When the response arrives it has to be send back to the client.
Figure 14 shows a UML class diagram of the consumer and all its corresponding classes. The Consumer itself reads his configuration from the XML configuration file. After reading the configuration it starts a dispatcher which keeps control of all started threads.
Within this dispatcher a *ServerSocket* is started so clients can connect. On every incoming request from a client a new thread is started. When starting a new thread it is put in the *ThreadPool*. This is to associate an *InternalMessage* with a thread. So responses can be handled by the corresponding thread.
The *ServiceThread* extends from Thread and is the class which finally deals with the InputStream and the OutputStream of the socket. It reads the data from

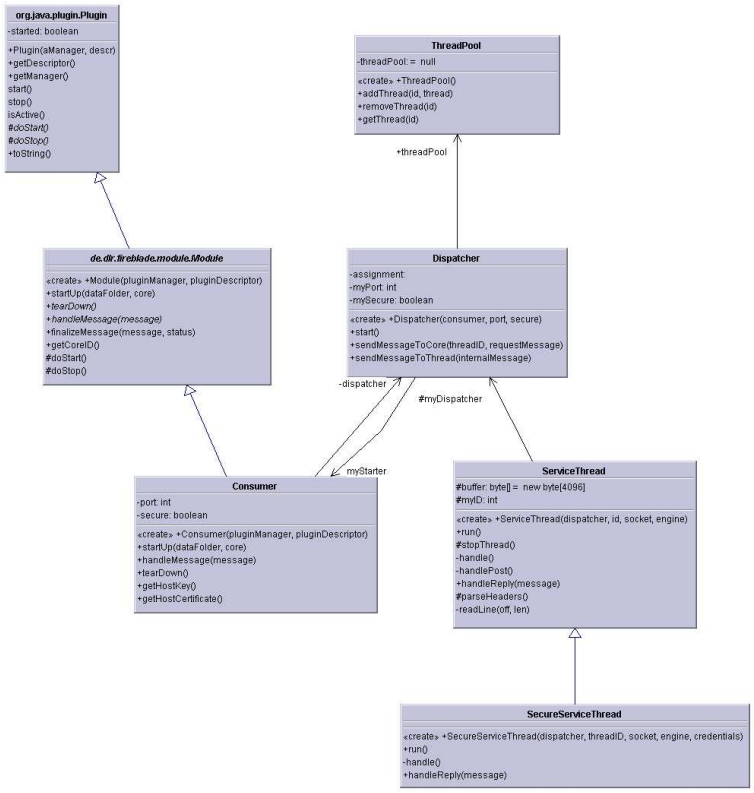

Figure 14: UML class diagram for the consumer module.

the InputStream and parses the HTTP headers. Only if a HTTP POST request is done the thread handles the message. After this verification a SOAP Message is generated by the help of the Axis engine. This SOAP message is placed in an *InternalMessage* and delivered to the core for further validation by the other modules.

3.4.6. Certificates

To support SSL encryption of the data the *SecureServiceThread* extends from *ServiceThread*. Within the constructor of this Thread the socket is initialized to support SSL encryption. All other methods are used from the *ServiceThread*.
To setup an encrypted socket connection credentials are needed. The consumer module has its own credentials. These credentials are used to verify if incoming request are allowed. The CoGkit installation provides the basic security infrastructure. This means all certificates are managed by the CoGkit.
So when the consumer module runs in secure mode a CoGkit has to be installed on the system.

3.4.7. The mapping module

This module realizes the main part of the verification process. The mapping module can be extended by several plug-ins itself. Therefore it needs to have an own extension-point to which plug-ins can extend. The module itself extends

from the core extension-point.

```
<extension-point id="de.dlr.fireblade.plugin">
    <parameter-def id="class"/>
    <parameter-def id="name"/>
    <parameter-def id="description" multiplicity="any"/>
</extension-point>
<extension plugin-id="de.dlr.fireblade.core"
        point-id="de.dlr.fireblade.module" id="MappingModule">
    <parameter id="class"
        value="de.dlr.fireblade.module.mapping.Mapping"/>
    <parameter id="name" value="MappingModule"/>
    <parameter id="description"
        value="Verifies and Modifies SOAP messages."/>
</extension>
```

Several plug-ins can dock onto the extension point of the mapping module. The basic idea of the mapping module is that for every available service a special plug-in is deployed. These plug-ins allow a very strict verification of the traffic passing through the ALG. A request of a type which is not supported by any plug-in is rejected.
Identification of the message type is done with XML schemas. These schemas are defined within the plug-ins. The mapping module fetches these schemas and tries to validate them against the SOAP message. If the SOAP message matches the schema it is being delivered to the appropriate plug-in for further validation. So the following steps are taken to verify an incoming message:

1. Identify and start all available plug-ins. This is done with the help of the plug-in framework.

2. Fetch all paths to the XML schema files from the available plug-ins. These are needed to identify the type of the message.

3. If a SOAP message matches a schema file send it to the plug-in which provided the schema.

4. Retrieve the SOAP message from the plug-in. Verify that no exceptions are thrown. If an exception is thrown the message should be marked as rejected.

Figure 15 shows the steps above in a sequence diagram.
If no schema matches the SOAP message the complete message is rejected. Due to the fact that for every service, which should be supported by the ALG, a plug-in has to be written it is important that the development of plug-ins should be simple. Therefore an abstract class *FirebladePlugin* is implemented. A developer of a plug-in can easily extend from this class to realize his implementation. The next section gives a closer look on the development of plug-ins for the mapping module.

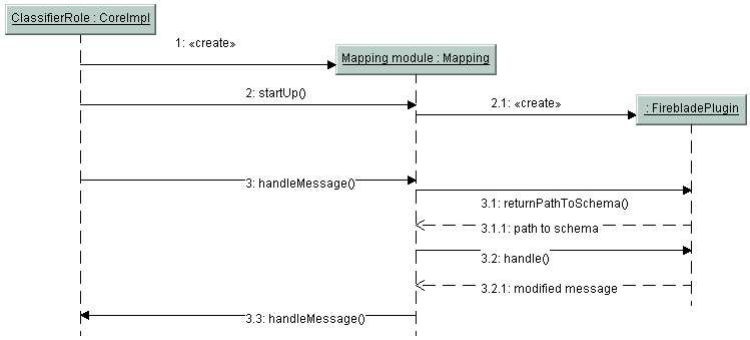

Figure 15: UML sequence diagram demonstrating message handling.

3.4.8. Plug-ins for the mapping module

The development of plug-ins for the mapping module is rather easy. A ANT build file is already provided. The developer only has to create a class which extends from *FirebladePlugin*. With the help of ANT a compress file is created which only has to be uncompress in the plug-in directory of the ALG system.
All plug-ins for the mapping module must extend from *FirebladePlugin*. These plug-ins can dock onto the mapping module by extending from the extension point *de.dlr.fireblade.plugin*. The developer only has to implement two methods. The method *returnPathToSchema* must return an absolut path to a XML schema file. This file is used to validate incoming SOAP requests. If they match this XML schema the method *handle* will be called. This is the second method the developer has to implement. Here he can realize further checks or mappings. For example executables for a Job submission can be verified here.

3.4.9. Plug-in for job submissions

The UML diagram in figure 16 shows a sample plug-in for the mapping module. The *returnPathToSchema* returns a path to the following XML schema definition.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" ...
        elementFormDefault="qualified">
        [...]
        <xs:element name="Envelope">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="soapenv:Header"/>
                                <xs:element ref="soapenv:Body"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:element name="Header">
                <xs:complexType>
                        <xs:sequence>
```

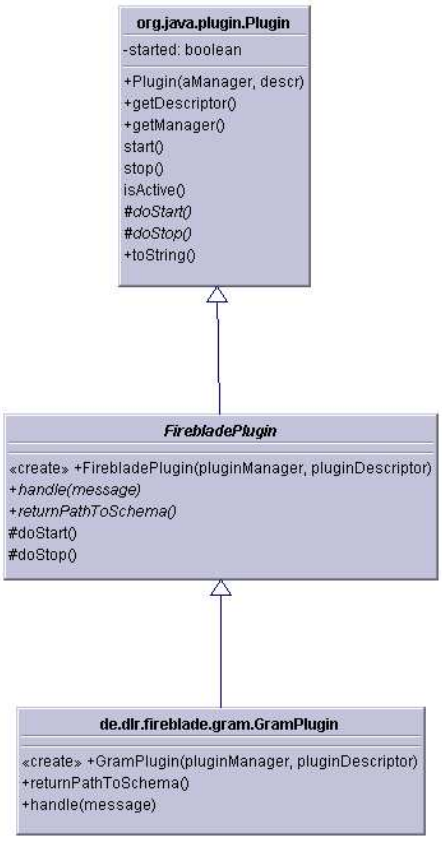

Figure 16: UML class diagram for a sample plug-in.

```
                        <xs:element ref="wsa:MessageID"/>
                        <xs:element ref="wsa:To"/>
                        <xs:element ref="wsa:Action"/>
                        <xs:element ref="wsa:From"/>
                        <xs:element ref="ns1:ResourceID"/>
                    </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:element name="Body">
                [...]
        </xs:element>
</xs:schema>
```

If the SOAP message validates against this schema it is delivered to the *handle* method. The *handle* method itself does only return the SOAP message back to the mapping module. This is for demonstration purposes only.
While the XML schema validation only looks for syntax and sematics (And a loose verification of the contents of the message) of the SOAP message a stricter verification can be realized in the *handle* method. Within this method it is possible to replace certain tags in the SOAP message. Or even more important to verify certain parts of the SOAP message which can not be evaluateted with

the help of XML schemas. For example executables during job submission could be send to a virus scanner.
The interesting part of this plug-in is the definition of the XML schema file. Within this schema file it is possible to realize a strict verification. For example it is possible to define list of possible values for the element with the name *wsa:To*. When doing this the SOAP messages would only validate against this XML schema when a certain service is requested. A list of services could look like:

```
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:enumeration value="https://saga.sistec.kp.dlr.de:7575
                /wsrf/services/ManagedJobFactoryService"/>
        <xs:enumeration value="https://elli.sistec.kp.dlr.de:7575
                /wsrf/services/ManagedJobFactoryService"/>
</xs:restriction>
</xs:simpleType>
```

With this definition only job request to the host *elli.sistec.kp.dlr.de* and *saga.sistec.kp.dlr.de* are allowed.

3.4.10. The supplier module

As last module on the path of a message to its destination the supplier module sends the verified messages. Again this plug-in is threaded to support multiple connections at the same time. The UML class diagram in figure 17 shows the implemented classes.
This plug-in extends to the module extension-point of the core. The same way the consumer unit does. After the method *handleMessage()* is called, a thread is started which opens a socket connection to the destination host. The message is now send and the response is read from the socket and handled back to the core. To support SSL encryption within this plug-in an extra class to support this special type of socket was written. It retrieves user credentials from the local CoGkit installation and opens a socket. The user credentials are then used to authenticate the ALG against the Globus container.

3.5. Countermeasures for Security Threats

In section 2.2, a number of security threats for systems accessible through networks have been listed. Each of these threats must be treated with an adequate countermeasure. The presented application level gateway provides a solution for some of these threats, namely spoofing, data manipulation, and uncontrolled use of resources. Denial of service attacks are usually treated by a firewall. The following table lists all security threats with their respective countermeasure.

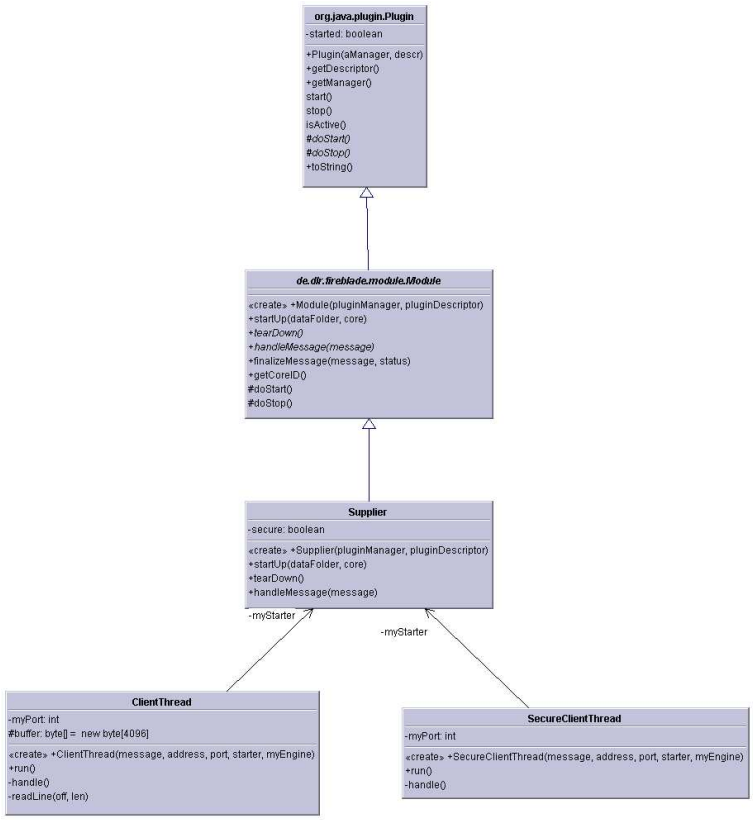

Figure 17: UML class diagram for the supplier module

Threat	**Countermeasure**
User acts as if he is somebody else (Spoofing)	The ALG system or security proxy authenticates and authorizes the user who tries to access the ALG system.
Data manipulation	All data is located in the secure local network. The external user requesting the services can be replaced by an internal user. Full access, and therefore control over the data, is guaranteed.
Uncontrolled use of resources	The ALG inspects incoming and outgoing data traffic. Requests which should not be allowed are rejected.
Denial of Service attacks	The use of packet filters together with the ALG secures the ALG system for these attacks.
Wiretapping of data	Connections are encrypted. Secure data transfer mechanisms (like SSL) are used.

Threat	Countermeasure
Unauthorized enhancement of privileges	The proxy validates incoming data connections. Dubious connections are rejected.

4. Quality management

Several actions where taken to ensure code quality. Quality management is used to avoid failures in software [11]. As showed in section 2.4 reliable code has to be developed. The next sections describe all actions taken during the development of this ALG.

4.1. Source code version control

A version control system allows the centralized storage of the source code. During development new versions of files can be committed to the server. Other users then can updated their locale copy.
All actions taken in a version control repository are logged. So users can keep track of all versions of a file. With a version control system users can merge two versions of a file. Older revisions of a file can be retrieved as well.
A repository can be compared to a tree. Next to the main branch several other branches can exist. In those branches variations in the source code can be stored. Branches can be merged together.
During the development a current release of the Subversion [12] system was used as repository.

4.2. Bug tracking

A web based bug tracking system is used to keep track of feature request and bugs during development. Mantis [13] is such a bug tracking system. It is written in PHP and uses a MySQL backend.
Within this system it is possible to

- report and modify bugs and feature requests. Users can report bugs and add notes to them. Bugs can be assigned to developers so they are solved.
- keep track of users and their assigned bugs. Statistics about bugs and the progress made to solve them can be reviewed.
- manage several projects. With the help of search and filter functions it is possible to keep track of the bugs of several projects.

4.3. Coding style checks

Code style is an important aspect in modern software development. A good coding style avoids faults in source code and allows the easy creation of API specifications.
To ensure that all coding style guidelines are followed up two quality measures are taken. First the Checkstyle [14] tool verifies the written source code. It looks for naming conventions, code style, and documentation offenses. This tool is integrated within the build process of the ALG.
Second method to provide a high coding standard is the use of code reviews. During this audit source code is verified to ensure that correct algorithms are used. During this reviews all guidelines are verified which can not be covered by Checkstyle.

4.4. Unit tests

Unit tests are written by the developer to test a specific area of the source code. By means of extreme programming a unit test is designed before the source code is written. So the developer first has to design what all methods of a class would do before he implements them. [15]
Unit tests ensure a clean design and reduce the amount of time spend in debugging. While unit tests are written at the beginning of a software development cycle it ensures that source code (even in later revisions) always act the same.
During the development of this ALG for every class file their exists a unit test. In this unit tests the methods of a class are tested like described in the following listing.

Test for success Methods are tested if they can be called with correct parameters. All parameter ranges should be included here.

Test for failure The methods are tested if they throw exceptions as expected. This can occur when variables are not set or parameters are out of the expected range.

Test for sanity Tests if methods return the right value when called with known parameters. The return value should be as expected. Mutators are tested if setted variables can be retrieved correctly.

The JUnit [16] framework was used to test the software.

4.5. Code coverage

The existence of unit tests alone is in most cases not enough. It does not warrant that the tests cover all methods and all statements. To test how good the tests cover the code base a tool like clover can be used.
Clover runs the unit tests and simultaneously keeps track of which statements are called. Conditional branch coverage is also verfied during a junit testrun with

enanlbed clover. Later on, in a report, clover states how many statements where called and how the conditional statements resolved. Clover [17] gives an actual percentage of how many lines of code where covered by the unit tests.

4.6. Teamsite

To store documents (Documentation, Conference submission) written during the development of the ALG a teamsite was arranged. Within this teamsite links related to this project where stored.
Information of corresponding projects was stored here. It also keeps track of upcoming events and mile stones. This teamsite can easily be integrated within the desktop environment of the developer.

5. Conclusion

This paper presents concepts for securing Grid Services and the rough design of an application level gateway. The work on this topic led to the conclusion that with an appropriate concept, modern services based distributed environments can be secured. This concept includes the use of firewalls and security proxies.
This implementation of the ALG demonstrated that it is possible to realize a security proxy. With the design concept, using plug-ins, it can be easily extended to match all kinds of requests. For further development the ALG will be extended to support Load balancing.
The main bottleneck in the system design is the support for performance critical applications. Inspecting requests on the application level is time consuming. Data transfers can be either handled with or without caching. If files are cached, they can be validated as well. But that would lead to file transfers not being performed in real-time. If files are not cached, they cannot be inspected, which in turn would lead to a potential security risk. But only this way file transfers can be performed with a minimum delay.
Parts of this work are being contributed to the Firewall Issue Research Group (FI-RG) [18] of the Global Grid Forum (GGF) [8].

A. Used software

This section describes the software used during the development of the ALG.

A.1. Eclipse platform

"The Eclipse platform is designed for anything, but nothing in particular" [19]. The main goal of the development of the Eclipse Platform is to provide an Integrated Development Environments (IDEs). These IDEs can be used for the creation of various applications like Java or C++ programs, but also to create extensions (plug-ins) to the Eclipse platform itself.
The Eclipse platform together with several tools plugged into it specifies an IDE. Eclipse' base packaging contains all tools needed for Java software development. Some of its key features are tools for management of all Java project files, an editor component, a refactoring tool, a debugger, Java compiler integration, etc. The platform's runtime environment is responsible for the management of all plug-ins. Plug-ins are attached to it and may provide extension points to other plug-ins.
Further information and downloads for the open source IDE Eclipse and the Eclipse Platform can be found on the Eclipse home page [20].

A.2. Object Domain

Object Domain [21] is a UML modeling tool. It supports all kinds of UML diagrams. The main advantage of this UML tool is the support for code synchronization and roundtrip engineering. This means that during the development the UML diagrams can be easily updated.
Object Domain supports Java and Python to generate source code. Reverse engineering and the generation of reports about the project are included as well.

A.3. Log4j

Log4j [22] is a logging toolkit for the Java programming language. It allows the logging of all actions within an application. The messages send to the logger can be of different levels. This messages can include information, debug, and warning messages.
These messages let the user keep track of what happens within the application. With the help of so called appenders it is possible to use different log levels for different parts of the codes. This means e.g. that only errors messages from the supplier should be logged and that all messages (including debug messages) from the core should be distributed.
Protocols are generated with the help of log4j. These protocols let the administrator of a site know which services were requested. Rejected requested are logged as well.

A.4. ANT

Apache Ant [23] is a Java based build tool. It integrates all the advantages of GNU make without the confusing makefiles. The tasks Ant has to realize are defined in a XML file. Serval kinds of tasks can be defined. E.g. for the creation of distributions or just the compiling of files.
Tools like junit, checkstyle, and clover have their own Ant tasks. So during development it is possible to execute these tasks and verify their output. With this concept it is also possible to realize Junit test during nightly-builds.

A.5. Axis

Apache Axis [24] is an implementation of SOAP. It hides the communication of SOAP messages from the developer. With Axis it is possible to write Services and Clients without the exact knowledge of communication protocols. Axis supports developers while developing services and their corresponding clients.

A.6. Java Plug-in Framework

The java plug-in framework [25] used here is an Open Source project. This framework implements a runtime engine which discovers the plug-ins and registers them to a registry. The framework maintains this registry of available plug-ins. Extensions and extension points can be used to let plug-ins communicate with each other. Plug-ins can be added to the running application on the fly. This framework is very similar to the plug-in framework used in the Eclipse platform.

A.7. Globus Toolkit

The Globus toolkit is an open source software product to create grids. In the newly distributed version 4.0 it uses a Service orientated approach. It is based upon standards like the OGSA and WSRF.
"The toolkit includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability. It is packaged as a set of components that can be used either independently or together to develop applications. Every organization has unique modes of operation, and collaboration between multiple organizations is hindered by incompatibility of resources such as data archives, computers, and networks. The Globus Toolkit was conceived to remove obstacles that prevent seamless collaboration. Its core services, interfaces and protocols allow users to access remote resources as if they were located within their own machine room while simultaneously preserving local control over who can use resources and when." [26]

A.8. CoGKit

"Commodity Grid (CoG) Kits allow Grid users, Grid application developers, and Grid administrators to use, program, and administer Grids from a higher-level

framework. The Java and Python CoG Kits are good examples. These kits allow for easier and more rapid application development. They encourage collaborative code reuse and avoid the duplication of effort among problem solving environments, science portals, Grid middleware, and collaboratory pilots."[27]

B. List of abbreviations

ALG Application Level Gateway

CFD Computational Fluid Dynamics

CORBA Common Object Request Broker Architecture

DMZ Demilitarized Zone

DoS Denial of Service

GASS Grid Access to Secondary Storage

HTTP Hyper Text Transfer Protocol

HTTPS Secure Hyper Text Transfer Protocol

IDS Intrusion Detection System

OGSA Open Grid Services Architecture

OGSI Open Grid Services Infrastructure

RMI Remote Method invocation

RPC Remote Procedure Calls

RTF Reliable File Transfer

SAML Security Assertion Markup Language

SMTP Simple Mail Transfer Protocol

SOA Service Orientated Architecture

SOAP Simple Object Access Protocol

SSH Secure Shell

TCP Transfer Control Protocol

UDP User Datagram Protocol

VO Virtual Organization

VPN Virtual Private Network

W3C WWW Consortium

WS-DBC Web Service Domain Boundary Controller

WSDL Web Service Description Language

WSRF Web Services Resource Framework

XML eXtensible Markup Language

List of Figures

References

[1] Foster, I., Kesselmann, C. The Grid - Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1998

[2] Foster, I., Kesselman, C., Tuecke, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International J. Supercomputer Applications, 15 (3), 2001.

[3] Foster, I., Kesselman, C., Nick, J. and Tuecke, S. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Globus Project, 2002, www.globus.org/research/papers/ogsa.pdf.

[4] Czajkowski, K., Ferguson, D., Foster, I., Frey, J., Graham, S., Sedukhin, I., Snelling, D., Tuecke, S., Vambenepe, W. 2004. The WS Resource Framework. http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf

[5] XML/SOAP Firewall, Xtradyne WS-DBC - the XML/SOAP Firewall for Enterprises. http://www.xtradyne.com/products/ws-dbc/ws-dbc.htm

[6] Meier, J. D., Mackman, A., Vasireddy, S., Dunner, M., Escamilla, R., Murukan, A. Improving Web Application Security: Threats and Countermeasures. Microsoft Corporation, 2003.

[7] Foster, I., Kishimoto, H., Savva, A., and others The Open Grid Service Architecture, Version 1.0. http://www.gridforum.org/documents/GWD-I-E/GFD-I.030.pdf

[8] Global Grid Forum. http://www.ggf.org

[9] Organization for the Advancement of Structured Information Standards. http://www.oasis-open.org/home/index.php

[10] SOAP Specifications. http://www.w3.org/TR/soap/

[11] Hunt, A., Thomas, D. The pragmatic Programmer - From Journeyman to Master, Addison Wesley Longman, Inc. 2000

[12] Subversion website. http://subversion.tigris.org/

[13] Mantis website. http://www.mantisbt.org/

[14] Checkstyle website. http://www.checkstyle.sourceforge.net/

[15] Hunt, A., Thomas, D. Pragmatic Unit Testing, The Pragmatic Bookshelf, 2004

[16] Junit website. http://www.junit.org/

[17] Clover website. http://www.cenqua.com/clover/

[18] Firewall Issues Research Group. http://forge.gridforum.org/projects/fi-rg

[19] Eclipse Platform Technical Overview, Object Technology International, Inc., February 2003.

[20] Eclipse home page. http://www.eclipse.org

[21] Object domain website. http://www.objectdomain.com/

[22] Log4j website. http://logging.apache.org/log4j/

[23] Apache Ant website. http://ant.apache.org/

[24] Apache Axis website. http://ws.apache.org/axis/

[25] Java plug-in framework. http://jpf.sourceforge.net/

[26] Globus website. http://www.globus.org/

[27] von Laszewski G., Foster I., Gawor J., Lane, P. A Java Commodity Grid Kit, Concurrency and Computation: Practice and Experience, vol. 13, no. 8-9, pp. 643-662, 2001.