

University “Politehnica” of Bucharest, Romania
Faculty of Automatic Control and Computers
Computer Science and Engineering Department
L3S Research Center and University of Hannover, Germany

Distributed Policy-Based Dynamic Negotiation for Grid Services Authorization

by

Ionut Constandache

September, 2005

Diploma Thesis
for Bachelor of Science Degree

Supervisors:

Prof. Dr. Wolfgang Nejdl

Prof. Dr. Valentin Cristea

Dipl. Ing. Daniel Olmedilla

Contents

1	Preface	4
2	Introduction	5
3	Grid overview	7
3.1	Grid: Concepts and Goals	7
3.2	Examples of Deployed Grids	10
4	Authentication and Authorization in Globus Toolkit 4.0	13
4.1	Introduction to Globus Toolkit 4.0	13
4.2	Grid Security Infrastructure	17
4.3	Credential Repositories	20
5	Grid Scenario	23
5.1	Submitting a job over the Grid	23
5.2	Limitations and Assumptions	26
6	Policy Based Negotiation for Trust Establishment	29
6.1	Trust Negotiation	29
6.2	PeerTrust Language	31
6.3	Example	33
7	Policies, Rules and the Semantic Grid	36
7.1	Relation with the Semantic Grid and the Semantic Web	36
8	Distributed Policy Based Negotiation for GRID Services Authoriza- tion	38
8.1	Negotiating Grid Service Access	38
8.2	Enhanced Authorization Architecture	46

9	Architecture. Implementation	48
9.1	High Level Architecture	48
9.2	Implementation	50
10	Related Work	59
10.1	VOMS	59
10.2	PERMIS	59
10.3	AKENTI	60
10.4	PRIMA	61
10.5	Community Authorization Service (CAS)	62
10.6	GridShib	62
10.7	XPOLA	63
11	Conclusions	64
11.1	Future Work	65
11.2	Summary	65
11.3	Acknowledgements	66
A	WSDL files	71
A.1	TrustNegotiation port WSDL file	71
A.2	Service WSDL file	73
A.3	Service Deployment Descriptor WSDD	75
B	Credentials	77
B.1	X.509 End Entity Certificate	77
B.2	X.509 Proxy Certificate	78
B.3	SAML Assertion	78
C	Code	81

List of Figures

3.1	VO - Virtual Organization for Grid resource sharing	8
3.2	Network for Earthquake Engineering Simulation resource distribution - http://www.nees.org/xcutting/about/facilities.php	11
3.3	TeraGrid resources http://www.teragrid.org/userinfo/guide_hardware_table.html	12
4.1	X.509 Certificate and Proxy Chain Certificate Validation	19
5.1	Simple Grid Scenario	25
8.1	Grid Portal MyProxy Server Negotiation	40
8.2	Job Submission - CAS, Grid Portal, Linux Cluster Negotiation . .	43
8.3	Job Negotiations	44
9.1	High Level Architecture	49
9.2	Low Level Architecture	52
9.3	Flow Diagram	54
9.4	Sequence Diagram	55
9.5	Proof and Proof Tree computed on the grid service side	56

Chapter 1

Preface

I developed my diploma thesis during a six month internship at L3S Research Center and University of Hannover, Germany. L3S Research Center has been founded in 2001 as a joint institute of University of Hannover, Technical University Carolo-Wilhelmina at Brunswick, and Brunswick School of Arts. In its first three years, the L3S has established itself as a well-renown partner in national and international IST projects. It coordinates Network of Excellence PROLEARN on Technology Enhanced Learning, in the context of the 6th EU/IST research program, it participates as core partner in the Networks of Excellence KnowledgeWeb and REWERSE, whose work focuses on Semantic Web and Knowledge Technologies and is partner in the EURON Network on Robotics and Wallenberg Global Learning Network (WGLN)). L3S projects include research, consulting, and technology transfer in all of these areas, provision of infrastructure and support for teaching and learning technologies at the participating universities, and collaboration with both German and international standardization bodies.

My work focused on research in Grid computing, an area just started at L3S. I was one of the initiators of the Grid effort currently undergone by L3S, and my diploma thesis is part of the research I conducted here on Grid enhanced authorization mechanisms. I was also involved in research on trust and policy based authorization schemes, in the context of the Network of Excellence REWERSE and the PeerTrust Project. During the six month internship at L3S Research Center I co-authored two papers submitted for publication at the time of this writing, and made several presentation on security and Grid related topics.

Chapter 2

Introduction

Grid environments provide the middleware needed for access to distributed computing and data resources. Distinctly administrated domains form virtual organizations and share resources for data retrieval, job execution, monitoring, and data storage. Such an environment provides users with seamless access to all resources they are authorized to. In current Grid infrastructures, in order to be granted access at each domain, user's jobs have to secure and provide appropriate digital credentials for authentication and authorization. However, while authentication can be automated and single sign-on achieved based on client delegation of X.509 Proxy Certificates [50] to the job being submitted, the authorization mechanisms are still mainly identity based. Due to the large number of potential users, different certificate authorities and static identity mappings, scalability and availability issues are inherent, calling for a complementary solution to the access control mechanisms specified in the current Grid Security Infrastructure (GSI) [21].

In this thesis we introduce an extension to the Grid Security Infrastructure and Globus Toolkit 4.0 in which policy-based negotiation mechanisms offer the basis for overcoming authorization limitations in Grid environments.

Our proposed extension relies on trust negotiation in which parties establish trust gradually through requests and disclosure of credentials in an iterative and bilateral process. Parties define authorization policies and verify during the trust negotiation process if an authorization decision can be inferred. The authorization policies we utilize, accommodate requirements for user capabilities thus the authorization decision relies on users demonstrated properties, rather than on their identities.

Access is negotiated, being no longer a one step action in which the client is expected to and provides the required credentials to the service. Access is managed during several steps in which clients query for service access policies, disclose credentials or require, at their turn, the service to make proof of certain properties, so that a trust relation can be established. The authorization policies

are self explanatory indicating where to obtain the credentials needed to satisfy them. Based on this the authorization mechanism we provide, permits automatic gathering of certificates thus releasing the user from the difficult task of tracking resources and their associated requirements, or hardcoding the repositories from where credentials are to be retrieved. Furthermore, the negotiation process is traced at any moment with the engaged entities able to check its status, find out why it has failed or how it evolved (what policies and credentials have been used) until access was granted.

The authorization decision is distributed, with parties able to push or pull rights from third parties configured either in the authorization policies, or pointed out during the trust negotiation process. Moreover the policies are specified autonomously at resource level, adhering to the Grid authorization model in which the resource provider has the ultimate control over the resources shared.

All these features provide the ingredients for an advanced self-explanatory and automatic access control for Grid resources, which we have achieved by the addition of semantics to authorization policies and reasoning on policies specified and credentials disclosed. Our implementation is distributed as an API and jar files, allowing a straightforward integration with GT 4.0 grid services and client development. With small changes to configuration files (descriptors), and just a few lines of code added to source files, deployed Grid services can easily plug in the authorization enhancements we propose.

The thesis is organized as follows: chapter 3 gives a brief overview of Grid concepts, chapter 4 introduces Globus Toolkit 4.0, authentication and authorization, and chapter 5 describes limitations and motivates our approach. Chapter 6 presents policy-based trust negotiation and chapter 7 links our approach to the Semantic Web and the Semantic Grid. We illustrate in chapter 8 our view of a fully automatized access scheme for Grid. Our proposed architecture along with a description of our implementation is given in chapter 9. Chapter 10 presents related work and chapter 11 concludes our presentation.

Chapter 3

Grid overview

3.1 Grid: Concepts and Goals

The Grid as described in [18] deals with *coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations*. Sharing encompasses direct access to remote computers, software tools, scientific instruments, data repositories and other physical devices pertaining to distinctly administrated institutions. Grid goes beyond sharing individual resources, providing the infrastructure needed for their coordinated use as grid applications involve computations scattered over distinct organizational domains. Sharing relationships among the organizations present over the Grid are established specifying what are the conditions, restrictions and rights for addressing these resources. Resource providers, organizations and consumers who have defined such sharing relationships form a Virtual Organization (VO). In figure 3.1 an example of a VO is depicted, with several institutions defining rules for sharing their locally available resources for a large scientific computational simulation. As the image emphasizes, a dynamic VO is created by the coordinate use of organizational distributed resources.

Grid is all about using distributed resources under different administrative domains. This raises a series of technological issues: credential management for consumer/resource authentication, specification and enforcement of authorization policies, secure access to remote resources, co-allocation of multiple resources, discovery of available services/resources, data location and transport to and from storage repositories,

Sharing relationships are highly-dynamic as resources may be part of several VOs (with associated rights), may pop up or go down more like in a peer-to-peer scenario, and new consumers may be interested in addressing them. Mechanisms for prompt enabling of such relationships, their discovery and understanding have to be provided to fully empower the Grid paradigm.

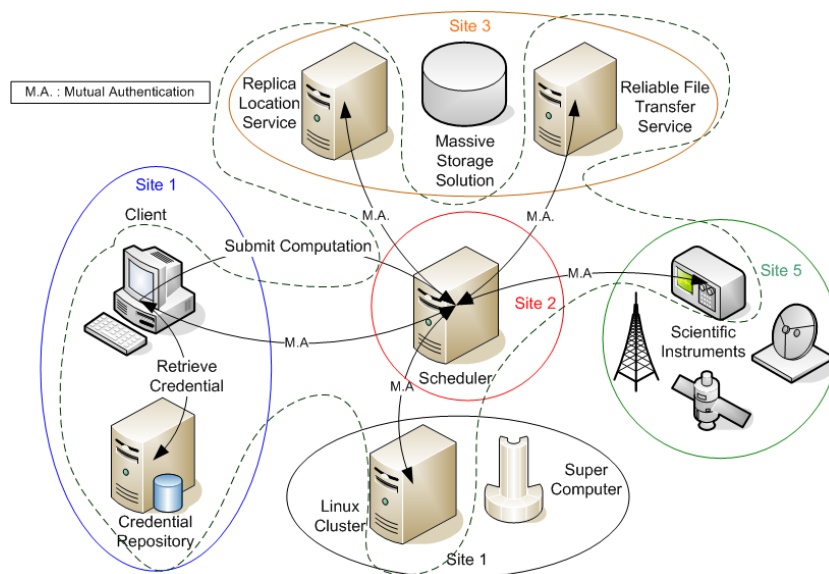


Figure 3.1: VO - Virtual Organization for Grid resource sharing

Grid Architecture has to provide the means for sharing among arbitrary parties. It has to abstract the characteristics of each site involved and offer a common infrastructure so interoperability of diverse resources can be accomplished. Achieving interoperability requires common protocols and standards. A common set of specifications facilitate the interactions between actors over the Grid, enabling rapid establishment of light-weight sharing relationships based on the same language for discovery, security and message representation. Based on standard protocols, standard services can be developed offering access to computational resources, data management, scheduling and resource discovery. The Grid has also to accommodate user applications and encourage their fast development by defining interfaces and APIs to the Grid Infrastructure.

A Grid architecture is required to glue together an extensive set of features as described in [18] of which we should mention:

- Mechanisms for starting programs and monitoring their execution
- Querying functions for determining resource characteristics: state information, load, hardware and software capabilities
- Data storage and location
- Manage network transfers: network characteristics and load
- Single sign on: users should be able to authenticate once and have access to all resource they are entitled to

- Delegation: users should have the possibility of delegating some of their rights to programs or resources acting on their behalf
- Integration with local security solutions: the Grid infrastructure should not affect the locally deployed security solutions, but bridge between them
- Directory services allowing VO members to dynamically register and query for resources available over the Grid
- Co-allocation, scheduling and brokering services leveraging the collaborative use of resources at different sites
- Data Replication Services managing the timely availability of data
- Problem solving environments
- Community Authorization Servers releasing capabilities and access policies established at the VO level
- Community accounting and payment services

In recent years, rapid development and wide acceptance of Web services offered the building blocks for leveraging Grid technologies through the adoption of Web services patterns in the development of an Open Grid Service Architecture (OGSA) [17]. Web services use XML based languages for describing exposed methods, parameters needed for invocation, and discovery mechanisms for identifying the relevant service providers for specific functionalities. Web services are primarily described and called through XML based messages being independent of the programming language, OS or hardware available with the service provider. Web services interoperate through the use of a set of standards like Simple Object Access Protocol (SOAP [25]), Web Services Description Language (WSDL [3]) and Universal Description, Discovery and Integration (UDDI [1]). Through the adoption of Web services, Grid can address requirements imposed by distributed heterogeneous environments: dynamic interactions with previously unknown services, composition of services, discovery of service functionality and virtualization (abstraction behind a common interface of diverse implementations with no interference to the hardware/software solutions adopted by local sites). The Open Grid Service Architecture (OGSA) [45] reflects the alignment and augmentation of Grid and Web services by proposing a set of interfaces, behaviors, resource models and bindings. OGSA proposes conventions and WSDL interfaces for Grid services construction, discovery and creation of transient Grid service instances. OGSA defines the Grid service as a potentially transient stateful web service with support for reliable and secure invocation, state notification, lifetime, policy and

credential management and virtualization. In conformity with OGSA recommendations, Grid should be composed of an extensible set of Grid services able to be aggregated in a multitude of configurations enabling the creation of virtual organizations.

Grid applications extend over numerous resources, may run for long periods of time and may need to use data that persists over web service invocations. For these reasons one identified Grid requirement was support for stateful web services. While web service providers have found different solutions for maintaining the state (named “resource”) associated with a web service from one invocation to another, Grid required defined specifications for interoperable use of web service state. Web Service Resource Framework (WSRF) [47] is a set of specifications describing creation, addressing, inspection, and lifetime management of stateful resources and the pairing between a web service and such a resource. WSRF stays at the basis of current grid services and is fully supported by the most popular toolkit for Grid deployments: Globus Toolkit.

3.2 Examples of Deployed Grids

Earth System Grid [13] has as its main goal enabling analysis of data generated by global Earth System models. Climate models produce data of petabytes order which must be made available to scientists at laboratories, universities, research centers etc. scattered on large geographical areas. Distributed federations of supercomputers need to conjugate with large-scale data repositories and analysis servers in order to locate and use the required data. On top of such a system portals are deployed in order to offer seamless access to the full power of the Earth System Grid Infrastructure. Efficient solutions are implemented as moving data across sites is costly. An important effort is invested in remote data access and discovery as often the computed data, due to its large size, has to be left at the originating site.

Network for Earthquake Engineering Simulation [30] offers an extensive infrastructure targeted at simulating and diminishing impact of earthquakes and tsunami like disasters. NEES comprises of 15 shared experimental facilities including shaking tables, a tsunami wave basin, geotechnical centrifuges, field experimentation and monitoring, repositories for sharing documents, experiments and simulation programs all of them building up one large-shared laboratory. The NEESgrid is constructed around a service oriented architecture providing reliable and secure transport of data, services for capturing data produced by scientific instruments, reliable and long term data archive services, collaboration services for discussion groups and document sharing, telepresence services for remote par-



Figure 3.2: Network for Earthquake Engineering Simulation resource distribution - <http://www.nees.org/xcutting/about/facilities.php>

participation in experiments and real-time video and data viewing, data search and analysis services, visualization services, all leveraged by a GRID IT infrastructure interconnecting resources spread over US territory (see figure 3.2).

TeraGRID [43] was completed in September 2004 offering to users over 40 teraflops of computing power and around 2 petabytes of data storage. TeraGrid bridges a total of nine sites: *National Center for Supercomputing (NCSA)* at the University of Illinois, Urbana-Champaign, *San Diego Supercomputer Center (SDSC)* at the University of California, San Diego, *Argonne National Laboratory* in Argonne, Illinois, *Center for Advanced Computing Research (CACR)* at the California Institute of Technology in Pasadena, *Pittsburg Supercomputing Center (PSC)* at Carnegie Mellon University and University of Pittsburg, *Oak Ridge National Laboratory (ORNL)*, *Purdue University*, *Indiana University* and the *Texas Advanced Computing Center (TACC)* at the University of Texas at Austin. Each of these sites share their specialized resources making available at GRID level massive CPU power, large data storage solutions and visualization services. Objectives of the TeraGrid infrastructure is to provide large computational capabilities to the open research community, to obtain an integrated environment capable of carrying out applications on distributed resources and to create an infrastructure permitting readily integration of additional sites. Figure 3.3 presents a summary of the resources available over the TeraGrid.

<i>Sites</i>	<i>Computational Resources</i>
SDSC	Itanium2/IA-64 SUSE Linux SLES8/ 2.4 SMP, 4 TF, 512 CPU Power4+ (DataStar) IBM AIX 5L 5.2, 0.65 TF, 96 CPU Online Storage: 540 TB / Archival Storage: 6 PB / Visualization: YES
SA	Itanium2/IA-64 SUSE Linux SLES8/2.4 SMP , 5.2 TF Phase I: 512 CPU Phase II: 1262 CPU, 6 TF SGI Altix SGI ProPack 3.4 6.5 TF 1024 CPU Online Storage: 230 TB / Archival Storage: 1.5 PB
UC/ANL	Itanium2/IA-64 SUSE Linux SLES8/2.4 SMP , 0.5 TF Phase I: 32 CPU Phase II: 92 CPU Xeon/IA-32 SuSE Linux SLES8/2.4 SMP, 0.5 TF, 192 CPU Online Storage: 20 TB / Visualization: YES
Caltech/ CACR	Itanium2/IA-64 HP SuSE Linux SLES8/2.4 SMP 32 CPU IBM SuSE Linux SLES8/2.4 SMP 0.8 TF 72 CPU Online Storage: 170 TB
PSC	Alpha EV68 (Lemieux) Tru64 Unix, 6 TF, 3000 CPU Alpha EV7 (Rachel) Tru64 Unix, 0.3 TF, 128 CPU Cray XT3 (Big Ben) Front End: SuSE Linux, Compute processors: Catamount, 10 TF, 2090 CPU Online Storage: 200 TB / Archival Storage: 2.4 PB / Visualization: YES
IU	Itanium2/IA-64 SUSE Linux SLES8/ 2.4 SMP, 0.166 TF, 32 CPU Online Storage: 6 TB / Archival Storage 150 TB / Visualization: YES
PURDUE	Heterogeneous IA-32 Cluster Debian/sarge Linux, 1.7 TF, 960 CPU IBM Power3-II IBM AIX 5.1L 320 CPU
ORNL	Intel Xeon SUSE Linux 9.1, 56 CPU
TACC	Intel Pentium 4/IA-32 (Lonestar) Redhat Linux 7.3, 5.2 TF, 856 CPU UltraSPARCIV (Maverick) Solaris 9, 128 CPU Online Storage: 50 TB / Archival Storage: 2 PB

Figure 3.3: TeraGrid resources
http://www.teragrid.org/userinfo/guide_hardware_table.html

Chapter 4

Authentication and Authorization in Globus Toolkit 4.0

Globus Toolkits offer the plumbing of different protocols, specifications, standards and interfaces to meet the requirements identified by OGSA for supporting a Grid Infrastructure. Along their development, starting in 1990, Globus Toolkits addressed issues like: resource discovery, remote execution monitoring and management, data movement and location and security in service oriented distributed environments. In the following sections we are going to provide a small introduction to Globus ToolKit 4.0 (GT4.0) emphasizing the broad area of services it supports. We will focus on the current Grid Security Infrastructure on which GT4.0 relies and further present some of the commonly used credential repositories that can be found over a Grid.

4.1 Introduction to Globus Toolkit 4.0

Distributed systems require loosely coupled interactions to support an expanding and robust environment. Individual resource addition, maintenance and manageability should be possible without disruption in the overall infrastructure. To achieve sharing and interoperability of heterogeneous resource standards and common interfaces have to be used.

To meet all these requirements the Globus Toolkit 4.0, relies on a Web service oriented architecture leveraging current web service standards for building interoperable distributed systems. GT4.0 uses Web services protocols, mechanisms and specifications for standardizing service interfaces, messages exchanged, security, addressing, monitoring and resource discovery. GT4.0 is built on the following web service specifications:

SOAP [25] version 1.2 specifies how information carrying structured and typed data can be packaged in a message for being exchanged between peers in a decentralized, distributed environment. SOAP is intended to be independent of the transport protocol but commonly it is conveyed over HTTP.

WSDL [3] version 1.1 describes a web service as a collection of abstract operations (actions supported by the service), abstract messages (typed data being exchanged for operation calls) and bindings (how abstract messages and operations are bound to a specific protocol). GT4.0 uses WSDL in conjunction with SOAP for service description and message exchange.

WS-Security and WS-SecureConversation [32, 55] deal with the protection of the information exchanged over the network at the SOAP message level.

WS-Trust [4] defines extensions to WS-Security for requesting and issuing security tokens, and to broker trust relationships. WS-Trust is used for delegating user rights to other entities.

WS-Addressing [53] defines transport neutral mechanisms to address Web services. It defines XML elements to identify Web service end-points and to secure end-to-end endpoint identification in messages.

WS Resource Framework (WSRF) [47] defines how a pairing of a web service and a stateful resource can be realized in order to model and access the state associated with a web service. WS Resource Framework defines the coupling of a web service and a resource in WS-Resource specification, the resource properties (views of the state of a resource at a certain time) and how these properties are queried and updated in WS-ResourceProperties, the management of the resource lifetime in WS-ResourceLifetime, the extension of web services and resources to groups in WS-ServiceGroup, and the faults that can appear during web service interaction in WS-BaseFaults.

WS-Notifications [54] family of specifications defines the terms in which publish subscribe methods can be used by web services. WS-BaseNotification [2] specifies how notifications can be received by clients registered with a provider. WS-Topics [48] deal with notifications delivered based on the topic of interest of the client.

In addition other specifications like Security Assertion Markup Language (SAML) [41] for releasing assertion of user rights and eXtensible Access Control Markup

Language (XACML) [14] for policy specification are partially supported and planned for extensive use in future authorization mechanism.

In conformity with all these specifications Globus Toolkit 4.0 includes the following Web services: *Globus Resource Allocation Manager Web service* (WS-GRAM) for job/execution management, *Reliable File Transfer Web service* (RFT) for data transfers, *Delegation Service* for rights delegation, *Monitoring and Discovery Systems* (MDS) for data collection, query/subscription for data and action triggering based on data collected, *Community Authorization Service* (CAS) for releasing access rights managed at VO level, *OGSA Data Access and Integration* (OGSA-DAI) for accessing and integrating data resources such as files, relational and XML databases into Grids and *Globus Teleoperations Control Protocol* for controlling scientific instruments.

Other service are available with GT4.0 yet not offering a Web service interface: *GridFTP* for reliable file transfer, *Replica Location Service* for distributed file copies location, and *MyProxy Online Credential Repository* for storing user certificates. Additionally pre-Web service compliant implementations of GRAM and MDS-Index are available for continuous support of legacy applications.

Globus Toolkit 4.0 hides the WS* specifications implementation in three flavors of containers: GT4 Java Container, GT4 C Container and GT4 Python Container. All service deployed (including the WS compliant services delivered with the container) make use of the container to exchange messages and expose interfaces adhering to these specifications.

Development of a GT4.0 web services for the Java container requires the definition of a WSDL file describing the service, a Java implementation of the service logic (may require use of Globus Toolkit APIs), the service configuration through the Web Service Deployment Descriptor (WSDD file), the generation of a Grid Archive (GAR file) and its deployment in the container.

In the following paragraphs we are going to present some of the functionalities provided by Globus Toolkit 4.0.

Execution Management WS-GRAM is concerned with initiation, execution, monitoring, management and cancellation of jobs submitted to remote resources. WS-GRAM permits running executables (available locally with the resource or submitted over the network) on remote resource, execution of parallel programs across multiple resources, scheduling and coordinate use of distributed resources for achieving a certain computation. WS-GRAM relies on the RFT service for staging files in and out resources and Delegation Service for delegating rights to the job submitted so that it can access other resources. WS-GRAM can use local schedulers like Condor [12], PBS [37], Torque [44] for job execution over a cluster or can be used by meta schedulers like Condor-G [11] and MPICH-

G2 [29] for submitting jobs to distributed resources. GT 4.0 provides to clients standard web service interfaces through which jobs can be submitted, monitored or canceled on remote computers.

Data Management GT4.0 relies on several services for location, transfer and availability of data to Grids. GridFTP offers high-performance, secure and reliable data transfers over high-bandwidth wide area networks. It supports striping (use of multiple resources for retrieving a single file), third-party transfers (allowing a client to mediate a transfer between two remote servers), partial file access (files can be read while being retrieved), large file support (file sizes, lengths and offsets represented on 64 bits), and parallel transfers (multiple stream between two parties involved in the file transfer). RFT Service is a Web service interface for coordinating third party transfers, deletion of files and directories using GridFTP. RLS is a registry which can be distributed over different sites keeping track of file copies spread over physical resources. Each file (and copies) registered with RLS would have a unique identifier based on which copies can be located either locally or remotely. Data Replication Service is built upon RLS and RFT its functionality being to interrogate RLS for file location and then create a transfer request to RFT.

OGSA-DAI is a data service framework for accessing and integrating data files, relational and XML databases into Grids. OGSA-DAI provides through a web service compliant interface: SQL query on relational resources, evaluation of XPath statements on XML collections, local transformation of data coming out of a data resource to avoid unnecessary data movement and support for metadata about data and data resources, .

Monitoring and Discovery MDS mechanisms deal with gathering information regarding resources and making it available to interested parties. These enables discovery of resources and monitoring of resource state. MDS-Index Service is responsible with indexing information retrieved by various means and its publishing as resource properties. Indexes can be organized hierarchically in order to aggregate data at several levels. Index entries have a limited lifetime and if not refreshed, they are automatically removed. MDS-Index can be easily used by WSRF compliant Web services which need only to present information relevant for indexing as resource properties. WS-Index can either pull resource properties using WSRF standard interface or can register for resource property changes with notifications pushed to the service through the use of WS-Notifications. MDS-Trigger collects data and allows clients to specify XPath queries against this information. If such a query is satisfied, an action can be triggered/program executed (e.g emailing the user about the query matching).

In the following section we are going to focus on the Grid Security Infrastruc-

ture and refer the interested reader to [5] and the Globus Alliance web pages [20] for more information on GT4.0.

4.2 Grid Security Infrastructure

Grid Security Infrastructure (GSI) has been motivated by the need of secure communication between entities over the Grid. GSI provide integrity protection and confidentiality for sensitive information passed over the network as well as a mean of employing security mechanisms across different organizations. In this section we are going to summarize the characteristics of the GSI currently supported in GT4.0.

At the basis of GSI stays public key cryptography employed over a Public Key Infrastructure (PKI). In PKI each entity is associated with a key pair. Data can be encrypted using any of the keys and decrypted with the other. Entities make publically available one of the keys which becomes the public key. The other key is securely and privately stored by the entity and is known as the private key. By using the private key, the holder can encrypt messages and send them over a network. Such messages can be decrypted only with the holder public key. If a counterpart can decrypt such a message with the public key associated with an entity, it is certain that no other entity could have sent that message (the private key is assumed to be securely kept by the entity holding it). Being able to decrypt the message with the public key of a certain entity, authenticates to the receiver that entity as being the source of the message.

Public and private keys can be used for digitally signing of messages. Signed information assures the recipient that no tampering of the data occurred since its transmission. For signing data a hash value (a unique, small size identifier) is computed over the data intended for sending. The hash value is encrypted with the private key of the entity, attached and sent with the data on which the hash was computed. At the receiving part the hash value is re-computed over the received data (the algorithm for computing the hash value is know at both sides) and the signed hash received is decrypted with the sender public key. If the re-computed and the decrypted hash match than the data must be in the same form as it has been sent.

Private and public keys have to be bounded to a certain identity in order to identify the user or the service using them. For this matter PKI uses certificates (figure 4.1). Certificates contain the following information: a distinguished name (DN) which uniquely identifies the entity holding the certificate, the public key belonging to the DN identified in the certificate, the DN of the Certificate Authority (CA) signing the certificate and thus attesting the relationship between the DN and the public key of the certificate, the CA digital signature and an expiration

date. Important to observe is that a trusted CA is used to certify the pairing DN - public key in the certificate. The CA guarantees that the two belong together and signs the certificate so that it can not be tampered. If the CA own certificate is trusted then the user certificate can be trusted.

GSI uses certificates represented in X.509 End Entity Certificate format (check appendix B.1 for an example). X.509 End Entity Entity Certificates can be obtained from a third party Certificate Authority (CA) or can be obtained from a “Simple CA” available with GT4.0 which can be installed for the local domain.

Authentication in GSI requires each party to trust the other part CA, and be able to prove the ownership of its certificate. Authentication of two parties starts with one party disclosing its certificate and being required to encrypt some random data generated by the second party. The encrypted data is decrypted at the second party (using the public key in the certificate already presented by the first party), and if it matches the initial data, authentication is achieved. The same process is repeated for having the second party authenticated as well.

The private key of a certificate is usually stored encrypted with a password, in a file residing on the local file system. This is done to prevent the use of the private key in case the file containing it is stolen (an unauthorized entity having the private key and the associated publically available certificate can easily impersonate the rightful holder of the certificate).

As we have seen the authentication process involves data encryption through the use of the private key. Each time authentication is required the user has to decrypt his private key by entering the protecting password. In a Grid environment, where a user program might access a large number of resources, typing this password each time access is attempted might not be a very comfortable approach, even more thinking that resources may be accessed at moments difficult to predict.

To solve this problem GSI make use of a delegation mechanism providing users with X.509 Proxy Certificates [50]. X.509 Proxy Certificates (appendix B.2) are short lived certificates generated with their associated private key by the user, and signed with his X.509 End Entity Certificate. If a proxy certificate private key is compromised, due to the certificate’s short lifetime (several hours) the harm that can be inferred is minimized. The proxy certificate private key can be protected only by local file system permission, in this way user applications are able to use it whenever required without user input. If the proxy certificate expires, the user can generate a new proxy certificate with a new private key (it is possible to automatize this process so that long running jobs can work without user intervention). Proxy certificates are also used for delegating user rights to other agents/resources requiring to contact other resources on user behalf. The process is the same with the delegated part generating a proxy certificate (and keeping private the private key) and submitting it for signing to the delegating part. A proxy certificate can be signed by another proxy certificate (agents holding a delegated

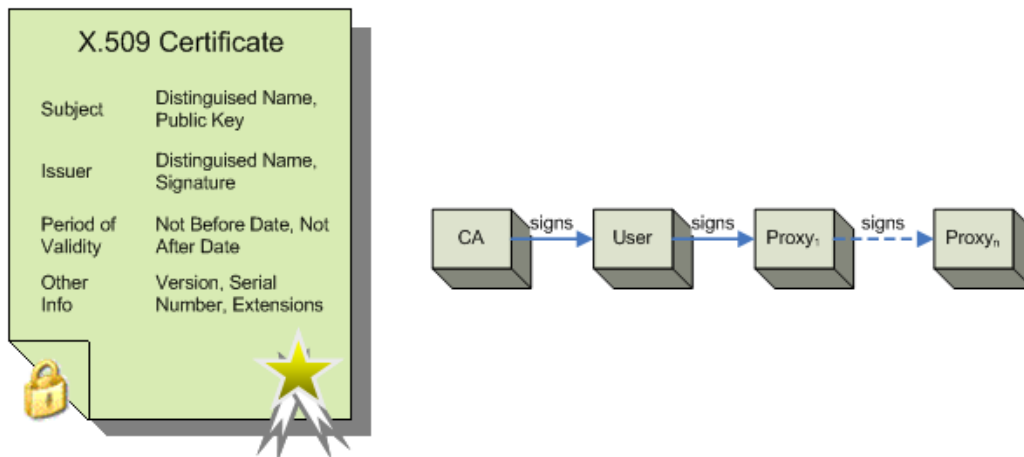


Figure 4.1: X.509 Certificate and Proxy Chain Certificate Validation

certificate may use it at their turn to delegate certificates to other agents on user's behalf) or an end entity certificate, with the process of authentication consisting of the pushing of the whole chain of certificates (having at root the user end entity certificate). The entire chain can be validated (by verifying with the public key of each certificate the signature on the next one in the chain - see figure 4.1) and authentication achieved (provided if the user end entity certificate is signed by a trusted CA).

GT 4.0 delivers message protection through two mechanism: transport level security (for transporting SOAP messages through secure channels) and message level security (by signing/encrypting parts of the SOAP messages). Transport level security is achieved by using X.509 certificates for authenticating parties and establishing a secure connection between the two over Transport Layer Security (TLS). Message level security is supported via the use of WS-Security standard or WS-SecureConversation specification. If WS-Security is used in combination with X.509 certificates (proxy or end entity) protection of individual messages can be achieved by having parts of them signed or encrypted (the information is encrypted using the destination entity public key, and therefore, only the destination party is able to decrypt the data as it is the only one holding the private key). For WS-Security a preexisting context between the two entities is not required. If WS-SecureConversation is used the keys of the two entities are used to establish a security context (containing a shared key which can be used for both encryption and decryption) allowing protection of messages with less overhead than using WS-Security.

Authorization. GT4.0 supports an Authorization Framework [24] for enforcing authorization on both client and service side. On the service side, autho-

Authorization mechanisms rely on a configured chain of Policy Decision Points (PDPs) to determine if authorization should be granted or denied for a client invocation. The framework supports user development and plugging of customizable PDPs for service authorization. The decision regarding authorization is made based on the conjunction of all PDPs decisions, that is authorization is granted only if all PDPs have returned a permit decision. PDPs logic can be implemented based on the client DN, the service accessed and the operation invoked.

Security descriptors are used for configuring authentication and authorization mechanism. Such security descriptors can be defined at the container, service or resource (only for WSRF compliant services) level. Clients can also use security descriptors for specifying how the accessed service should be authenticated and authorized.

Through the use of security descriptors, the container and each service can be configured to use different credentials. Credentials can be either a proxy certificate or an end entity certificate and the associated private key. Authentication/encryption methods can be specified at service operation level. The available options are: GSISecureMessage and GSISecureConversation for message level security and GSITransport for transport level security. Messages can be protected for integrity (signed) or privacy (encrypted).

On the service side the authorization mechanisms available with GT4.0 rely on PDPs part of the distribution. These PDPs are configured through the security descriptor of a service/container. The options are: self - identity of the service and client are expected to be the same, gridmap - the identity of the client must be mapped to a local user account in a gridmap file, identity - a certain identity of the client is expected, host - a certain host name is expected of the entity requiring access, samlCallout - a SAML authorization callout to an external OGSA Authorization compliant service [33], userName - username and password based authorization (identity refers to the DN present in client certificate). On the client side the service authorization options are: self, host or identity.

4.3 Credential Repositories

Credential repositories offer persistent storage and allow dynamic retrieval of user credentials either by the user himself or by entities delegated to act on his behalf. They provide a single location for user credential keeping, and are expected to be administrated and run on resource specially protected against compromise.

MyProxy Online Credential Repository [7] Client mobility may be desired over the distributed system used. The nature of a client work may require frequent change of the computer from where Grid applications are launched. Manually copying of keystores between hosts over the network, might inflict a number of

security breaches either due to client negligence or usage of insecure/comprised systems. The MyProxy Online Credential Repository addresses these issues by providing a repository for secure credential storage and retrieval over the network. MyProxy Repositories are intended to be run on well-protected hosts, similar to a Kerberos KDC. MyProxy Repositories allow storage of client credentials in the form of X.509 End Entity Certificates or X.509 Proxy Certificates and later retrieval of delegated X.509 Proxy Certificates with short lifetime. The protocol used by the MyProxy Repository is the same as in GSI delegation capability with delegated certificates generated locally and signed by the private key of a user certificate stored with the repository. Private keys are not transferred over the network thus minimizing risk of theft. Clients have user accounts and passwords set with the MyProxy Credential Repository and can store credentials and encrypt the private keys with their own supplied passwords. Only if a correct password is provided the private key of the delegating credential on the MyProxy Repository can be decrypted and a valid X.509 Proxy Certificate delegated. Clients can also specify simple regular expressions protecting stored credentials and requiring a check of the requester DN against the specified expression.

In many Grid deployments user job submission is handled through Grid Portals (web interfaces to the Grid environment). Such Grid Portals need credentials for submitting jobs on user behalf. Provided with the password protecting a previously stored credential on a MyProxy Server, a Grid Portal can retrieve a delegated X.509 Proxy Certificate and use it for client job submission.

MyProxy supports also renewal of user credentials for long running jobs. A scheduler (e.g Condor-G) can monitor job credentials and when near expiration require renewal from a MyProxy Credential Repository. In this situation, if the DN of the service running the scheduler matches the user supplied regular expression and the service also makes proof of a previous delegated credential the job credential is renewed.

Administrators can specify server wide access lists for retrieving and renewing credentials from a MyProxy Repository. In these lists the DN of the expected requesters have to be entered. Entities requiring a credential have to pass an extra check with their DN verified against the administrators lists.

Community Authorization Service (CAS) [35] CAS permits definition and retrieval of authorization policies setup at the VO level. Often organizations form a VO and establish policies that govern rights over the shared resources. CAS provides a common point for the VO members to establish consistent policies, administrate, maintain and release of these policies to sites. Usually a CAS server is installed per VO with an identity known by all VO members of which some are designated to run and administer the server. Resource providers set at resource level the access rights granted to the VO as whole. A user interested in accessing a resource retrieves from the CAS server a SAML signed assertion containing the

rights entitled by the VO to the user. The assertion contains the identity of the user and information about where access is granted and what actions are permitted, all this being signed by the CAS (it may contain all user rights or only those relating to the resource where access is requested). The user packages the assertion in a proxy certificate and presents it to the resource. At the resource, the assertion is verified (the identity of the CAS server is known/trusted and the policy embedded is in conformity with the rights given by the resource to the VO). The resource should enforce the rights at the intersection of the rights entitled by the CAS to the client (contained in the SAML assertion presented), those locally granted to the VO as a whole, and those configured for the identity of the user.

Chapter 5

Grid Scenario

In this chapter we are going to motivate our work through one scenario illustrating how current Grids handle authentication and authorization, job submission and credential management. We consider that the middleware at hand consists of the latest version 4.0 of the Globus Toolkit providing all the mechanisms available with this distribution. By the end of this chapter we will summarize the limitations and assumptions of current approaches preparing the setting for our proposed enhancements.

5.1 Submitting a job over the Grid

Grid environments provide the middleware needed for collective job solving through dynamic creation and usage of large and remote collections of resources. This comes to numerous and distinctly administrated domains providing resources for input data retrieving, job execution, monitoring, and output data mass-storage saving. In order for access to be granted at each domain the job has to secure and provide enough credentials for authentication and authorization. While authentication along with single sign-on have found an answer in the client delegation of X.509 Proxy Certificates to the job being submitted using an end user X.509 Certificate, authorization mechanisms are still mainly identity based.

Let us consider the following illustrative scenario depicted in figure 5.1. A group of scientists at the Computer Science and Engineering Department of University “Politehnca” of Bucharest (U.P.B.) would like to obtain some data regarding oceanic water waves in order to develop signalling instruments for tsunami hazards avoidance. Fortunately, the university and the Navy Institute have an agreement, allowing U.P.B. members to use any of the Institute scientific instruments as long as it is not already in use, and U.P.B. has not exceeded a number of monthly allocated hours using institute resources. Both the university and the

Navy Institute support GlobusToolkit 4.0 [20] providing the middleware the scientists need to collect the required data from a Wave Tank available at the institute site.

Sometime ago, Alice, the leader of the group of scientists, using her U.P.B. CA issued X.509 End Entity Certificate, has delegated to the U.P.B. MyProxy Repository an X.509 Proxy Certificate. Using this certificate the repository can sign other proxy certificates and deliver them to entities requiring to act on Alice behalf. A requester can retrieve a proxy certificate if he can provide the username and password protecting the delegating credential on the repository. Also a requester can renew a proxy certificate if he can make proof of a previous delegated certificate and have his own DN match a simple regular expressions setup for the proxy certificate on the MyProxy Repository.

Alice prepares a job and submits it over the Grid Infrastructure. First, she logs into the U.P.B. Grid Portal and instructs it to retrieve for the job, on her behalf, a delegated X.509 Proxy Certificate from the U.P.B. MyProxy Repository [31]. The Grid Portal authenticates using its own credential to the MyProxy Repository and since the DN of the certificate presented is in the allowed retrieve list of the repository, permission is granted and a request is made for a delegated certificate. Together with the request for the proxy certificate, the Grid Portal sends also the username and password provided by Alice. The MyProxy Repository verifies them and delegates a proxy certificate for the job, on Alice behalf. The delegated X509 Proxy Certificate is generated by the Grid Portal and signed by the certificate on the MyProxy Repository. The job ends up with a chain of certificates containing Alice X.509 End Entity Certificate, the X.509 Proxy Certificate previously delegated to the MyProxy Repository and the X.509 Proxy Certificate delegated to the job. This chain of certificates identifies the job as pertaining to Alice.

The job is sent to the U.P.B. HPC Center Linux cluster. Here the job will retrieve the input data needed to setup the Wave Tank, make computational intensive operations to refine and error correct the output of the Wave Tank, and store the results. First the Linux cluster has to authenticate and authorize the job submitted. The certificate chain provided with the job has at its root Alice X.509 End Entity Certificate signed by the U.P.B. Certificate Authority(CA). The CA is trusted at the HPC Center Linux Cluster thus the job gets authenticated. Authorization is also granted as a grid map-file entry maps Alice's DN to a local user account permitting the job to start its work with the privileges of the local user.

As the job needs additional resources a new X509 Proxy Certificate is delegated by the Grid Portal to the U.P.B. HPC Center Linux Cluster. Using this additional proxy certificate the job can authenticate and gain access to other Grid resources. First it will contact the U.P.B. Reliable File Transfer Service (RFT) for retrieving the input data. Authentication and authorization are satisfied in the

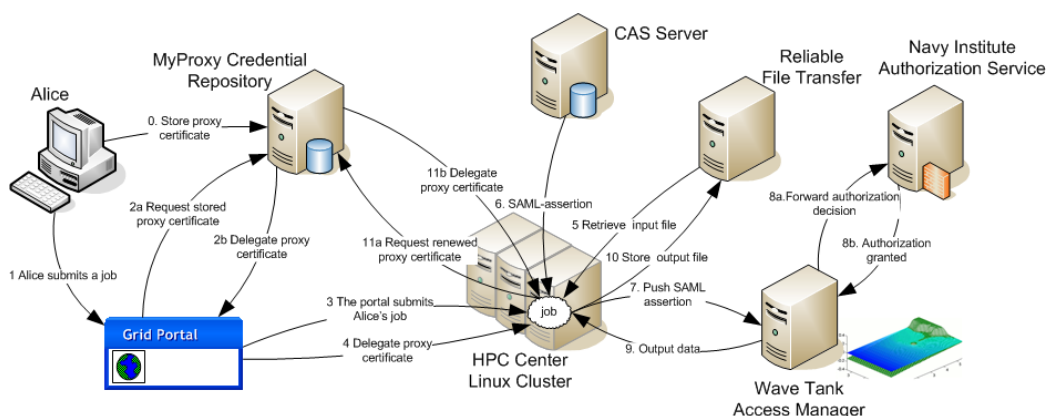


Figure 5.1: Simple Grid Scenario

same way, based on Alice DN (the chain of certificates presented to the RFT is one certificate longer containing also the one just delegated to the Linux Cluster).

Alice has already been informed that the Navy Institute requires a proof of her University affiliation, so she has programmed the job to contact the U.P.B. CAS Server and retrieve a statement attesting her involvement with the university. Again a round of authentication and identity based authorization takes place and the job retrieves a SAML assertion attesting Alice university affiliation. The job is hardcoded to create a new X.509 Proxy Certificate containing the previous obtained assertion. This proxy certificate is going to be used for gaining access at the Navy Institute Wave Tank site.

The Wave Tank forwards the authorization decision as a SAML authorization callout to the Navy Institute Authorization Service, able to obtain and verify the assertion in the certificate provided. Assuming that all the other wave tank local requirements are fulfilled (the wave tank is not in use and U.P.B. has not exceeded its allocated hours) data starts to arrive at the U.P.B. HPC Center Linux Cluster.

Unfortunately the task proves to be a long one and the credential delegated to the Linux Cluster is to expire. The Linux Cluster has to renew its credential, so it contacts the U.P.B. MyProxy Repository requiring for a renewed credential. The distinguish name in U.P.B. HPC Center Linux Cluster own certificate is in the renew allowed list of the MyProxy Repository and matches Alice specified regular expression for renewing requests. The Linux Cluster proves also the ownership of a previous delegated credential, so a new credential gets delegated to the Linux Cluster on Alice behalf.

Finally when the data has been corrected and refined the Linux Cluster contacts the U.P.B. RFT Service and after another round of authentication and authorization the final data is available to Alice and her group.

5.2 Limitations and Assumptions

As we have observed during the interactions depicted in our scenario authorization decisions were made based on user identity. The user had already an account set at the remote locations with a grid map file entry mapping his DN to his local account. Identities were already known by the MyProxy Server as it checked requester DN against the allowed retrieve and renew lists. Setting identity based access control lists requires site administrators to keep track of clients allowed to request services, a task difficult to handle as user mappings may be available due to user associations with certain organizations and projects. Once such relations cease to exist resource administrators have to remove these entries so access is no longer allowed. On the other hand as soon as new users require to utilize the Grid infrastructure new mappings have to be entered to accommodate the new identities.

One of the assumptions in the above scenario, was that the U.P.B. CA is trusted at each location the job is requiring access (U.P.B. resources and Navy Institute Wave Tank). There has been no intermediary step for the job and the contacted resources to argue about trusted authorities. The CA trusted at each site is expected to be known by the client and an appropriate certificate to be provided.

In these conditions, mapping client identities to local accounts raises serious scalability problems due to the large number of potential users, even more taking into account the difficult requirement of having a single trusted Certificate Authority (which is hard enough to have within one Grid and not feasible at all when trying to integrate different Grids). Property based certificates have started to be used (PRIMA [27], VOMS [6], CAS [35] and X.509 attribute certificates [16]) so clients can be mapped to local accounts based on their attributes. Still there is no standard interface for using these certificates and their integration is achieved for individual Grids and not in a large scale Grid environment. Nevertheless, the trusted Certificate Authorities are advertised using alternative solutions, so potential users know in advance where to request certificates from. Moreover, there is no query for resource access requirements and no negotiation for authorization granting. The authorization decision is a one step process in which the client is assumed to be aware and be able to satisfy the service requirements. Such an approach makes identity management hard to cope and an automated authorization scheme infeasible to implement.

In our example we assumed that the university administrators have carefully set up the available resources, so that the university students and staff may put them to the best use. This may be indeed not an issue for groups associated with a certain institution but for the future predicted large scale Grid each of the computational resources used by a job may be under a different authority. Entities with no previous interaction may have to communicate and is unrealistic to assume that

in a large scale Grid, identity based authorization may bring the versatility needed in such an environment. Consider the following example: a job which spawns a large number of sub jobs each having to find the best resources available on the Grid to be executed. In such a setting the sub jobs have to find a way to inform themselves autonomously of the authentication/authorization requirements at each identified resource and find automatically if they can meet or not these requirements.

Some of the deployed Grids base their authorization on assertions issued by services established at the virtual organization level (e.g. CAS [35]). Grid service providers might find uncomfortable to rely only on remote assertions mainly in the case of a larger Grid environment. The interest in using, advertising and enforcing policies may shift to the local domain so that the administrators have fine grained access control over the resource they provide. Even if authorization decisions might be based on statements made by third parties at the VO level, still the authorization decision might involve local information and state (in our example the wave tank should not be in use and the University use of instruments should not have exceeded a certain limit).

On the other hand the client might have his own concerns regarding the resources accessed and the credentials he has to provide. He may ask the service to provide additional credentials so that a trust relationship can be established or be interested in protecting his credentials through a set of policies imposed on the service side and only if satisfied have the protected credentials revealed.

A Grid-scale application is expected to require a large number of resources under diverse local administrations making the user face an additional problem: keeping track of each of his credentials in relation to the location in which they are needed. This holds to the same extent, for the submitted job due to the difficulty in predicting job needs prior to its execution. At the moment a certain credential is required the user might not be available, while providing the job with all user credentials might imply disclosing sensitive information which user may not be comfortable to. The issue the user/job might encounter is providing the right/required credential to the Grid service it tries to access.

We assumed in our example that Alice already knew that she had to provide a credential attesting her university membership, but in a large scale and pervasive Grid environment such requirements may be impossible to predict. In addition, currently there is no unique credential issuer which is widely accepted as trustworthy by the Grid community and no available mean for the service to advertise the credential issuers it trusts. Clients have to keep track of the Certificate Authorities trusted at the remote location in order to provide a valid credential and also know in advance what kind of additional credentials are needed. Users may not be granted access to certain resources because not enough credentials have been submitted even if such credentials were available.

The authorization options previously discussed may be limiting for dynamic Grid environments. Globus Toolkit 4.0 options for client authorization are: none, username, host, identity, grid map-file or a callout to a SAML compliant authorization service. For the client service authorization options are: none, self, host or identity. The interactions using each of these options consist of a one step decision which is either access granted or refused. There is no provided API for the service to advertise extra requirements such that the client might query the service and make his request with all the necessary credentials. Furthermore, the service can not dynamically inform the client about the third party credential issuers that are trusted.

Although it seems that the scenario (depicted in figure 5.1) fits Alice's and the resource owners' needs, the assumptions for Alice job to succeed are:

- One credential/chain of certificates for all. A job is submitted together with a proxy certificate signed by a Certificate Authority (CA) which is further used to authenticate to other resources. This assumes that all resources must trust such a CA.
- Identity Based Authorization. Resources, where a job is allowed access, have to know in advance the identity of the certificate.
- Simple Authentication/Authorization. It is based on a one-shot process where the job requests access and the resource grants or denies it.
- Manual Credential Fetching. Users need to find out in advance which credentials are required to access each resource and hardcode their jobs to fetch and give these credentials while authenticating/requiring authorization.

These requirements become liabilities when the Grid grows more complex and the number of resources a job has to access increases. We argue in the following chapters that Grid with support for specifying, advertising and enforcing service level access control policies together with capabilities for automatic fetching of credentials can indeed suit large scale collection of resources, enabling dynamic negotiation for authorization and access granting based on parties properties. We will show in the following chapters how such features can be implemented on top of Globus Toolkit 4.0.

Chapter 6

Policy Based Negotiation for Trust Establishment

Distributed environments like World Wide Web, B2B systems, Grid are built with the assumption that service providers and consumers are known to each other. In common scenarios before allowing access to (possibly) sensitive resources, trust relations are established among entities by having clients pass a registration phase with the resource they try to access. This phase may involve the creation of an account, profile, addition of the user identity to some kind of access control lists or some offline contacts specifying exactly who and what kind of service provides to which consumers. Passing through this process usually requires users to provide personal information which they may not be comfortable to disclose (e.g. home address, phone number, position or credit card number). This becomes a real issue as users usually have no mean of verifying if the service provider is trustworthy for disclosing such sensitive information. In this chapter we are going to describe how entities can build a trust relation through semantic annotations, rule-oriented access control policies and automated trust negotiation.

6.1 Trust Negotiation

Trust relations are constructed based on a set of digital credentials (e.g. certificates, signed assertions, capabilities or roles) which are disclosed by two entities to prove certain properties they poses. Digital credentials attest a certain quality of the holder. They are issued and signed by a credential issuer and may bind the identity of the holder to a public key, attest a certain attribute of him or a relationship of the holder with regard to another entity. Credentials are signed by the private key of the issuer and thus they can not be tampered by malicious users. Credentials carry the public key of their holder. This means that if one entity dis-

closes a credential and is able to sign a challenge with the private key associated to the public key present in the credential then that entity is the rightful holder of the credential (which may confer him a property).

The credential may include sensitive information (e.g. holder roles, capabilities ...) so users may have their own requirements with regard to the entity to which they release this information. A more adequate approach would be that distributed environments treat clients and service providers equally, similarly to P2P systems, both having the ability of imposing restrictions on the other side.

During a trust negotiation process [51] trust is established gradually through disclosure of credentials and requests for credentials in an iterative and bilateral process. The requests for credentials are in fact authorization policies having to be satisfied. Trust reached through a negotiation between one client and a resource may be regarded as an authorization decision to access or not the resource. This approach distinguishes from the identity based access control in the following regards:

- Trust is established between previously unknown parties based on their properties proved through credential disclosure.
- Service providers can define policies to express which credentials are required for access granting.
- Both service providers and consumers can define policies to protect their credentials.
- Authorization is no more a one step decision, rather being established incrementally through a sequence of mutual requirements and disclosures of credentials.
- In the authorization process may be involved more than two entities and their trusted credential issuer, as requirements for a credential may determine a new negotiation with another entity, which at its turn may trigger another and so on.

Trust negotiation is triggered when one party requests to access a resource owned by another party. The goal of a trust negotiation is to find a sequence of credentials (C_1, \dots, C_k, R) where R is the resource where access is attempted, such that when credential C_i is disclosed, its access control policy has been satisfied by credentials disclosed earlier in the sequence, or to determine that no such credential disclosure sequence exists.

We are going to rely on PeerTrust [19] language and project [36] and show how trust negotiation can extend the Grid Security Infrastructure with policy based

authorization mechanisms and automatic credential gathering. Resources using PeerTrust are considered items on the Semantic Web with properties represented as RDF properties, thus facilitating a direct integration with the Semantic Grid. Nevertheless, Grid environments might make use of PeerTrust due to its semantic independence of sides' property representations, even more taking into account that current Grid certificates are able to carry properties in different formats. We are going to describe in the following section the PeerTrust syntax.

6.2 PeerTrust Language

PeerTrust is a rule-based language using definite Horn rules for expressing policies. Rules have the following form:

$$\text{lit}_0 \leftarrow \text{lit}_1, \text{lit}_2, \dots, \text{lit}_n.$$

where lit_i is a positive predicate $P(a_1, a_2, \dots, a_k)$ with a_i its arguments.

The head of the rule is lit_0 while its body is formed of lit_i . The meaning of such a rule is that if all the predicates in the body of the rule (all lit_i) are true then the head of the rule (lit_0) evaluates to true. Definite Horn rules are used in logic programming and stay at the basis of the rule layer of the Semantic Web being specified in the RuleML effort [22, 23].

Authorities and Requesters The PeerTrust language supports delegation of evaluation to another party, meaning the identification of an entity able to evaluate a certain predicate. Any literal lit_i can be extended with an *Issuer* argument using the @ operator

$$\text{lit}_i @ \text{Issuer}$$

identifies who is authoritative to evaluate lit_i (and possibly who would be asked to evaluate lit_i if no information is locally available).

The *Issuer* argument can be a nested term containing a sequence of issuers, which are evaluated starting at the outermost layer. For example:

$$\text{lit}_i @ \text{Issuer}_k @ \text{Issuer}_p$$

denotes that $\text{lit}_i @ \text{Issuer}_k$ must be proven by Issuer_p , which if unable to make such a proof, would forward the evaluation of lit_i to Issuer_k .

The answer to a certain query may depend on the entity who required the evaluation of the query. For that a *Requester* argument, identified by \$ operator, is added to the literal. The form of a literal may be:

$lit_i @ Issuer \$ Requester$

where *Requester* is instantiated with the entity asking for the evaluation of $lit_i @ Issuer$.

Local and Signed Rules The policies set for a resource are formed by definite Horn rules with additional arguments for Issuers and Requester. Each resource may define its own local rules for access granting and credential protection, rules that may have meaning only for the resource they act upon. Signed rules represent what is considered to be true by the entity signing them (presumably a third party). They can be used along with local rules and credentials to evaluate the state of a predicate.

$lit_i @ Issuer$
signedBy [Issuer].

proves that $lit_i @ Issuer$ is true according to issuer judgement and that the issuer attests this, by digitally signing the rule. This rule may be represented externally to PeerTrust syntax as a certificate or signed assertion possibly having also an expiration period.

A rule of the form:

$lit_i @ Issuer_i \leftarrow lit_k @ Issuer_k$
signedBy [Issuer_i].

indicate that the head ($lit_i @ "Issuer_i"$) of the rule is signed (signedBy ["Issuer_i"]) and it is disclosed only if the body of the rule is satisfied ($lit_k @ Issuer_k$). This may indicate that a credential (encoded as the head of the rule) is to be disclosed only if the body of the rule is fulfilled.

Signed rules can represent delegation of authority:

$lit_i @ Issuer_k \leftarrow$
signeBy[Issuer_k]
 $lit_i @ Issuer_p$

states that Issuer_k delegates its authority to evaluate lit_i to Issuer_p. Such rules may be cached and used to prove $lit_i @ Issuer_k$ with a positive evaluation of lit_i with Issuer_p.

Public and Private Predicates Predicates can be public or private. Only the public ones are able to be queried by external parties. A public predicate may express access granting if evaluated to true. Such a predicate may be the head of a rule and when queried, it would require the satisfaction of the predicates in the body of the rule which may be either private (locally evaluated) or public (remotely evaluated by third parties or by the entity requiring access). In this setting if the evaluation of the body of the rule is true, the head predicate becomes true and a granted authorization decision is drawn. Private predicates may be derivable only at the resource site and may for example verify local conditions about which the requester is not and should not be informed of.

Guards The order of evaluation of the literals in the body of a rule is not specified. Because literals may be delegated to other parties, there is no need to sequential their order of evaluation which can be done in parallel. However there are situations in which certain predicates must be satisfied, before proceeding with the evaluation of the rest of the predicates. In order to allow this, we define another operator called *guard* and represented as $|$. Guards introduce order over the set of predicates. For example:

$$\text{lit}_i \leftarrow \text{lit}_j @ \text{Issuer}_m | \text{lit}_k @ \text{Issuer}_p.$$

imposes first the evaluation of $\text{lit}_j @ \text{Issuer}_m$ and only if it succeeds $\text{lit}_k @ \text{Issuer}_p$ would be evaluated. lit_i is evaluated to true only if first $\text{lit}_j @ \text{Issuer}_m$ was proven to be true and then $\text{lit}_k @ \text{Issuer}_p$ evaluated to true.

The guard (literals before “ $|$ ” operator) in the body of a rule may contain a set of predicates. All of them have to be evaluated and if all are true, the rest of the predicates are evaluated (those after the operator “ $|$ ”).

6.3 Example

We will make the convention of representing instantiated rule arguments between double quotes(“”) and uninstantiated arguments by writing them in upper case.

Let’s consider a simple example to show how PeerTrust rules can be used for authorization (a more complex scenario is presented in chapter 8):

Bob is a student at University of Hannover. He would like to make a large matrix computation and needs a powerful computer able to handle such a request. Bob knows that L3S Research Center of University of Hannover has a cluster which exposes a Grid service for matrix operations. Still he does not know what kind of credentials he has to provide in order to be allowed to use the Grid service

operations. Bob (actually his grid client program) sends a request to the Grid service asking about its requirements for allowing the multiplication operation. Bob would send a rule like *request("multiply")* requiring its evaluation.

The Grid service policies for the multiplication operation are:

```
[Grid Service L3S Research Center]
request("multiply") $ Requester ←
    student(Requester) @ "UniHannover" @ Requester |
    check(Requester).
request("multiply") $ Requester ←
    employee(Requester) @ "L3S" @ Requester.
request("multiply") $ Requester ←
    member(Requester,"D-Grid") @ "D-Grid" @ Requester.
check(Requester) ←
    researchAssistant(Requester) @ "L3S" @ Requester.
check(Requester) ←
    studentID(Number) @ "UniHannover" @ Requester |
    verify(Number,"FEECS")@ "FEECS".

registeredUniResource("L3S") @ "UniHannover"
    signedBy ["UniHannover"].
```

In order to allow access to the multiply operation the Grid service demands the requester to prove he is student at University of Hannover, an employee at L3S Research Center or a member of the D-Grid project. The policy rules *student(Requester) @ "UniHannover" @ Requester*, *employee(Requester) @ "L3S" @ Requester* and *member(Requester,"D-Grid") @ "D-Grid" @ Requester* (in our case *Requester* argument is instantiated to "Bob") are forwarded to the Requester. Bob receives three queries *student("Bob") @ "UniHannover"*, *employee("Bob") @ "L3S"* and *member("Bob","D-Grid") @ "D-Grid"*. Bob is neither an employee at L3S Research Center nor a member of the D-Grid project and does not know how to retrieve such credentials, so he responds with a fail message to each of these queries. However he is a student and knows that University of Hannover has an online credential repository where he could forward the query regarding his student status, but fortunately he already has a cached credential signed by the university attesting he is a student. Let's take a look at Bob's policies:

```
[Bob]:
student("Bob") @ "UniHannover"
    signedBy ["UniHannover"].

studentID("1234") @ "UniHannover" $ Requester ←
```

registeredUniResource(Requester) @ "UniHannover" @ Requester,
signedBy["UniHannover"].

Since Bob's credential proving that he is a student has no protecting policies, the credential is disclosed to the Grid service. Next the Grid Service requires Bob to either prove he is a research assistant at L3S (*researchAssistant("Bob") @ "L3S"*) or provide his student ID (*studentID(Number) @ "UniHannover"*). The Grid service requires Bob student ID to check if Bob is registered with the Faculty of Electrical Engineering and Computer Science (FEECS). The Grid service avoids sending a direct query to Bob about his FEECS attendance in order not to reveal to all potential users that only FEEC students might be allowed access. Bob is not a research assistant at L3S, he does not have this kind of credential and can not obtain one, so a failed message is answered to the *researchAssistant("Bob") @ "L3S"* query.

Nevertheless Bob has a signed credential from University of Hannover containing his student ID, but he is reluctant in revealing this information to other resource than those registered with the university. When asked *studentID(Number) @ "UniHannover"*, Bob requests the Grid service to prove *registeredUniResource("L3S") @ "UniHannover"*. The Grid service has such a credential and no policy protecting it. It discloses this credential and therefore Bob has his policy satisfied and reveals his student ID credential. Bob is a student at FEECS so the negotiation succeeds and Bob is permitted to call the multiply operation against the Grid service on the L3S Research Center Cluster.

Chapter 7

Policies, Rules and the Semantic Grid

The Semantic Grid was introduced in 2001 [39] and regarded as a mean for *achieving easy-to-use, seamless automation, flexible collaborations and computations on a global scale*. Semantic Grid objective is closely related to the Semantic Web aim [52] to *smoothly interconnect personal information management, enterprise application integration, and the global sharing of commercial, scientific and cultural data* by putting on the Web data that *can be shared and processed by automated tools as well as by people*. To synthesize this relation a refined vision of the Semantic Grid [40] was defined: *the Semantic Grid vision is to achieve a high degree of easy-to-use and seamless automation to facilitate flexible collaborations and computations on a global scale, by means of machine-processable knowledge both on and in the Grid*.

7.1 Relation with the Semantic Grid and the Semantic Web

A series of requirements [40] have been identified for the Semantic Grid, all of them claiming knowledge about resources for facilitating transparent usage, composition, security and trust in relation with diverse resources, agreement for resource usage, and autonomous application behavior. We address all these requirements from the perspective of authorization for grid services invocation. The architecture we propose, relies on policies for access granting expressed at resource level, able to be queried and satisfied automatically.

In our design the access requirements of a grid service appear to a client as policies having to be satisfied. Internally the policies are represented as first order logic rules, on which a rule engine makes its decisions. The policy format and lan-

guage we propose has the richness of first order logic rules, being able to express general requirements for access granting. Forward mapping of policies into rules permits a rule engine to ground rule parameters, infer on proofs and queries received, and generate further requirements or decisions regarding access. We place our architecture at the “Rules” and “Logic” layers of the Semantic Web layer-cake as we provide both the rule representation and the logic for an authorization resolution.

At this moment we use the Peertrust language in interactions between parties requiring and granting access. It can accommodate RDF statements about resources either as proofs or facts in the rule engine during the authorization process. Relations of the type “Subject-Predicate-Object” can be easily translated in our syntax as “Predicate(Subject)@Object”. On the other hand we argue that ontologies and ontological agreements relying on the Web Ontology Language (OWL [34]) can be used, so that other languages may be placed on top of our architecture, as long as their vocabulary achieves a similar functionality. Transformations between ontologies would also permit the understanding of statements presented in different languages.

Furthermore we can accustom agreement for resource usage, as the negotiation process builds upon preferences/requirements(in terms of authorization) of both parties. The process is fully automated (based on sides policies) with credential/proofs fetched from third parties and an agreement for authorization inferred and recorded (a trace of grounded parameter rules satisfying the access policies). Based on parties authorization policies, service composition can be achieved as the application or a broker can dynamically negotiate authorization at different sites and check if access can be granted.

All this capabilities would become more apparent in the following chapter, in which we provide an extensive example of how a Grid security infrastructure can be enhanced through the adoption of semantics in authorization policies.

Chapter 8

Distributed Policy Based Negotiation for GRID Services Authorization

In this chapter we are showing how PeerTrust might be integrated to fit a large, loosely coupled, Grid environment. Each service available will be self explanatory with regard to its access control policies, requiring minimum human intervention for access management resuming to setting up policies. Self describing services in terms of authorization requirements bring a major benefit to a heterogeneous Grid environment permitting automatic and autonomous negotiation for access granting based on sides' policies and dynamic credential fetching. Rule-based policy authorization schemes have been identified in [8] as a solution for transparent resource sharing over the Grid. We build our approach on the observations made in [8] and propose also automatic credential fetching. Our work resulted in a functional architecture (described in chapter 9) which we have implemented and tested based on Globus Toolkit 4.0.

8.1 Negotiating Grid Service Access

In our initial example (figure 5.1) we illustrated how different resources may be used in a deployed GlobusToolkit 4.0 Grid environment. We had to deal with only two administrative domains: one administrated by the University "Politehnica" of Bucharest (U.P.B.) and the other by the Navy Institute. All university resources had been setup to easily interoperate and authorize university members. Previous contacts with the Navy Institute had already been established so that its scientific instruments could be addressed by the university. As resources are expensive in terms of cost, maintenance and management, a truly federated environment would place the resources involved in job execution under several administrative domains requiring more complicated access relations, rather than an single agree-

ment between two actors (U.P.B. and the Navy Institute).

In such a scenario setting at each individual location the identities needed for the submitted job to succeed, would indeed be a cumbersome venture as establishing offline contacts for configuring resources might take more time than the job would take to run. In addition the job may have a goal in itself to find the best location for its execution, as prompt results may be expected, making impossible to foretell the resource on which the job would end up. A more suitable scenario for a Grid environment, would be that clients submit jobs and have no further interactions (either online or by hardcoding resources and credential repositories), relying only on their jobs and their defined policies to locate the best resources available, gather credentials and acquire authorization.

Our proposed solution consists of having each grid service advertise its authorization requirements through use of access control policies, each client specify his credential disclosure policies and be able to query resource policies. If such policies are advertised, services and clients can negotiate authorization by increasing iteratively their trust relationship. The process of policy querying and fulfillment can be automated in a self describing environment, achieving the versatility required for a dynamic and widely available Grid.

Let us follow again our initial example. However, we will modify our scenario so that it accommodates the situations highlighted above. We will assume that more administrative domains are involved and each party can advertise its access control policies.

Alice is attending a Grid Conference organized by Global Grid Forum (GGF) in another country. Her laptop has no Grid job submitting capabilities but she thinks of starting the job for collecting the needed data so that her group can continue the experimental work.

Alice can not log into the U.P.B. Grid Portal as for security issues it can only be addressed from the university network. Fortunately the conference provides its own policy enabled Grid Portal, for demonstrative purposes, and Alice thinks to try it out by requesting to submit her job.

The Conference Grid Portal advertised policies are:

[Conference Grid Portal]:

```
allowRequest(MyProxyServer) $ Requester ←  
    validConfRegistration(Name, RegNumber,Requester),  
    proxyAccessData(MyProxyServer,UsrName,Pwd) @ Requester,  
    permitted(MyProxyServer) |  
    retrieveCredential(UsrName,Pwd) @ MyProxyServer.
```

```
validConfRegistration(Name,RegNumber,Requester) ←  
    registered(Name,RegNumber) @ Requester |
```

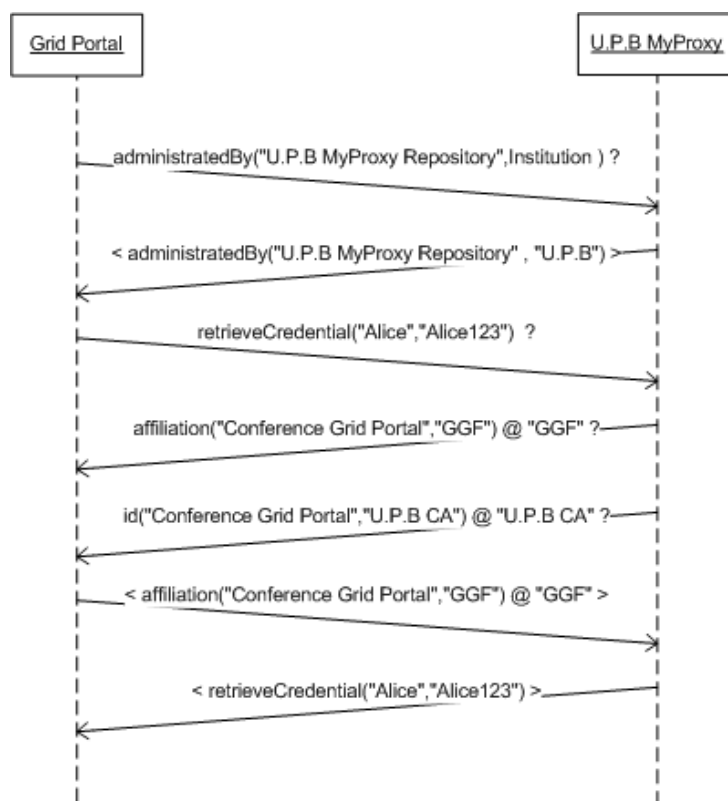


Figure 8.1: Grid Portal MyProxy Server Negotiation

checkReg(Name,RegNumber).

permitted(MyProxyServer) ←
 administratedBy(MyProxyServer,Institution) @ MyProxyServer |
 registeredWithConference(Institution).

affiliation("Conference Grid Portal","GGF")@"GGF"
 signedBy ["GGF"].

Alice makes a request to the Conference Grid Portal to retrieve a credential from the U.P.B. MyProxy Repository (*allowRequest("U.P.B. MyProxy Repository")*). The Grid Portal enforces access policies regarding clients who require its services. It verifies if the user is registered with the conference (*validConfRegistration(Name,RegNumber,"Alice")*) by asking Alice to enter her name and conference registration number (*registered(Name,RegNumber) @ "Alice"*) and then checks locally if the provided data is correct (*checkReg("Alice Smith","123abc")*).

The Grid Portal asks Alice also for the username and password with the MyProxy Repository as they are needed during credential retrieval (*proxyAccessData*("U.P.B. MyProxy Repository",*UsrName,Pwd*) @ "Alice") and tries to determine if it is permitted to interact with the indicated resource (*permitted*("U.P.B. MyProxy Repository")). For that the U.P.B. MyProxy Repository is required to reveal a credential showing the institution responsible with its administration (*administratedBy*("U.P.B. MyProxy Repository",*Institution*) @ "U.P.B. MyProxy Repository").

The U.P.B. MyProxy Repository has the following access policies:

[U.P.B. MyProxy Repository]:

retrieveCredential(*UserName,Password*) \$ Requester ←
valid(*Username,Password*),
trusted(*Requester*).

trusted(*Requester*) ←
affiliation(*Requester,"GGF"*) @ "GGF" @ *Requester*.

trusted(*Requester*) ←
id(*Requester,"U.P.B. CA"*) @ "U.P.B. CA" @ *Requester*.

administratedBy("U.P.B MyProxy Repository","U.P.B.") @ "U.P.B."
signedBy ["U.P.B."].

The U.P.B. MyProxy Repository has a credential attesting its administration by the U.P.B. and no policy protecting it. The credential is disclosed and the Conference Grid Portal proceeds further with verifying if U.P.B. is an institution registered with the conference (*registeredWithConference*("U.P.B.")). Since this is the case the U.P.B. MyProxy Repository is considered trusted and the Conference Grid Portal requests a credential on Alice behalf providing her username and password (*retrieveCredential*("Alice Smith", "123abc") @ "U.P.B. MyProxy Repository").

When asked to delegate a credential on behalf of a user, the repository has its own requirements to receive either a certificate signed by the U.P.B. Certificate Authority attesting the identity of the requester (*id*("Conference Grid Portal", "U.P.B. CA") @ "U.P.B. CA" @ "Conference Grid Portal"), or a proof of the requester being affiliated with Global Grid Forum (*affiliation*("Conference Grid Portal", "GGF") @ "GGF" @ "Conference Grid Portal"). The Conference Grid Portal has a signed Global Grid Forum (GGF) credential attesting that the Grid Portal is affiliated with this group. This credential has no protecting policy, and by disclosing it the Grid Portal proves its affiliation with GGF. Alice username and

password are correct and the trust negotiation is successful with the Conference Grid Portal retrieving a delegated credential on Alice behalf.

Alice is interested in obtaining results as soon as possible, since the data is requested by her research group. For that, Alice instructs the Conference Grid Portal to query the Monitoring and Discovery Hierarchical Service (MDHS) in her home country for searching the best Linux Cluster, in terms of memory size and number of available processors.

The MDHS policy asks the Grid Portal to have a credential signed by one of the recognized state institutions. The Conference Grid Portal provides the credential delegated on Alice behalf by the U.P.B. MyProxy Repository, and gets authorized to query for information available with the MDHS.

```
[MDHS]:
queryingAllowed(Credential) $ Requester ←
    belongsTo(Requester,Credential),
    locallySignedCredential(Credential).

locallySignedCredential(Credential) ←
    signedBy(Credential,“U.P.B.”).
locallySignedCredential(Credential) ←
    signedBy(Credential,“Navy Institute”).
...

```

The best available Linux Cluster pertains to the Research Center for Aeronautical Sciences (RCAS). The Grid Portal initiates a new trust negotiation process for submitting Alice job. The RCAS Linux Cluster policies require the Conference Grid Portal to provide a credential attesting the job acting on behalf of member of a project present in Ministry of Education(MinEducation) data base (figure 8.2):

```
[RCAS Linux Cluster]:
submit(Job) $ Requester ←
    actingOnBehalfOf(User,Job),
    member(User,Project) @ “MinEducation” @ Requester.

```

By providing Alice delegated credential, the Grid Portal demonstrates that the job is acting on Alice behalf, but the portal does not have yet a credential proving that Alice is a member of a certain project. For that, the Grid Portal forwards a query to the Ministry of Education CAS server, provides Alice’s credential and retrieves an assertion attesting that Alice participates in a project regarding signalling instrumentation. This credential is sent to the RCAS Linux Cluster and

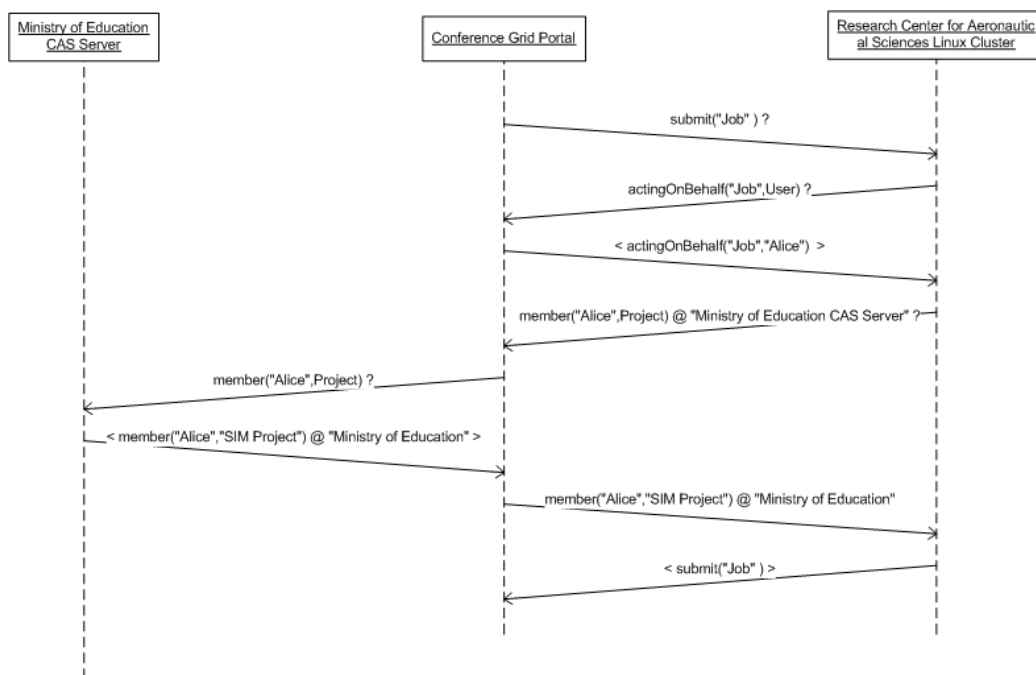


Figure 8.2: Job Submission - CAS, Grid Portal, Linux Cluster Negotiation

finally the Grid Portal is granted job submission. Since the job would need to contact other resources, a proxy credential is also delegated by the Grid Portal, on behalf of Alice, to the RCAS Linux Cluster.

[U.P.B. RFT Service]:

```

retrieve(File) $ Requester ←
  member(Requester,"Staff") @ "U.P.B. CAS" @ Requester |
  check(Rights,Requester,File) @ "U.P.B. CAS".
  
```

For retrieving the input data from the U.P.B. Reliable File Transfer(RFT) Service the job has only to prove that Alice is part of the U.P.B. staff (figure 8.3). For this, the job authenticates to the U.P.B. CAS, demonstrates to act on Alice behalf, and retrieves a credential attesting that Alice is a member of the university staff. Once Alice has demonstrated association with U.P.B., the RFT Service checks by itself, with the same U.P.B. CAS if the required file can be accessed by Alice. The RFT Service acquires an assertion which identifies the input file as belonging to Alice so the file retrieval operation is allowed.

The job contacts the Navy Institute Wave Tank and finds out that it has to provide Alice roles with the University. As Alice is careful with whom her roles are disclosed to, she has specified to the job one policy requiring entities asking

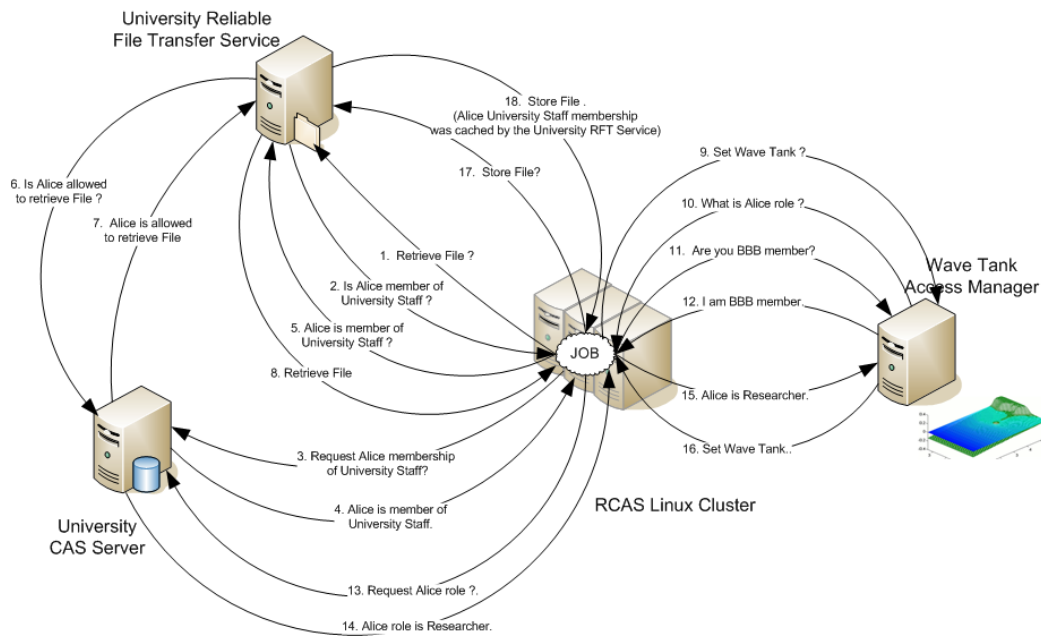


Figure 8.3: Job Negotiations

for Alice roles, to prove their liability by disclosing a Better Business Bureau (BBB) membership credential.

[Job]:

```
role("Alice",Role) @ "U.P.B. CAS" $ Requester ←
  bbbMemeber(Requester,"BBB") @ "BBB".
```

[Wave Tank]:

```
access("Wave Tank") $ Requester ←
  role(Requester,Role) @ "U.P.B. CAS" @ Requester |
  Role = "Researcher"
  notInUse("Wave Tank"),
  usedResources(Hour,"U.P.B."),
  timeLimit(Limit,"U.P.B."),
  Hour < "Limit".
  bbbMember("Wave Tank","BBB") @ "BBB"
  signedBy ["BBB"].
```

The Wave Tank has no policy protecting its BBB credential, discloses it, and Alice job proceeds with retrieving Alice roles from U.P.B. CAS. Since Alice is a

Researcher and assuming that the other local policies are fulfilled access is granted at the Wave Tank site.

The refined and corrected data generated by the Linux Cluster are saved using the U.P.B. RFT Service which allows file storage if the action is requested by a university staff member. The job has cached this credential (during a previous interaction with U.P.B. RFT Service) and can reveal it, without requiring it again from the U.P.B. CAS.

```
[U.P.B. RFT Service]:
store(File) $ Requester ←
    member(Requester,“Staff”) @ “U.P.B. CAS” @ Requester.
```

As in our initial example, the credential delegated to the Research Center for Aeronautical Sciences is to expire so the RCAS Linux Cluster has to renew its credential by contacting the MyProxy Repository. The policies enforced for renewing such a credential are that the requesting resource makes proof of a previously delegated credential and has a credential signed by Ministry of Education.

```
[U.P.B. MyProxy Repository]:
renew(User,Credetail) $ Requester ←
    previouslyDelegated(User,Credential) @ Requester.
    id(Requester,“MinEducation”) @ “MinEducation”.
```

Since RCAS Linux Cluster has the previously delegated credential and no policy regarding the disclosure of its Ministry of Education signed certificate trust negotiation succeeds again and the job certificate is renewed.

```
[RCAS Linux Cluster]:
id(“RCAS Linux Cluster”,“MinEducation”) @ “MinEducation”.
    signedBy [“MinEducation”]
```

As we have seen, the job has been submitted with only one credential, but then dynamically negotiated authorization at each resource accessed and whenever required, knew where to retrieve the needed credentials from. This was possible due to the contacted grid service ability to advertise policies, to indicate what credentials are needed and to specify where to retrieve them from. Resources were shared with no implied previous interactions with entities interested in accessing them, and with administrators involvement resumed to setting policies for access control and credential disclosure.

8.2 Enhanced Authorization Architecture

We have seen in the previous section how authorization decisions can be made automatically based on each side specified policies. Due to the expressiveness of these policies dynamic gathering of credential can also be achieved as resources advertise both their requirements and how they can be satisfied. Our proposed authorization architecture comes with a set of enhancements over the existing approaches, permitting autonomous entities to reach an agreement for authorization granting (if policies and credential disclosed permit). We summarize all the characteristics of our authorization architecture showing why it is most suitable for a large scale, pervasive and heterogeneous distributed environment as Grid.

Authorization mechanisms are distributed Resources define their own access control policies and part of the authorization process, assertions regarding user's capabilities can be obtained through interrogation of other resources (e.g repositories). Users may retrieve assertions on their own from repositories indicated by the service being accessed, or they may require at their turn the accessed service to retrieve and prove at runtime certain properties. Moreover local administrators can set their policies autonomously without needing to contact each other each time a new user needs to be authorized.

Access is negotiated Users and services may have preferences to the credentials disclosed. The access decisions is no longer a one step action as services and clients disclose iteratively their credentials increasing the level of trust until a final decision regarding authorization can be made. During the negotiation process several paths (in credential disclosure) can be followed based on what is more convenient or available for the client/service to disclose.

Support for both credential push and pull model Clients can push to the server the set of credentials for which the two have negotiated or even indicate to the server where to retrieve client credentials. The service can also pull client capabilities from other services if those services are identified in access policies involving delegation of evaluation. For example the University Reliable File Transfer Service checks with the University CAS server if Alice is allowed to retrieve a certain file. Clients are themselves part of the pull model as service may direct them to repositories from where certain capabilities can be retrieved. (the Wave Tank requires the job to pull a credential from the University CAS Server and once available to push it to the Wave Tank Access Manager)

Clients and Services are symmetric Clients and services are equal as both can specify policies for protecting available credentials. Only if these policies are satisfied by the requesting part credentials are disclosed or access granted.

Dynamic Credential Fetching Clients/Services have no longer to hardcode where to retrieve credential from, or to expose all the available credentials as they can be fetched at runtime if defined policies allow that.

Resource level policies The proposed architecture uses policies set up at the resource level, adhering to the Grid authorization model in which the resource provider has the ultimate control over the resources shared. Credential might be gathered from different entities, pushed by the client or pulled by the service but the final authorization decision is locally made.

Capability based authorization architecture Decisions regarding authorization are inferred on user capabilities, relying on client properties and not on their identities. In this way resource providers have the extensibility, flexibility and ease of enforcing access granting policies at the resource level with minimal effort. We do not impose any restriction regarding the format of the credential: attribute certificates, end entity certificates, signed assertions etc. can be used to prove client properties.

Explanation of the authorization decision The negotiation process can be traced at any moment offering to the client information on its status, where and why it has failed or how it evolved until access was granted. Using this trace the client can find out why the negotiation has failed and which credentials were required and could not be obtained.

No previous trust relationships required Trusted authorities are not required from start, as trust can be built from scratch, starting for example with usage of self signed certificates just to establish a secure connection and then exchange through policies, information regarding who is trusted at each side and where to retrieve the required capabilities from. This feature would be most useful at the authentication level as different grid service may advertise in this way, who is considered trusted, have clients automatically retrieve, if possible, credentials from those trusted authorities and be able to continue negotiation for authorization.

Chapter 9

Architecture. Implementation

It is possible to implement the extensions presented in the previous chapter on top of current Grid toolkits. We are going to present how we have integrated such an extension to the recently released Globus Toolkit version 4.0 (GT 4.0) allowing advanced access control mechanisms based on policies, dynamic negotiation for authorization and automatic fetching of credentials. One of our main design goals was easy integration with current grid services paradigms, our extension being easily pluggable to any Globus Toolkit 4.0 grid service, providing a straightforward installation with almost no additional effort. In the following sections we are going to present technical details regarding the architecture and implementation we propose as an extension to the current Grid Security Infrastructure [21]. A brief description of our work is also presented in [26].

9.1 High Level Architecture

The architecture on which our enhanced authorization scheme is built upon had to provide solutions to the following issues:

1. interception of client calls for service protection against unauthorized requests
2. augmentation of service/clients with capabilities for dynamic negotiation for authorization granting
3. reasoning on access control policies and credentials

The block diagram depicted in figure 9.1 presents the high level components of our architecture.

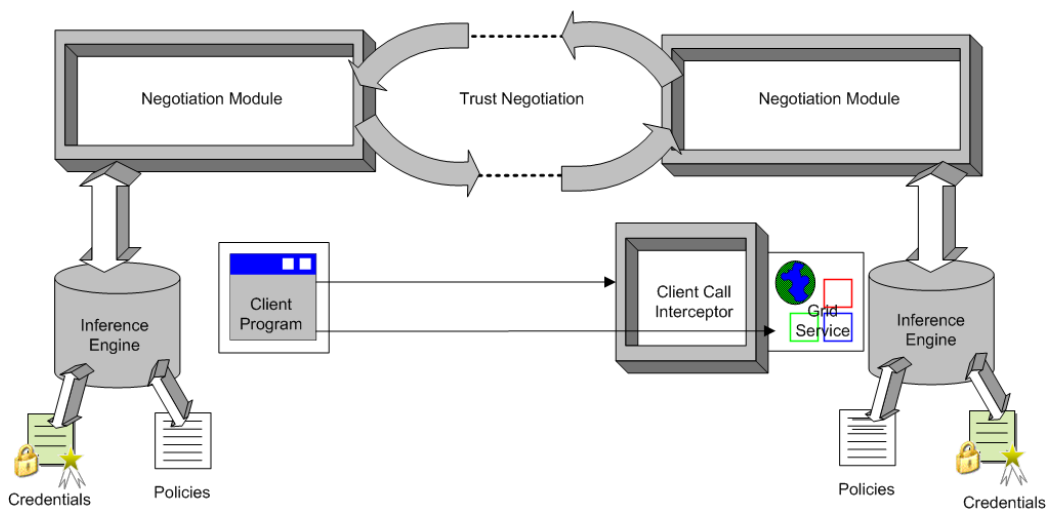


Figure 9.1: High Level Architecture

The **Client Call Interceptor** is responsible with grid service guarding against unauthorized calls. It checks if the client attempting the execution of one of the grid services operation has passed through a successful trust negotiation process and allows or denies the call accordingly. The Client Call Interceptor manages the identities of the authorized clients caching their access granted decision, for a certain period of time, so that they are not required to retake the negotiation process.

The **Negotiation Module** is responsible with sending and receiving messages consisting of client and server credentials and requirements. It interfaces with the Inference Engine and is responsible with the encapsulation and decapsulation of messages from PeerTrust language format to messages deliverable over a Grid infrastructure.

The **Inference Engine** reasons on queries and answers made by negotiation parties and the set of policies and credentials available locally, computes an answer (either send or query for a credential) and delivers it to the Negotiation Module for formatting and network sending. The Inference Engine loads at startup policies and information regarding local credentials, and dynamically adds to its knowledge base credentials retrieved during the negotiations undertaken.

We have tried to decouple as much as possible the above components from client and service code so they do not interfere with the usual client and grid service programming. For the service side, we relied mostly on configurations made through the service descriptors (with only a few lines of code added to the service itself), while for client side programming we have developed some jar files and an API for enabling negotiation for authorization.

Let's proceed in more detail with the description of each of these components.

9.2 Implementation

Our project is developed for the Globus Toolkit 4.0 Java Container, is entirely written in Java requiring Java based grid services and clients for a straightforward integration.

The GT 4.0 Authorization Framework supports pluggable Policy Decision Points (PDP) [24]. A PDP is intended to be used as an authorization mechanism, It checks what service operation has been invoked, identifies the client and returns a decision regarding access. Basically a PDP intercepts client calls and returns an authorization decision which can be either "granted" or "denied".

Several PDPs can be specified allowing a chain of verifications to be performed. The final authorization decision is the conjunction of all of them, that is, all of them must return "granted" in order to permit access. This introduces a set of limitations as configuring custom PDPs for each access policy forces the client to follow only one chain of requirements and be granted access only if all policies have been satisfied. As we have seen in our previous discussions regarding PeerTrust capabilities, we allow for several paths to be followed in making an authorization decision, with the client able to choose according to his preferences (expressed in his own policies) in which conditions his credentials are to be disclosed (available locally/less sensitive information is revealed 9.5).

One client may be asked to disclose several credentials but he may find convenient to present only a subset of them. Still the negotiation process is able to proceed with its evaluation, using what the client has agreed to provide, and check if a trust relationship can be fully established. Due to the restricting support of GT 4.0 in terms of PDPs decision aggregation, we have developed a custom PDP part of the *Client Call Interceptor*, responsible only with client call filtering and checking of a successfully completed negotiation process. The detailed architecture of our implementation is presented in figure 9.2.

The use of a PDP for grid service authorization is configured through a security descriptor (a XML file pointed out in the service WSDL file). A PDP implements one Java interface (*PDP*) and is invoked automatically by the Globus Toolkit container, when an operation is requested of the service using the PDP. The container calls one function (*isPermitted*), defined in the PDP, with parameters like the operation name and the subject making the invocation (client DN, certificates). We developed the *Interceptor Policy Decision Point* which checks if the client has succeeded in negotiating trust with the grid service, allowing invocation or denying access with a negotiation exception thrown to the client. The exception indicates to the client that a negotiation step has to be fulfilled.

If negotiation is successful the identity of the client and the operation allowed will be stored as an entry by the Negotiation Module. When invoked the Interceptor PDP searches for an entry containing the client identity and the operation attempted and makes a decision regarding the existence of a previous negotiation. Ideally, this entry should be removed once the credential from those disclosed by the client with least time till expiration has outlived its lifetime. Since we plan to support a wide variety of credentials (requiring a custom parsing for their validity), for this version of our implementation, we relied only on an entry containing the time of expiration of the client proxy certificate (the one with which the client is authenticated). A client entry contains: the client identity (client DN and CA DN), the operation allowed, and the time of expiration of the proxy certificate authenticating the client at the moment when the authorization negotiation succeeded. The Globus container refuses further client access with an expired proxy certificate, while authorized entries are periodically checked for removal (in case the proxy certificate expiration date present in a client entry has expired).

If authorization is denied, the client can start a negotiation with the service, having his negotiation module contact the service negotiation module and requesting access (the client will send a query of the form request("OperationName")). The service sends back the access policy for the operation requested and the client may answer disclosing the credentials specified by the service policy or sending his own access policy for the requested credentials. The entire negotiation process is automated, requiring the client to send only the first query indicating which operation the client is interested to call (even this step can be automatized as the exception thrown indicates the name of the operation refused).

Negotiation Module consists of two components: the *Grid Connectors* and the *PeerTrust Module*. The Grid Connectors are implemented over the Globus Toolkit 4.0 container and facilitate client server communication over a Grid environment. The PeerTrust Module is used for managing and evaluating queries through the interrogation of the Inference Engine.

On the service side the Grid Connectors are composed of two components: the *Trust Negotiation Provider Module* and the *Negotiation Topic Module*. The Trust Negotiation Provider permits user pushing of requests and credentials to the grid service while the Negotiation Topic is used for pushing service requests and credentials to the client.

A grid service describes the operations supported and their parameters through its WSDL file. At the WSDL level the service is identified as a port type having a name and a set of operations using messages for carrying input and output data to and from the service. Globus provides a useful feature in writing WSDL code for a grid service, the WSDLPreprocessor namespace. This namespace contains a tag "extends" which permits the inclusion of definitions of other port types in the current grid service definition. Using this feature it is possible to include the func-

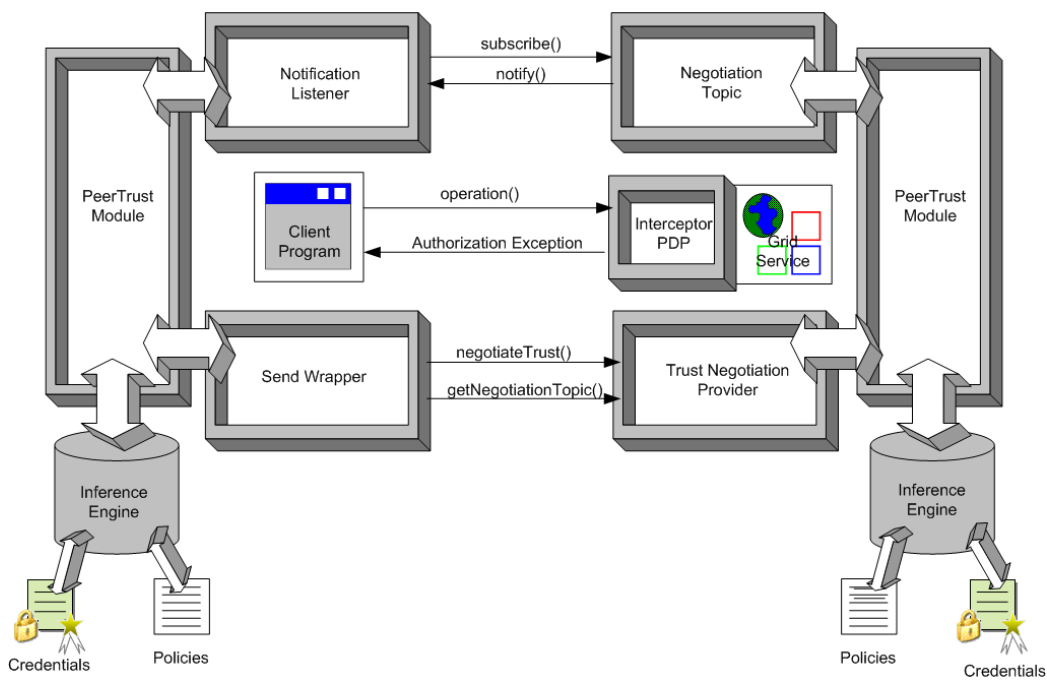


Figure 9.2: Low Level Architecture

tionality of a service (giving that this functionality is also implemented in code) into another service. We have defined a WSDL file (appendix A.1) describing one port type called *TrustNegotiation*, which we extend (appendix A.2) to confer to a general grid service the description of messages targeted for trust negotiation. The functionality of such a service is implemented outside the grid service in our *TrustNegotiationProvider*. For a grid service to use the features of the *TrustNegotiation* port and the functionality of the *TrustNegotiationProvider* it has only to extend the *TrustNegotiation* port type in its WSDL code (appendix A.2) and specify in the service deployment descriptor the *TrustNegotiationProvider* as one of the service providers (appendix A.3). Configuring a grid service to use these mechanisms makes it expose two operations: *getNegotiationTopic* and *negotiateTrust* which are going to be used for client pushing of policy requirements and proofs to the grid service during the trust negotiation process. We emphasize that in order to confer this functionality to a usual grid service we have to modify only its descriptors (configuration) files.

The disclosure of a policy from one entity to another one may result in a credential being fetched from a service which at its turn may result in another policy based trust negotiation process, making it difficult to predict when one request can finish its evaluation. In order to overcome this problem we used the GT 4.0 support for WS-BaseNotification [2] and WS-Topics [48] as mechanisms

for asynchronously pushing policies and proofs from the grid service to the client.

The WS-Notifications family of specifications includes WS-BaseNotification and WS-Topics, standardizing asynchronous communications between consumers and providers. WS-Topics specifies how topics can be organized and categorized as items of interest for subscription. We associate each trust negotiation (triggered by client requests for service operations) with a topic (Negotiation Topic) of interest through which messages can be delivered to the client side. Consumers register themselves with a certain topic of interest, while providers deliver the notifications to the consumers, each time the topic has changed. Notification producers (in our case grid services) expose a subscribe operation through which, each consumer (either grid service or client) can register with a certain topic. Notification consumers expose a notify function that producers use to send the notification.

In our architecture each client interested in negotiating trust has to call the `getNegotiationTopic` function (exposed by the grid service through the extension of the `TrustNegotiation` port and usage of the `TrustNegotiationProvider`). The client receives a topic [48] (actually a negotiation identifier) associated with a unique namespace and subscribes to it. For sending client policy requirements and proofs the client uses the `negotiateTrust` function while service side policies and disclosed credentials reach the client asynchronously as notifications. The notification producer functionality is plugged into the grid service also by means of providers (configuring in the service descriptor file two providers `SubscribeProvider` and `GetCurrentMessageProvider` which are part of the GT 4.0). The client and service sequence of operation calls during a negotiation process is presented in figure 9.3.

The use of the GT 4.0 implementation of the notification paradigm imposes one limitations to the grid service that of exposing its state as a “Resource” in conformity with the Web Service Resource Framework [47]. “Resources” are used for storing a web service state from one invocation to another. Thus the only requirement we impose for integrating our policy based architecture for grid service authorization is having the service use such a “Resource”. The addition of a resource to a grid service is straightforward, with only minimal additions to its descriptor files and code (it is only a matter of having the service implement one interfaces *Resource* with no methods). The “Resource” used by the service (which may be implemented part of the service or as a separate instance) has also to implement one interface (*TopicListAccessor*), with only one function (`getTopicList`) used by the `TrustNegotiationProvider` to store and remove negotiation topics.

The client has a complementary functionality to that of a grid service. For client programming we have developed a jar file and an API, to facilitate easy integration of trust negotiation capabilities to client code. The client has to use one class (`GridClientTrustNegotiation`) for setting the grid service address and the namespace of the negotiation topic received. Also the client has to hide the service invocations by implementing one interface (`SendWrapper`) part of our API, for

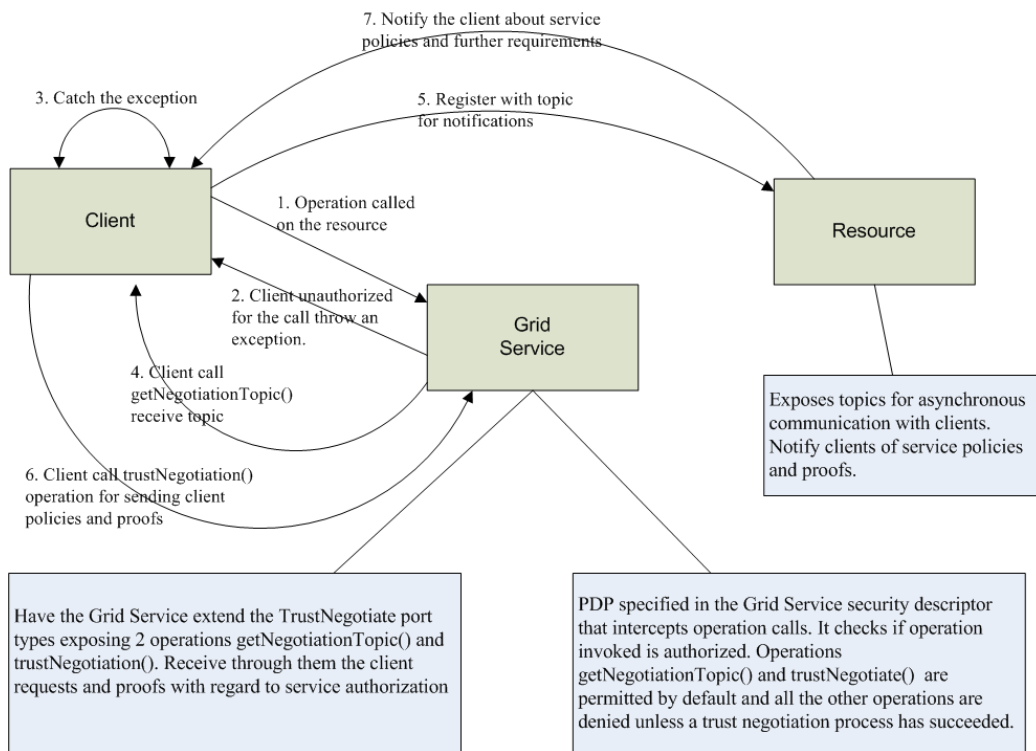


Figure 9.3: Flow Diagram

using the stub of the service with whom he wants to negotiate trust.

Service stubs mask the intricacies of communication with a service and can be generated automatically from the grid service WSDL file. In Java, stubs are represented as objects with methods for each operation exposed by the service. Since each stub is specifically generated for the service it targets to invoke, we have developed an interface abstracting the calls made to a stub with negotiation capabilities (a stub supporting our defined getNegotiationTopic and negotiateTrust operations). The client has to implement this interface and in one of its functions (sendMessage) to call the trustNegotiate operation on the stub. (this can also be automated by means of reflection). By implementing the same interface (SendWrapper), we have provided to the client an object/class he can use for automatic fetching of a SAML assertion from a CAS server.

On the client side a thread (GridClientNotificationThread) registers with the topic returned by the grid service and listens for notifications. When a notification is delivered, it is first processed by the thread and then passed to the PeerTrust Module for the computation of the next step in the negotiation process. SendWrappers are used on the client side for sending queries to the service with which the negotiation process is undergoing, or to third party services (e.g. CAS) for cre-

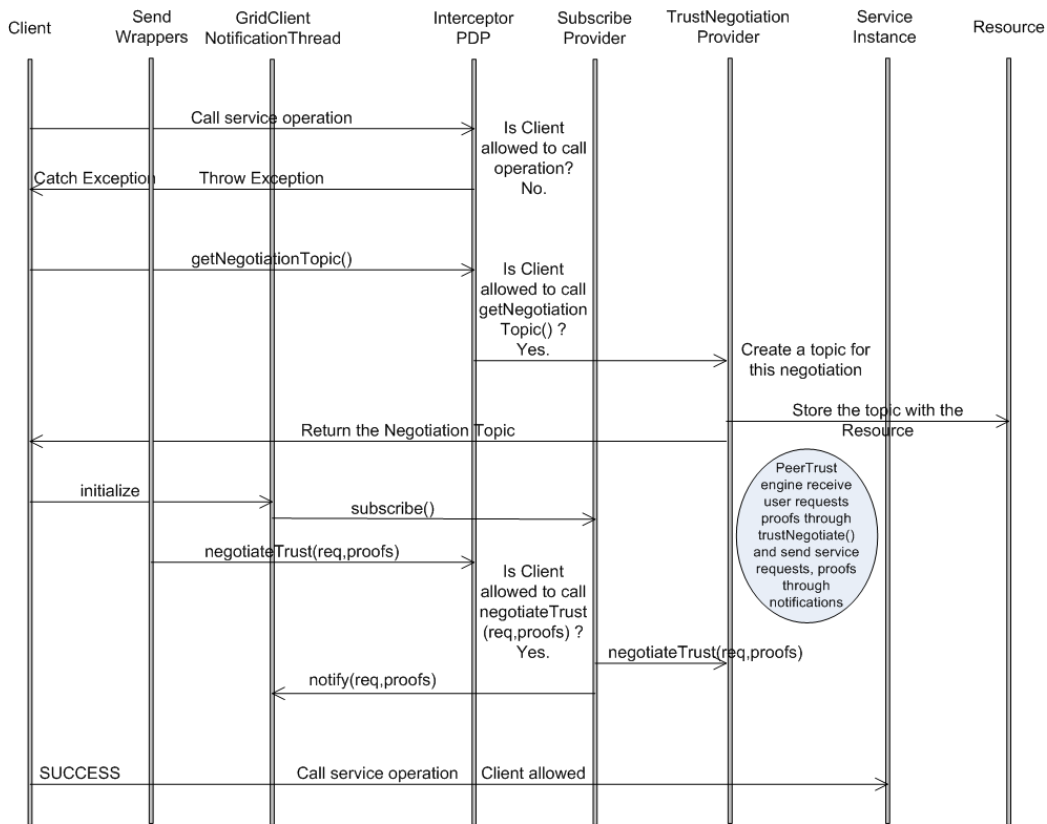


Figure 9.4: Sequence Diagram

dential retrieval. The sequence diagram of client calls to the service is depicted in figure 9.4. The same client code can be used by the service to register for negotiation with other services or to retrieve credentials, as one service may be the client of another service with whom it negotiates trust.

PeerTrust Module is responsible for query and answer management. Messages exchanged between entities carry the following information:

- a query or an answer
- trace of the messages exchanged
- proof attesting the level of trust reached
- the credentials validating the proof

The proof contains information about the credentials and signed rules disclosed up to the moment between the two negotiating entities, and possibly other credentials required and fetched for the current negotiation from third parties. The

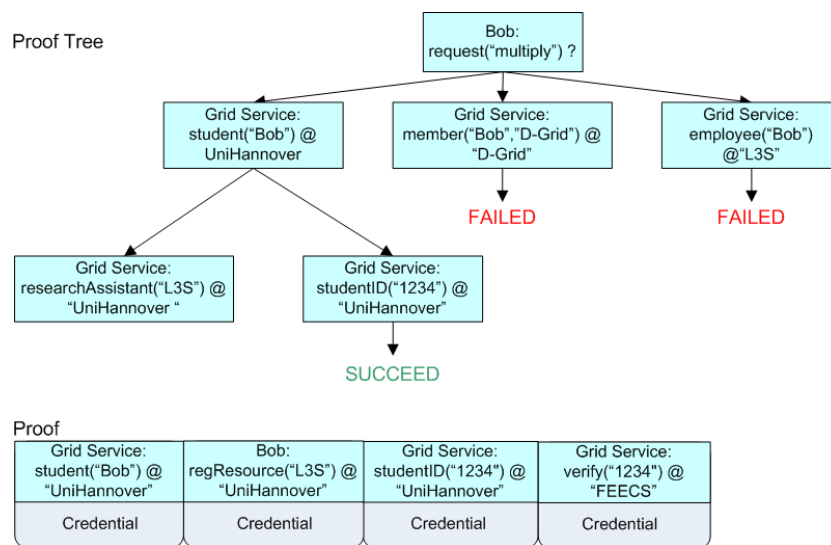


Figure 9.5: Proof and Proof Tree computed on the grid service side

proof attests that the client/service has met the goal of a query (either for access or for credential disclosure) during a negotiation process. It includes the rules which have been requested and the chain of credential satisfying them.

The evaluation of a query is based on the (possibly) cooperative distributed computation of a proof tree. At any moment a proof tree includes the policy rules that have been satisfied up to the moment, the instantiations of variables in those rules and the credentials themselves.

The PeerTrust Module receives messages from the Grid Connectors, represents or integrates them into existing proof trees and evaluates them further. A proof tree represents the different paths on which the negotiation may evolve. The root of a proof tree is a query either for access granting or for disclosing a credential. Starting from the root of the tree the PeerTrust Module builds new levels of nodes by adding new queries (sent to third parties or to the party formulating the initial query). A proof tree is represented in figure 9.5. For each subquery sent the PeerTrust Module expects to receive either a chain of certificates or a failed answer. Each query sent to third parties initiates the building of a new proof tree on these parties.

The PeerTrust Module on an entity may evaluate more than one proof tree as it may receive queries from different parties for access or credential disclosure. A proof tree is considered to be answered when at least one path of disclosed credentials leads to the satisfaction of the initial query (the one present in the root of the tree). In this case a message is forwarded through the Grid Connectors to the originating part. A proof tree may be further evaluated based on local

policies until it cannot be expanded anymore due to subqueries that have to be delegated for evaluation to either the originating part or to other third parties. Each subquery is forwarded as a new query requesting evaluation to the targeted parties. When a query is satisfied the entire chain of certificates (represented as the message proof 9.5) is forwarded to the part who made the root query. The chain of certificates is plugged by the initiator in the originating proof tree which may complete a path and thus a further chain of certificates is disclosed and forwarded to another entity and so on. If a proof tree failed evaluation on all its paths the entity who sent the query in the root of the tree is informed, so the path in its own proof tree on which the query resided is not followed anymore.

The Inference Engine is based on Minerva Prolog which provides a Java interface to a Prolog engine. The Inference Engine loads local policies and is queried by the PeerTrust Module for evaluating the queries available with the proof trees. Based on its answers new queries can be generated, access granted or credential disclosed. The inference engine reasons only on the policy rules/credentials available locally and queries/credentials received from third parties while the other modules are responsible with addition of the credentials to the messages exchanged. Policies are represented by the Inference Engine as Prolog rules requiring to be satisfied. A query received from an entity is transformed in a Prolog query and requested from the Inference Engine. The answer may be the request for proving another Prolog rule which is translated in a policy rule and sent to its targeted destination or a credential having to be released, which at its turn is identified by the PeerTrust Module and delivered by the Grid Connectors.

Credentials used in our implementation may virtually be expressed under any kind of format (certificates, signed assertions, signed RDF statements) as the policies and the negotiation modules are decoupled of credential types. The only requirement is that the credentials used are able to be mapped to the policy rules we use. Currently we have support for X.509 Certificates which carry in non-critical extension the expression of the policy they satisfy. In addition to this we have integrated the use of proxy certificates which carry SAML Assertions retrieved from a Community Authorization Service CAS.

The client authenticates and is authorized by the CAS based on his proxy certificate and retrieves a SAML assertion regarding his allowed actions. The SAML (appendix B.3) assertion indicates its issuer, the subject of the statement (the client DN), a resource which is an object part of a namespace, and an action allowed on that object. The client can wrap this SAML Assertion in a proxy certificate and push it as a proof for a policy rule. The assertion basically expresses a Subject-Action-Object relationship and can be easily mapped to policy rules with only one argument or requiring only a capability of the Subject (through these use of CAS assertions we can prove simple policy rules e.g Client-member-Project may prove a rule member(Client)@Project signedBy[CAS], or

Client-student-University for student(Client)@University signedBy [CAS]).

Our future research will focus on online credential repositories in order to improve automatic fetching of credentials capabilities. We have integrated CAS servers and plan to do the same with MyProxy repositories. The difficulty of supporting various credentials comes from the wide variety of their representations. Furthermore credential repositories releasing such credentials do not share a common interface (requiring an ad-hoc integration by means of wrappers) and do not yet allow us to store arbitrary credentials which would be desirable for more complex scenarios.

Chapter 10

Related Work

Authorization has been tackled in different projects, motivated by different scenarios and applied to some extent also to Grid middleware. Answers have been found and solutions offered but none fully addressing the authorization needs of a large scale, dynamic Grid environment. We present some approaches to authorization and present the ways in which they differ from the architecture we propose.

10.1 VOMS

Virtual Organization Management Service (VOMS [6]) was developed by the European Data Grid Project to allow X.509 Attribute Certificate based access control to the Globus Toolkit Resource Allocation Manager responsible for job execution. The user has to contact several VOMS servers, retrieve VOMS pseudo certificates containing his attributes (groups and roles) and include these pseudo certificates in a non-critical extension of a usual proxy certificate. The user pushes the proxy certificate to the resource Gatekeeper authorizing the submitted job on the attributes included in the certificate received. The limitations of this approach is that the user has to know in advance what attributes has to provide and where to obtain them from and can not impose any restrictions on the resource accessed.

10.2 PERMIS

The PERMIS [10] project devised a role based authorization scheme using X.509 Attribute Certificates. PERMIS architecture consists of a privilege allocation subsystem and a privilege verification subsystem. The privilege allocation subsystem is responsible for allocating roles to users in the form of X.509 Attribute Certificates and storing them in LDAP directories for later retrieval by the privilege verification subsystem. The privilege verification subsystem is initialized with a

list of LDAP directories from where it retrieves the policy specifying what actions each role is allowed to take and for subsequently retrieving X.509 Attribute Certificates containing users' roles. Users are authenticated and their roles can be either pushed to the privilege verification subsystem or pulled from the LDAP directories. Based on the policy specifying each role's allowed operations and user role the authorization is granted or denied.

PERMIS project had as its main goal the construction of an X.509 role based Privilege Management Infrastructure that could accommodate diverse role oriented scenarios. It does not addresses issues like querying resources for needed attributes for access granting, nor imposes restrictions on the resources accessed as the service providers are considered to be trusted being associated with public institutions. (e.g. city hall, city parking tickets database, city electronic tendering application). Also the authorization decision is a one step try consisting of the gathering of users X.509 Attribute Certificates from previously known locations and the verification of the user requested action against his roles.

10.3 AKENTI

AKENTI [46] is an authorization mechanism (PDP) which uses distributed policy certificates signed by stakeholders from different domains, in order to make a decision regarding access to a resource. Akenti uses an XML format for representing three types of certificates: policy certificates, use-condition certificates and attribute certificates. *Policy certificates* consist of trusted CAs and stakeholders who may issue use-condition certificates for the resource as well as a list of URLs where use-condition certificates and attribute certificates are placed. *Use-condition certificates* contain a relational expression of the attributes a user must hold in order to get a set of rights and the list of authoritative principals for the indicated attributes. Attributes can be defined in the context of the resource and can be combined with boolean operators such as && or || or express real-time or system attributes. If the Akenti engine can not evaluate such attributes it may return them to the Policy Enforcement Point (PEP) for processing. *Attribute certificates* contain an identity to whom an attribute-value pair is entitled being signed by attribute authorities specified in use-condition certificates. Attribute certificates are used to increase the set of rights a user has making an unavailable attribute certificate to limit or deny the required access.

Akenti operates as follows. A user is authenticated using his X.509 certificate and requests access to a resource. The Akenti engine gather all certificates available with the user (attribute certificates) and resource (use-condition certificates) in a pull model, verifies them and makes a decision regarding user rights with the resource.

Limitations of the Akenti approach consists of the already known locations of user attributes and the lack of user control over the rights disclosed. Our approach takes into account this situations and gives both to the client and service the possibility of protecting credential disclosure and of specifying where to retrieve them from.

10.4 PRIMA

System for Privilege Management and Authorization, PRIMA [27] focuses on management and enforcement of fine-grained privileges for authorization. PRIMA uses X.509 Attribute Certificates that carry privileges and policy statements. Resource administrators and stakeholders delegate privileges to Grid users which can further delegate these privileges to other users. The user presents his privileges to the resource Policy Enforcement Point (PEP) which validates user attributes and verifies with the resource Policy Decision Point (PDP) if the issuers are authoritative for user's presented privileges. This verification is made against a privilege management policy available with the PDP. All acknowledged privileges are gathered by the PEP in a dynamic policy which is presented to the PDP to check against the configured access control policies (enable resource administrators to overrule access rights assigned to users, so that the final authorization decision stays local). The PDP returns to the PEP an authorization decision (deny or permit) and some recommendations for setting up an environment (local account) with support for the validated user privileges. The PEP may be able to apply only a set of user privileges to the created account, ignoring the inapplicable privileges in an opportunistic approach hoping that the user job would succeed even with less privileges. The PEP relies on the operating system support to configure user privileges (file access permissions, user quotas, network access) to the local user account [28].

Our approach differs from PRIMA through the services' ability to advertise their authorization requirements (in PRIMA design users decide manually what privileges to present to the accessed resource) and through support for a several steps authorization in which users and services increase iteratively their trust relationship. Also PRIMA does not addresses client requirements in terms of the services accessed. On the other hand PRIMA deals with enforcement of policies at the file system and network level while we are considering enforcement of policies for access and credential disclosed.

10.5 Community Authorization Service (CAS)

CAS is used to release SAML assertions about user rights established at the VO level. The assertion contains the DN of the user and the actions allowed on a certain resource. The assertion is packed by the client into a user proxy certificate and pushed to the service, which would be responsible of parsing and interpreting it. Permissions given to the user are at the intersection of the rights given by the resource to the VO, those entitled by the VO to the client and those established locally for the user DN. In our architecture, we have made use of CAS assertions to prove simple signed rules consisting of a predicate with a single parameter, by mapping the action of the assertion to the predicate, the subject to the parameter of the predicate, the resource to rule issuer parameter (@“Issuer”) and the CAS identity for the “signedBy” argument. While CAS is utilized as a central server for releasing assertions, our approach can accommodate distributed credentials repositories releasing various credentials (CAS issues assertions stating who is entitled to take what action at which resource while we can infer the access decision on richer assertions and the relations between them).

10.6 GridShib

GridShib [49] is a newly started project which aims at integrating Shibboleth [42], an attribute authority service developed by Internet2 community for cross-organizational identity federation into the Globus Toolkit Grid Security Infrastructure. Shibboleth is a system permitting assertions of user attributes between organizations. When a user requires access to a resource pertaining to another organization, it passes a handle token pointing to its home organization attribute authority from where the resource can retrieve user attributes. The attribute authority enforces privacy restrictions, as the user can specify a set of policies identifying resources to which user attributes may be disclosed.

GridShib intent is to allow Grid services to determine the address of the Shibboleth attribute service, obtain the user attributes authorized to be revealed to the Grid service and use them in the Grid service authorization decision making. The GridShib project focuses on the use of Shibboleth issued attributes for Grid service authorization. These attributes can be used also in our architecture as an alternative mean for proving user properties. GridShib does not support negotiation for access granting and the authorization decision relies only on credentials pulled from a single source indicated by the client.

10.7 XPOLA

XPOLA [15] provides a capability-based authorization infrastructure for Grids, enabling realization of a user-level, peer-to-peer collaboration Grid. XPOLA proposes the creation of a peer to peer Grid that allows group of people to share a set of services. In this model one service provider deploys a service on a remote resource, creates a set of capability tokens with authorization policies containing the identities of users allowed to access the service and distributes the tokens to users. Users present their token to the remote resource running the service and if the contained policies allow the operation required by the user, the service is invoked. In case the user needs to access other services, the initial service provider makes these invocations on user behalf, assuring that user capabilities respect the service entitled capabilities to other service providers. The main difference between XPOLA and our approach to authorization is that we do not couple authorization decision with user identities. Previous interactions (such as having the service release client capabilities) are not required in our design, as the authorization decision we infer can utilize client third parties issued attributes.

Chapter 11

Conclusions

The main contributions this thesis brings to Grid authorization mechanisms are:

- Policy-based authorization relying on sides' properties rather than their identities.
- Self-describing resources for access requirements.
- Dynamic negotiation for Grid service authorization supporting both pulling and pushing of client attributes.
- Automatic credential fetching from credential repositories.

All these features are implemented over the Globus Toolkit 4.0 as an extension to current Grid Security Infrastructure. Our proposed architecture is general enough to deal with different policy languages and policy evaluation procedures, which may be plugged in the negotiation module of each entity. In our implementation we have made use of the PeerTrust project and language [19] with built-in support and verification of X.509 certificates. We have tested our implementation accessing different Grid services and automatic credential fetching from a CAS server with positive results. Our implementation is distributed as jar files and APIs and allows easily integration with GT 4.0 grid services and client development.

We are also planning to perform more experiments in order to study the performance loss induced by policy integration into current grid services. However, although we know in advance that access will be somewhat more expensive than based on current approaches, still the trade permits automation of actions which rather have to be done offline (through hardcoding or previous interactions with resource owners). We investigate an extreme scenario with a two step negotiation resumed at querying resource access policies and their fulfillment.

11.1 Future Work

Our future research will focus on online credential repositories in order to improve automatic fetching of credentials and user capabilities. We have integrated CAS servers and we plan to do the same with MyProxy repositories. Unfortunately MyProxy is not web service compliant and as with other credential repositories do not share a common interface (requiring an ad-hoc integration by means of wrappers). Current credentials express simple user attributes (e.g. role) or a user-resource-action relation and not more complex relations (e.g. signed rules) which would be desirable for the complex scenarios we advertised.

As Grid Portals are popular interfaces to Grids, we intend to develop a trust negotiation enabled Grid Portal for facilitating job submission and initial credential fetching through a web-like interface. Also we envision trust negotiation in a broader context as part of a Grid logon entry point able to coordinate and monitor where a negotiation for access may succeed.

Another area of research is how our traceable negotiation process can be utilized for billing and logging. Our infrastructure supports negotiation recording at each resource involved, and the iterative process of requests and disclosures for credentials may be extensible to negotiation for pricing providing also the proofs backing up the agreement reached.

In addition we plan to integrate a more expressive language than PeerTrust, for example to allow action execution when certain policies are satisfied (e.g. logging information). We plan to do so in the context of European Union REVERSE Network of Excellence [38] in which the Protune [9] language is identified as the future European Semantic Web policy language.

11.2 Summary

We started this paper with a brief outlook over Grid concepts and goals and a description of the state of the art in Grid Security Infrastructure. We identified the limitations of current authorization schemes, and proposed trust negotiation as a solution for overcoming identity based access, hardcoding of credential fetching, and previously known access requirements. The enhancements we promote are: policy based authorization, self-describing resources in terms of access requirements, negotiation for access granting and credential disclosing, and automatic credential fetching. We have shown how these improvements can be realized using rule based access policies and a back end rule inference engine. We implemented these extensions on top of Globus Toolkit 4.0 and presented here the architecture we relied on.

11.3 Acknowledgements

During my internship at L3S Research Center, I had the opportunity to discuss some of the aspects described in this thesis with Frank Siebenlist, senior software architect at Argonne National Laboratory, and part of the Globus Alliance team. I would like to thank Frank Siebenlist for the interesting views and suggestions for future developments of here proposed approach to Grid authorization. I would like also to thank Daniel Olmedilla for his advice and guidance on policy related issues and for all his support during the six month I spent at L3S Research Center.

Bibliography

- [1] Universal description, discovery and integration. <http://www.uddi.org/specification.html>.
- [2] Web services base notification. <http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-03.pdf>.
- [3] Web services description language (wsdl). <http://www.w3.org/TR/wsdl>.
- [4] Web services trust language (ws-trust). <http://specs.xmlsoap.org/ws/2005/02/trust/WS-Trust.pdf>.
- [5] A globus toolkit primer. http://www.globus.org/toolkit/docs/4.0/key/GT4_Primer_0.6.pdf.
- [6] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agello, A. Frohner, A. Gianoli, K. Lórentey, and F. Spataro. Voms: An authorization system for virtual organizations. In *Proceedings of the 1st European Across Grids Conference*, Santiago de Compostela, Feb. 2003.
- [7] J. Basney, M. Humphrey, and V. Welch. *The MyProxy Online Credential Repository*. Software: Practice and Experience, 2005.
- [8] J. Basney, W. Nejdil, D. Olmedilla, V. Welch, and M. Winslett. Negotiating trust on the grid. In *2nd WWW Workshop on Semantics in P2P and Grid Computing*, New York, USA, may 2004.
- [9] P. A. Bonatti and D. Olmedilla. Driving and monitoring provisional trust negotiation with metapolicies. In *6th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2005)*, pages 14–23, Stockholm, Sweden, jun 2005. IEEE Computer Society.
- [10] D. Chadwick and O. Otenko. The permis x.509 role based privilege management infrastructure. In *7th ACM Symposium on Access Control Models and Technologies*, 2002.

- [11] Condor-g. <http://www.cs.wisc.edu/condor/condorg/>.
- [12] Condor project. <http://www.cs.wisc.edu/condor/>.
- [13] Earth system grid project. <http://www.earthsystemgrid.org/>.
- [14] extensible access control markup language. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.
- [15] L. Fang, D. Gannon, and F. Siebenlist. Xpola - an extensible capability-based authorization infrastructure for grids. In *4th Annual PKI R&D Workshop*, 2005.
- [16] S. Farrel and R. Housley. An internet attribute certificate profile for authorization, rfc3281.
- [17] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. Open Grid Service Infrastructure WG, Global Grid Forum, 2002.
- [18] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. In *International J. Supercomputer Applications*, 2001.
- [19] R. Gavriiloaie, W. Nejdl, D. Olmedilla, K. E. Seamons, and M. Winslett. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *1st European Semantic Web Symposium (ESWS 2004)*, volume 3053 of *Lecture Notes in Computer Science*, pages 342–356, Heraklion, Crete, Greece, may 2004. Springer.
- [20] Globus toolkit 4.0. <http://www.globus.org/toolkit/docs/4.0/>.
- [21] Grid security infrastructure. <http://www.globus.org/security/overview.html>.
- [22] B. Grosz. Representing e-business rules for the semantic web: Situated courteous logic programs in ruleml. 2001.
- [23] B. Grosz and T. Poon. Sweetdeal: Representing agent contracts with exceptions using xml rules, ontologies, and process descriptions. In *Proceedings of the 12th World Wide Web Conference*, 2003.
- [24] Gt 4.0 authorization framework.
<http://www.globus.org/toolkit/docs/4.0/security/authzframe/>.
- [25] Soap specification. <http://www.w3.org/TR/soap/>.

- [26] D. O. Ionut Constandache, Wolfgang Nejdl. Policy based dynamic negotiation for grid services authorization. submitted for publishing.
- [27] M. Lorch, D. Adams, D. Kafura, M. Koneni, A. Rathi, and S. Shah. The prima system for privilege management, authorization and enforcement in grid environments. In *Proceedings of the 4th Int. Workshop on Grid Computing - Grid 2003*, Phoenix, AZ, USA, Nov. 2003.
- [28] M. Lorch and D. G. Kafura. The prima grid authorization system. *J. Grid Comput.*, 2(3):279–298, 2004.
- [29] Mpich-g2. <http://www3.niu.edu/mpi/>.
- [30] Network for earthquake engineering.
- [31] J. Novotny, S. Tuecke, and V. Welch. An online credential repository for the grid: MyProxy. In *Symposium on High Performance Distributed Computing*, San Francisco, Aug. 2001.
- [32] OASIS. Web service security: Soap message security, 2004.
- [33] Ogsa authorization working group. <https://forge.gridforum.org/projects/ogsa-authz>.
- [34] Owl web ontology language. <http://www.w3.org/TR/owl-features/>.
- [35] L. Pearlman, C. Kesselman, V. Welch, I. Foster, and S. Tuecke. The community authorization service: Status and future. In *Proceedings of the Conference for Computing in High Energy and Nuclear Physics*, La Jolla, California, USA, Mar. 2003.
- [36] Peertrust project. <http://sourceforge.net/projects/peertrust/>.
- [37] Portable batch system. <http://www.openpbs.org/>.
- [38] Reverse project. i2 group on policy language, enforcement, composition. <http://reverse.net/I2/>.
- [39] D. D. Roure, N. Jennings, and N. Shadbolt. *Research Agenda for the Semantic Grid: A Future e-Science Infrastructure*. National e-Science Center, Edinburgh, UK, 2001.
- [40] D. D. Roure, N. R. Jennings, and N. R. Shadbolt. The semantic grid: Past, present, and future. pages 669–681. *IEEE* 93(3), 2005.

- [41] Security assertion markup language. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security.
- [42] Shibboleth project, internet2. <http://shibboleth.internet2.edu>.
- [43] Teragrid. <http://www.teragrid.org/>.
- [44] Tera-scale open-source resource and queue manager. <http://www.supercluster.org/>.
- [45] The open grid services architecture, version 1.0.
- [46] M. R. Thompson, A. Essiari, and S. Mudumbai. Certificate-based authorization policy in a pki environment. *ACM Trans. Inf. Syst. Secur.*, 6(4):566–588, 2003.
- [47] Web services resource framework. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf.
- [48] Web Services Topics. <http://docs.oasis-open.org/wsn/2004/06/wsn-WS-Topics-1.2-draft-01.pdf>.
- [49] V. Welch, T. Barton, K. Keahey, and F. Siebenlist. Attributes, anonymity, and access: Shibboleth and globus integration to facilitate grid collaboration. In *4th Annual PKI R&D Workshop*, 2005.
- [50] V. Welch, I. Foster, C. Kesselman, O. Mulmo, L. Pearlman, S. Tuecke, J. Gawor, S. Meder, and F. Siebenlist. X.509 proxy certificates for dynamic delegation. In *3rd Annual PKI R&D Workshop*, Apr. 2004.
- [51] W. H. Winsborough, K. E. Seamons, and V. E. Jones. Automated trust negotiation. DARPA Information Survivability Conference and Exposition, IEEE Press, Jan 2000.
- [52] Worldwide Web Consortium. *Semantic Web Activity Statement*, 2004.
- [53] Ws-addressing. <http://www.w3.org/Submission/ws-addressing/>.
- [54] Ws-notification. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn.
- [55] Ws-secureconversation. <http://specs.xmlsoap.org/ws/2005/02/sc/WS-SecureConversation.pdf>.

Appendix A

WSDL files

A.1 TrustNegotiation port WSDL file

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="TrustNegotiation"
  targetNamespace="http://linux.egov.pub.ro/ionut/TrustNegotiation.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://linux.egov.pub.ro/ionut/TrustNegotiation.wsdl"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <wsdl:types>
    <xsd:schema
      targetNamespace="http://linux.egov.pub.ro/ionut/TrustNegotiation.wsdl"
      xmlns:tns="http://linux.egov.pub.ro/ionut/TrustNegotiation.wsdl"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">

      <xsd:complexType name="peer">
        <xsd:sequence>
          <xsd:element name="alias" type="xsd:string"/>
          <xsd:element name="address" type="xsd:string"/>
          <xsd:element name="port" type="xsd:int"/>
        </xsd:sequence>
      </xsd:complexType>

      <xsd:element name="negotiateTrust">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="source" type="tns:peer"/>
            <xsd:element name="target" type="tns:peer"/>
            <xsd:element name="trace" minOccurs="1" maxOccurs="unbounded"
              type="xsd:string"/>
            <xsd:element name="goal" type="xsd:string"/>
            <xsd:element name="proof" type="xsd:string"/>
            <xsd:element name="status" type="xsd:int"/>
            <xsd:element name="reqQueryId" type="xsd:long"/>
            <xsd:element name="messageType" type="xsd:int"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </wsdl:types>

```

```

</xsd:element>

<xsd:element name="negotiateTrustResponse">
  <xsd:complexType/>
</xsd:element>

<xsd:element name="getNotificationURI">
  <xsd:complexType/>
</xsd:element>

<xsd:element name="getNotificationURIResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="namespaceURI" type="xsd:string"/>
      <xsd:element name="localPart" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<!-- Custom Notification Message -->

<xsd:element name="TrustNegotiationNotificationMessage"
  type="tns:TrustNegotiationNotificationMessageType"/>

<xsd:complexType name="TrustNegotiationNotificationMessageType">
  <xsd:sequence>
    <xsd:element name="source" type="tns:peer"/>
    <xsd:element name="target" type="tns:peer"/>
    <xsd:element name="trace" minOccurs="1" maxOccurs="unbounded"
      type="xsd:string"/>
    <xsd:element name="goal" type="xsd:string"/>
    <xsd:element name="proof" type="xsd:string"/>
    <xsd:element name="status" type="xsd:int"/>
    <xsd:element name="reqQueryId" type="xsd:long"/>
    <xsd:element name="messageType" type="xsd:int"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="TrustNegotiationNotificationMessageWrapperType">
  <xsd:sequence>
    <xsd:element ref="tns:TrustNegotiationNotificationMessage"/>
  </xsd:sequence>
</xsd:complexType>

</xsd:schema>
</wsdl:types>

<wsdl:message name="NegotiateTrustRequest">
  <wsdl:part name="request" element="tns:negotiateTrust"/>
</wsdl:message>

<wsdl:message name="NegotiateTrustResponse">
  <wsdl:part name="response" element="tns:negotiateTrustResponse"/>
</wsdl:message>

<wsdl:message name="GetNotificationURIRequest">
  <wsdl:part name="request" element="tns:getNotificationURI"/>
</wsdl:message>

<wsdl:message name="GetNotificationURIResponse">
  <wsdl:part name="response" element="tns:getNotificationURIResponse"/>
</wsdl:message>

```

```

<wsdl:portType name="TrustNegotiation">
  <wsdl:operation name="negotiateTrust">
    <wsdl:input message="tns:NegotiateTrustRequest"/>
    <wsdl:output message="tns:NegotiateTrustResponse"/>
  </wsdl:operation>
  <wsdl:operation name="getNotificationURI">
    <wsdl:input message="tns:GetNotificationURIRequest"/>
    <wsdl:output message="tns:GetNotificationURIResponse"/>
  </wsdl:operation>
</wsdl:portType>
</definitions>

```

A.2 Service WSDL file

```

<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MathService"
  targetNamespace="http://www.globus.org/gt4ide/example/MathService"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://www.globus.org/gt4ide/example/MathService"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdpp="http://www.globus.org/namespaces/2004/10/WSDLPreprocessor"
  xmlns:wslw=
    "http://docs.oasis-open.org/wsr/2004/06/wsr-WS-ResourceLifetime-1.2-draft-01.wsdl"
  xmlns:wsrp=
    "http://docs.oasis-open.org/wsr/2004/06/wsr-WS-ResourceProperties-1.2-draft-01.xsd"
  xmlns:wsrpw=
    "http://docs.oasis-open.org/wsr/2004/06/wsr-WS-ResourceProperties-1.2-draft-01.wsdl"
  xmlns:wstnw=
    "http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-01.wsdl"
  xmlns:wstn=
    "http://linux.egov.pub.ro/ionut/TrustNegotiation.wsdl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <wsdl:import
    namespace=
      "http://docs.oasis-open.org/wsr/2004/06/wsr-WS-ResourceProperties-1.2-draft-01.wsdl"
    location="../../../wsrf/properties/WS-ResourceProperties.wsdl" />

  <wsdl:import
    namespace=
      "http://docs.oasis-open.org/wsr/2004/06/wsr-WS-ResourceLifetime-1.2-draft-01.wsdl"
    location="../../../wsrf/lifetime/WS-ResourceLifetime.wsdl" />

  <wsdl:import
    namespace=
      "http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-01.wsdl"
    location="../../../wsrf/notification/WS-BaseN.wsdl"/>

  <wsdl:import namespace="http://linux.egov.pub.ro/ionut/TrustNegotiation.wsdl" location="TrustNegotiation.wsdl"/>

  <types>
  <xsd:schema targetNamespace="http://www.globus.org/gt4ide/example/MathService"
    xmlns:tns="http://www.globus.org/gt4ide/example/MathService"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <!-- Requests and responses -->

```

```

<xsd:element name="add" type="xsd:int"/>
<xsd:element name="addResponse">
<xsd:complexType/>
</xsd:element>

<xsd:element name="subtract" type="xsd:int"/>
<xsd:element name="subtractResponse">
<xsd:complexType/>
</xsd:element>

<xsd:element name="getValueRP">
<xsd:complexType/>
</xsd:element>
<xsd:element name="getValueRPResponse" type="xsd:int"/>

<!-- Define your Resource Properties here. For example:-->

<xsd:element name="Value" type="xsd:int"/>
<xsd:element name="LastOp" type="xsd:string"/>

<xsd:element name="MathResourceProperties">
<xsd:complexType>
<xsd:sequence>
<xsd:element ref="tns:Value" minOccurs="1" maxOccurs="1"/>
<xsd:element ref="tns:LastOp" minOccurs="1" maxOccurs="1"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<!-- For Trust Negotiation Stuff-->

</xsd:schema>
</types>

<!--BEGIN MESSAGE DEFINITIONS - DO NOT MODIFY THIS BLOCK!!! -->
<message name="AddInputMessage">
<part name="parameters" element="tns:add"/>
</message>
<message name="AddOutputMessage">
<part name="parameters" element="tns:addResponse"/>
</message>

<message name="SubtractInputMessage">
<part name="parameters" element="tns:subtract"/>
</message>
<message name="SubtractOutputMessage">
<part name="parameters" element="tns:subtractResponse"/>
</message>

<message name="GetValueRPInputMessage">
<part name="parameters" element="tns:getValueRP"/>
</message>
<message name="GetValueRPOutputMessage">
<part name="parameters" element="tns:getValueRPResponse"/>
</message>

```

```
<!--END MESSAGE DEFINITIONS -->
```

```
<portType name="MathPortType" wsdlpp:extends="wsrpw:GetResourceProperty
  wsrpw:GetMultipleResourceProperties
  wsrpw:SetResourceProperties
```

wstn:TrustNegotiation

```
  wsrpw:QueryResourceProperties
  wsntw:NotificationProducer"
  wsrp:ResourceProperties="tns:MathResourceProperties">
```

```
<!--BEGIN OPERATION DEFINITIONS - DO NOT MODIFY THIS BLOCK!!! -->
```

```
<operation name="add">
  <input message="tns:AddInputMessage"/>
  <output message="tns:AddOutputMessage"/>
</operation>
```

```
<operation name="subtract">
  <input message="tns:SubtractInputMessage"/>
  <output message="tns:SubtractOutputMessage"/>
</operation>
```

```
<operation name="getValueRP">
  <input message="tns:GetValueRPInputMessage"/>
  <output message="tns:GetValueRPOutputMessage"/>
</operation>
```

```
</portType>
```

```
</definitions>
```

A.3 Service Deployment Descriptor WSDD

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<deployment name="defaultServerConfig" xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
<service name="ionut/services/MathService"
  provider="Handler" use="literal" style="document">
  <parameter name="className" value="g4mfs.impl.MathService"/>
  <parameter name="allowedMethods" value="*"/>
  <parameter name="handlerClass" value="org.globus.axis.providers.RPCProvider"/>
  <parameter name="scope" value="Application"/>
  <parameter name="providers" value=
  "GetRPPProvider GetMRPPProvider SetRPPProvider QueryRPPProvider
```

```
SubscribeProvider g4mfs.impl.gridpeertrust.net.server.TrustNegotiationProvider GetCurrentMessageProvider
```

```
"/>  
  
<parameter name="securityDescriptor"  
  value="share/schema/gt4ide/MathService/mysec.xml"/>  
<parameter name="fool-operations" value="func1 func2 func3"/>  
<wsdlFile>share/schema/gt4ide/MathService/Math_service.wsdl</wsdlFile>  
</service>
```

Appendix B

Credentials

B.1 X.509 End Entity Certificate

Certificate:

```
Data:
  Version: 3 (0x2)
  Serial Number: 2 (0x2)
  Signature Algorithm: md5WithRSAEncryption
  Issuer: O=Grid, OU=GlobusTest, OU=simpleCA-ionucomp, CN=Globus Simple CA
  Validity
    Not Before: May 17 22:09:43 2005 GMT
    Not After : May 17 22:09:43 2006 GMT
  Subject: O=Grid, OU=GlobusTest, OU=simpleCA-ionucomp, CN=Ionut Constandache
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    RSA Public Key: (1024 bit)
      Modulus (1024 bit):
        00:d0:3a:f3:88:5c:98:5a:21:86:0c:55:9f:26:10:
        17:59:b5:6e:70:c5:c4:6b:bc:84:35:c7:36:07:0f:
        99:b8:d9:13:96:e7:67:db:63:1b:b3:bb:83:d7:02:
        cc:62:91:cc:cf:af:38:30:7f:cf:9b:ac:3c:7f:c1:
        c3:06:54:6d:92:b1:e7:86:4e:00:f6:f2:4f:0c:07:
        d8:aa:b0:75:73:8b:f0:20:3c:26:be:88:08:71:6a:
        ee:b2:de:06:96:af:84:fb:6b:d3:8b:56:02:7d:d0:
        cf:de:20:47:37:63:29:22:7a:5c:d8:73:44:47:73:
        59:9a:f2:7d:10:bc:60:b2:8f
      Exponent: 65537 (0x10001)
  X509v3 extensions:
    Netscape Cert Type:
      SSL Client, SSL Server, S/MIME, Object Signing
    Signature Algorithm: md5WithRSAEncryption
      44:39:e7:a5:af:b5:48:d1:4a:d5:a7:3a:9b:82:e4:35:cc:8b:
      d1:8b:aa:f1:85:56:a9:5c:b3:e7:da:0a:11:1d:2f:c2:0a:89:
      75:1f:d0:1d:7c:e0:d9:c7:40:04:15:11:a1:30:c7:0c:3f:96:
      7b:2c:0e:83:f7:a2:3e:fe:aa:b0:a0:e5:d3:78:df:f3:7c:ae:
      65:bd:42:a3:48:b5:4e:eb:d4:0e:b5:b7:7d:7e:b1:50:7d:ac:
      c1:ea:7d:cb:3f:ce:72:b5:9e:3d:00:47:5f:21:c3:a1:4b:de:
      a1:1e:e2:b1:d1:3c:b8:ad:7c:d9:9e:9f:4f:7e:8c:68:61:b0:
      e2:07
-----BEGIN CERTIFICATE-----
MIICQzCCAaygAwIBAgIBAJANBgkqhkiG9w0BAQQFADBmMQ0wCwYDVQQKEwRHcm1k
MRMwEQYDVQQLEwPbG9iY291dG91dXNlcnV5dXN0MR0wGAYDVQQLEXFzaW1wbGVDQS1pb251Y29t
```

```

cDEZMBcGA1UEAxMQR2xvYnVzIFNpbXBsZSBDQTAEFw0wNTA1MTcyMjA5NDNaFw0w
NjA1MTcyMjA5NDNaMF0xDALBgNVBAoTBEhyaWQxEzARBgNVBAsTCkdsb2Jlc1Rl
c3QxGjAYBgNVBAsTEXNpbXBsZUNBLWlbnVjb2lwMRswGQYDVQQDExJb251dCBD
b25zdGFuZGFjaGUwZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBANA684hcmFoh
hgxVnyYQF1mlbnDFxGu8hDXHNgcPmbjzE5bnZ9tjG7O7g9cCzGKRzM+vODB/z5us
PH/BwwZUBzKx54ZOAPbyTwwH2KqwdXOL8CA8Jr6ICHFq7rLeBpavhPtr04tWAn3Q
z94gRzdjKSJ6XNhZREdzWZryfRC8YLKPAgMBAAGjFTATMBEGCWCsAGG+EIBAQQE
AwIE8DANBgkqhkiG9w0BAQQFAAOBgQBEOeelr7VI0UrVpzqbgUQ1zIvRi6rxhVap
XLPn2goRHS/CCo1lH9AdfODZx0AEFRGhMMcMP5Z7LA6D96I+/qqwoOXTeN/zfK5l
vUKjSLVO69QOtbd9frFQfazB6n3LP85ytZ49AEdfIcOhS96hHuKx0Ty4rXzZnp9P
foxoYbDiBw==
-----END CERTIFICATE-----

```

B.2 X.509 Proxy Certificate

Certificate:

Data:

Version: 3 (0x2)

Serial Number: 462472419 (0x1b90c4e3)

Signature Algorithm: md5WithRSAEncryption

Issuer: O=Grid, OU=GlobusTest, OU=simpleCA-ionucomp, CN=Ionut Constandache

Validity

Not Before: Aug 26 13:53:47 2005 GMT

Not After : Aug 27 01:58:47 2005 GMT

Subject: O=Grid, OU=GlobusTest, OU=simpleCA-ionucomp, CN=Ionut Constandache, CN=462472419

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

RSA Public Key: (512 bit)

Modulus (512 bit):

00:fc:c4:95:36:6a:c9:61:bb:e4:5f:74:8c:16:c2:

e9:d7:8c:fe:03:d6:b3:91:79:b3:29:80:69:f1:45:

5c:23:4c:b7:52:44:8d:71:42:7b:05:31:1f:8f:6a:

d1:8a:9a:5c:b3:6b:74:64:c1:0d:67:24:18:ea:2a:

dc:b0:e6:fc:25

Exponent: 65537 (0x10001)

X509v3 extensions:

Proxy Certificate Info Extension (old OID): critical

Proxy Policy:

Policy Language: GSI impersonation proxy

Policy: EMPTY

Signature Algorithm: md5WithRSAEncryption

49:f1:a9:4e:b3:08:cf:df:de:5f:79:e9:ee:49:8d:ae:ee:ec:

6f:2e:54:a4:49:94:82:7f:b8:36:0b:85:f0:cc:15:db:28:90:

b9:07:7e:f3:f0:92:84:82:a4:dd:b8:ff:36:09:2b:c6:45:48:

91:87:8c:f2:c9:27:b7:7a:91:18:78:50:da:17:46:0f:5d:c8:

47:c1:08:a8:68:c7:50:65:a0:00:c2:e0:30:bd:4c:a3:73:79:

e9:d0:c8:ba:9a:7c:c2:72:c3:9a:18:4b:50:e7:ae:86:a3:82:

55:5b:8d:c1:a6:29:cf:8f:22:07:b6:01:d1:d9:c1:b1:1a:c2:

63:3a

B.3 SAML Assertion

<Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion"

AssertionID="19538c20-f2d2-11d9-9434-bf358f7e94c9"

IssueInstant="2005-07-12T12:40:16Z"

Issuer="O=Grid,OU=GlobusTest,OU=simpleCA-ionucomp,CN=Globus Simple CA"

```

MajorVersion="1" MinorVersion="0">
<Conditions NotBefore="2005-07-12T12:40:16Z" NotOnOrAfter="2005-07-12T12:40:16Z">
</Conditions>
<AuthorizationDecisionStatement Decision="Permit" Resource="egov|MyProject">
  <Subject>
    <NameIdentifier Format="#X509SubjectName" NameQualifier=
      "O=Grid,OU=GlobusTest,OU=simpleCA-ionucomp,CN=Ionut Constandache">
      /O=Grid/OU=GlobusTest/OU=simpleCA-ionucomp/CN=Ionut Constandache
    </NameIdentifier>
    <SubjectConfirmation>
      <ConfirmationMethod urn:oasis:names:tc:SAML:1.0:am:X509-PKI>
      </ConfirmationMethod>
    </SubjectConfirmation>
  </Subject>
  <Action Namespace="membership">isMember</Action>
</AuthorizationDecisionStatement>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod
      Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315">
    </ds:CanonicalizationMethod>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1">
    </ds:SignatureMethod>
    <ds:Reference URI="">
      <ds:Transforms xmlns:signs="urn:oasis:names:tc:SAML:1.0:assertion">
        <ds:Transform Algorithm="http://www.w3.org/2002/06/xmldsig-filter2">
<dsig-xpath:XPath xmlns:dsig-xpath="http://www.w3.org/2002/06/xmldsig-filter2"
  Filter="intersect">
  here()/ancestor::signs:Assertion[1]
</dsig-xpath:XPath>
<dsig-xpath:XPath xmlns:dsig-xpath="http://www.w3.org/2002/06/xmldsig-filter2"
  Filter="subtract">
  here()/ancestor::ds:Signature[1]
</dsig-xpath:XPath>
        </ds:Transform>
        <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
<ec:InclusiveNamespaces
  xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"
  PrefixList="code ds kind rw saml samlp signs #default xsd xsi">
</ec:InclusiveNamespaces>
        </ds:Transform>
      </ds:Transforms>
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
      </ds:DigestMethod>
      <ds:DigestValue>
kNW9+zRs3pBcCn/RLh9Z/V76KMY=
      </ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>
    d81zL+kWQ9nHJHoLD3KstBthWfYgJvRtyht28XsgeR1bRku+848ocG291rjalzgzXTGrEo3+z9pej
    BhkiN2VvwFnDKLsahhcTZ/ZvDSnyZVh3V5h+Glb17Dhik1DlulrkdcTVoh1Zc/iJHpWcvI5ktY/B
    RfEn7qthGtyqz3f19Us=
  </ds:SignatureValue>
  <ds:KeyInfo>
    <ds:X509Data>
      <ds:X509Certificate>
MIICPTCCAaagAwIBAgIBBDANBgkqhkiG9w0BAQQFADBBMQ0wCwYDVQQKEwRhcmlkMRRMwEQYDVQQQL
EwpHbG9idXNUZXN0MR0wGAYDVQQLEFZaW1wbGVDQ01pb251Y292cDEZMBCGAlUEAxMQR2xvYnVz
IFNpbXBsZSBDbQTAeFw0wNTA3MDYxMzQ4NTVaFw0wNjA3MDYxMzQ4NTVaMFcxDTALBgNVBAoTBEdy
aWQxARBgNVBAsTCkdsb2Jlc1Rlc3QxGjAYBgNVBAsTEkxNpbXBsZUNBLW1vbnVjb21wMRUwEwYD
VQQDEWxjYXNvaW9udWNvbXAwdz8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMfUanCHF594YKco

```

```
UGxI/zgKv1KGjGd5PbOkyviGM8z8yhE9ut12+PgXjuT/JqE9S7pP48eEV5MWYwd1ehs8bIbY+g57
PV0gfFSDF7txW6EtAdonM7VuFm2YdWw1DJ76QRP0wUWWbWwND7+PDA72kgNSCMfU8PLpSk3minlI
MMKHAgMBAAGjFTATMBEGCWCGSAGG+EIBAQQEAWIE8DANBqkqhkiG9w0BAQQFAAQBKnlXl3YB1
W+Fo37oA7C+lLDlrPuS/qKUX8ivxFBkYT2M9SnmS8C943s7rbP/Wws3CZ39bUWAKorAK9GLZGn21
iciaK9M/+IxYXayYg/GsNnz2gfsLnuWUKUY5U14LiPO+wRBQME2bmWydlu/Q8Ue4hlsgVc0ki45f
c76ZbIwH4A==
</ds:X509Certificate>
</ds:X509Data>
</ds:KeyInfo><
/ds:Signature>
</Assertion>
```

Appendix C

Code

```
package g4mfs.impl.gridpeertrust.net.server;

import ro.pub.egov.linux.ionut.TrustNegotiation_wsdl.GetNotificationURI;
import ro.pub.egov.linux.ionut.TrustNegotiation_wsdl.GetNotificationURIResponse;
import ro.pub.egov.linux.ionut.TrustNegotiation_wsdl.NegotiateTrust;
import ro.pub.egov.linux.ionut.TrustNegotiation_wsdl.NegotiateTrustResponse;

import g4mfs.impl.gridpeertrust.util.CertificateExtractor;
import g4mfs.impl.gridpeertrust.util.ConverterClass;
import g4mfs.impl.gridpeertrust.util.InitializationHolder;
import g4mfs.impl.gridpeertrust.util.InitializeNegotiationEngine;
import g4mfs.impl.gridpeertrust.util.LocalPeer;
import g4mfs.impl.gridpeertrust.util.SuperMessage;
import g4mfs.impl.gridpeertrust.util.SyncQueue;
import g4mfs.impl.gridpeertrust.util.TopicHelper;
import g4mfs.impl.gridpeertrust.wrappers.NotificationSendWrapper;
import g4mfs.impl.gridpeertrust.wrappers.SendHelper;
import g4mfs.impl.gridpeertrust.wrappers.SendWrapper;
import g4mfs.impl.org.peertrust.net.Message;
import g4mfs.impl.MathQNames;

import java.security.cert.X509Certificate;
import java.util.Date;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Set;

import javax.security.auth.Subject;
import javax.xml.namespace.QName;
import javax.xml.rpc.handler.MessageContext;

import org.globus.gsi.jaas.GlobusPrincipal;
import org.globus.gsi.jaas.JaasSubject;
import org.globus.wsrp.Constants;
import org.globus.wsrp.Resource;
import org.globus.wsrp.ResourceContext;
import org.globus.wsrp.Topic;
import org.globus.wsrp.TopicList;
import org.globus.wsrp.TopicListAccessor;
import org.globus.wsrp.impl.SimpleTopic;
import org.globus.wsrp.impl.security.SecurityManagerImpl;
```

```

import org.globus.wsrfl.impl.security.authentication.ContextCredential;
import org.globus.wsrfl.impl.security.util.AuthUtil;
import org.globus.wsrfl.security.SecurityManager;
import org.ietf.jgss.GSSCredential;
import org.ietf.jgss.GSSException;

/**
 * @author ionut constandache ionut_con@yahoo.com
 * Provides the functionality of Trust Negotiation if a provider for a service
 *
 */

public class TrustNegotiationProvider
{

    Hashtable ht;

    private static final String LOG_CONFIG_FILE =
"/home/ionut/PeertrustFiles/demoClient/.logconfig" ;

    SyncQueue syncQueue = new SyncQueue();
    // the messages received from the client are put in this queue. The message is popped out
    // from the queue in GridNetServer and returned by the listen method
    private SendHelper sendHelper = new SendHelper();

    InitializeNegotiationEngine initializeNegotiationEngine; //for initializing the negotiation engine
    ClientManager clientManager;

    public TrustNegotiationProvider()
    {
        ht = new Hashtable();
        init();
    }

    // init the whole Trust Negotiation Engine
    public void init()
    {
        initializeNegotiationEngine = new InitializeNegotiationEngine(
"hpclinuxcluster", "/home/globus/PeertrustFiles/demoServer/entities.dat2",
syncQueue, sendHelper, "demo",
"/home/globus/PeertrustFiles/demoServer/", "demoPolicies.eLearn1",
"minervagui.mca", false, false);
        InitializationHolder.setInitializeNegotiationEngine(initializeNegotiationEngine);

        clientManager = new ClientManager();
        InitializationHolder.setClientManager(clientManager);
    }

    public NegotiateTrustResponse negotiateTrust(NegotiateTrust request)
    {

        String[] a = request.getTrace();

        Message mesg = ConverterClass.negotiateTrustToPtMessage(request);

```

```

syncQueue.push(mesg);
//put the received message in the gridNetServeQueue
//for being processed by the negotiation engine

NegotiateTrustResponse ntr = new NegotiateTrustResponse();
return ntr;

}

public GetNotificationURIResponse getNotificationURI(GetNotificationURI u)
{
    QName response = null; // = MathQNames.RP_TRUST_NEGOTIATION;
    NotificationSendWrapper nsw = new NotificationSendWrapper();
    Topic negotiationTopic = null;
    TopicList topicList = null;
    CertificateExtractor certificateExtractor = null;

    try
    {

        org.apache.axis.MessageContext messageContext =
        org.apache.axis.MessageContext.getCurrentContext();
        Subject sub = (Subject) messageContext.getProperty
        (org.globus.wsrfl.impl.security.authentication.Constants.PEER_SUBJECT);

        Set pc = sub.getPublicCredentials();

        Iterator it = pc.iterator();

        String res;

        int i = 0;
        while(it.hasNext())
        {
            Object o = it.next();
            X509Certificate[] tab = (X509Certificate[]) o;

            if(i == 0 )
            {
                certificateExtractor = new CertificateExtractor(tab);
            }
            i++;
        }

        catch(Exception e)
        {
            e.printStackTrace();
        }

        String subjectDN = certificateExtractor.getSubjectDN();
        String issuerDN = certificateExtractor.getIssuerDN();
        Date expirationDate = certificateExtractor.getExpirationDate();

    }

    try
    {
        Resource resource = ResourceContext.getResourceContext().getResource();
    }

```

```

    topicList = ((TopicListAccessor)resource).getTopicList();
}
catch(Exception ex)
{
    ex.printStackTrace();
}
//add a new topic for this client

//compute the local part for a QName used by the new topic for the client
String localPart = TopicHelper.getUniqueLocal(subjectDN,issuerDN);

// obtain the namespace of the service
response = new QName(MathQNames.NS,localPart);
negotiationTopic = new SimpleTopic(response); //new topic for client

topicList.addTopic(negotiationTopic); //test if a topic for this client already exists

//add subjectDN issuerDN topic date to ClientManager
clientManager.addTopicClient(subjectDN, issuerDN, negotiationTopic, expirationDate);

nsw.setNegotiationTopic(negotiationTopic);
// associate this sendwrapper with the local part of the notification topic URI
sendHelper.putSendWrapper(response.getNamespaceURI()+response.getLocalPart(),nsw);

LocalPeer localPeer = initializeNegotiationEngine.getMetaInterpreter().getLocalPeer();

localPeer.add(response.getNamespaceURI()+response.getLocalPart(),
"https://127.0.0.1:8443/wsrf/services/ionut/services/MathService");

String callerIdentity = SecurityManager.getManager().getCaller();

GetNotificationURIResponse resp = new GetNotificationURIResponse();
resp.setLocalPart(response.getLocalPart());
resp.setNamespaceURI(response.getNamespaceURI());
return resp;
}
}

```

```

/*
 * Created on Jun 5, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package g4mfs.impl.gridpeertrust.net.client;

import g4mfs.impl.gridpeertrust.util.SyncQueue;
import g4mfs.impl.gridpeertrust.wrappers.MessageDescription;
import g4mfs.impl.org.peertrust.net.Message;
import g4mfs.impl.org.peertrust.net.Peer;

import java.util.ArrayList;

import javax.xml.namespace.QName;

import org.apache.axis.message.addressing.EndpointReferenceType;

/**
 * @author ionut constandache ionut_con@yahoo.com
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class GridClientTrustNegotiation
{

    boolean finish = false; // if the negotiation process has finished
    boolean succeeded = false; // if the negotiation process has succeeded
    boolean isSleeping = false; // if the thread using this object is sleeping
    GridClientNotificationThread notificationThread = new GridClientNotificationThread();
    SyncQueue queue;
    QName notificationQName;

    public GridClientTrustNegotiation()
    {

    }

    public synchronized boolean isFinished()
    {
        return finish;
    }

    public synchronized void setFinished()
    {
        finish = true;
        if(isSleeping)
        {
            notify(); // in case the thread is sleeping
        }
    }

    public synchronized boolean isSuccess()
    {
        return succeeded;
    }
}

```

```

}

public synchronized void setSuccess(boolean flag)
{
    succeeded = flag;
    setFinished(); // it is also finished
}

public void setNotificationQName(String namespaceURI, String localPart)
{
    notificationQName = new QName(namespaceURI, localPart);
}

public void setSyncQueue(SyncQueue sq)
{
    queue = sq;
}

public boolean checkPolicyException(Exception e)
{
    String str = e.getMessage();
    if(str.indexOf("e clar ceva in neregula")>-1)
    {
        return true;
    }
    else return false;
}

public void startListening(String serviceURI)
{
    notificationThread.setServiceURI(serviceURI);
    notificationThread.setNotificationURI(notificationQName);
    notificationThread.setSyncQueue(queue);
    notificationThread.start();
}

public void startListeningEPR(EndpointReferenceType epr)
{
    notificationThread.setServiceEPR(epr);
    notificationThread.setNotificationURI(notificationQName);
    notificationThread.setSyncQueue(queue);
    notificationThread.start();
}

public synchronized boolean negotiate()
{
    while(true)
    {
        if(isFinished())
        {
            isSleeping = false;
            return isSuccess();
        }
        else
        {
            try
            {
                isSleeping = true;

```

```
wait();
isSleeping = false;
if(isFinished() || isSuccess())
{
return isSuccess();
}
}
catch(InterruptedException ex)
{
ex.printStackTrace();
}
}
}
```

```

package g4mfs.impl.gridpeertrust.wrappers;

/**
 * @author ionut
 * this interface is used to abstract the process of message sending to another peer
 * as the ways to interact with a peer may be unknown
 * The intent use is to provide a SendWrapper for each service that might be contacted by the client
 */
public interface SendWrapper
{

/**
 * @param peer peer is going to be of org.peertrust.net.Peer type and it would
 * be transformed in the Peer type used by the
 * destination port
 */
public void setPeer(Object peer);
/**
 * @param msg msg is going to be of org.peertrust.net.Message type and
 * it would be transformed in the message type
 * used by the destination port
 */
public void setMessage(Object msg);
public Object getPeer();
public Object getMessage();
public void sendMessage();
}

```

```

package g4mfs.impl.gridpeertrust.net.client;

import g4mfs.impl.gridpeertrust.util.ConverterClass;
import g4mfs.impl.gridpeertrust.util.SuperMessage;
import g4mfs.impl.gridpeertrust.util.SyncQueue;
import g4mfs.impl.org.peertrust.net.Answer;
import g4mfs.impl.org.peertrust.net.Message;

import java.util.List;

import javax.xml.namespace.QName;

import org.apache.axis.message.addressing.EndpointReferenceType;
import org.globus.wsrp.NotificationConsumerManager;
import org.globus.wsrp.NotifyCallback;
import org.globus.wsrp.WSNConstants;
import org.globus.wsrp.encoding.ObjectDeserializer;
import org.oasis.wsn.NotificationProducer;
import org.oasis.wsn.Subscribe;
import org.oasis.wsn.TopicExpressionType;
import org.oasis.wsn.WSBaseNotificationServiceAddressingLocator;
import org.w3c.dom.Element;

import ro.pub.egov.linux.ionut.TrustNegotiation_wsdl.TrustNegotiationNotificationMessageType;

/**
 * @author ionut constandache ionut_con@yahoo.com
 *
 */
public class GridClientNotificationThread extends Thread implements NotifyCallback
{
    String serviceURI;
    EndpointReferenceType epr;
    QName notificationURI;
    SyncQueue queue;
    GridClientTrustNegotiation gridClientTrustNegotiation;
    boolean exitFlag = false;

    public GridClientNotificationThread()
    {
    }

    public GridClientNotificationThread(GridClientTrustNegotiation gridClientTrustNegotiation)
    {
        this.gridClientTrustNegotiation = gridClientTrustNegotiation;
    }

    public void setNotificationURI(QName notificationURI)
    {
        this.notificationURI = notificationURI;
    }

    }

    public void setSyncQueue(SyncQueue q)
    {
        queue = q;
    }
}

```

```

public void setGridClientTrustNegotiation(GridClientTrustNegotiation gridClientTrustNegotiation)
{
    this.gridClientTrustNegotiation = gridClientTrustNegotiation;
}

public void deliver(List topicPath, EndpointReferenceType producer, Object message)
{
    try
    {
        TrustNegotiationNotificationMessageType notif;
        notif = (TrustNegotiationNotificationMessageType) ObjectDeserializer.toObject((Element) message, TrustNe
        if(notif != null)
        {

String[] a = notif.getTrace();

Message mesg = ConverterClass.trustNegotiationNotificationMessageToPtMessage(notif);
if (mesg instanceof Answer)
{
    if (((Answer) mesg).getStatus() == Answer.LAST_ANSWER) // negotiation is finished with success
    {
        exitFlag = true;
        gridClientTrustNegotiation.setSucces(true);
    }
    else
    if (((Answer) mesg).getStatus() == Answer.FAILURE) // negotiation is finished without success
    {
        exitFlag = true;
        gridClientTrustNegotiation.setSucces(false);
    }
}
queue.push(mesg);
}

}
catch(Exception e)
{
    e.printStackTrace();
}

}

public void setServiceURI(String s)
{
    serviceURI = s;
}

public void setServiceEPR(EndpointReferenceType s)
{
    epr = s;
}

public void run()
{

```

```

try
{

// the notificationconsumer sets up an endpoint where notifications will be delivered
NotificationConsumerManager consumer;
consumer = NotificationConsumerManager.getInstance();
consumer.startListening();
EndpointReferenceType consumerEPR = consumer.createNotificationConsumer(this);

//create the request to the remote Subscribe() call
Subscribe request = new Subscribe();

// the notification must be delivered using Notify operation
request.setUseNotify(Boolean.TRUE);

//Indicate what the client's EPR is
request.setConsumerReference(consumerEPR);

//The TopicExpressionType specifies what topic we want to subscribe to
TopicExpressionType topicExpression = new TopicExpressionType();
topicExpression.setDialect(WSNConstants.SIMPLE_TOPIC_DIALECT);
topicExpression.setValue(notificationURI);
request.setTopicExpression(topicExpression);

//Get a reference to the NotificationProducer portType
WSBaseNotificationServiceAddressingLocator notifLocator =
    new WSBaseNotificationServiceAddressingLocator();
//EndpointReferenceType endpoint = new EndpointReferenceType();
//endpoint.setAddress(new Address(serviceURI));
//NotificationProducer producerPort = notifLocator.getNotificationProducerPort(endpoint);

NotificationProducer producerPort = notifLocator.getNotificationProducerPort(epr);

producerPort.subscribe(request);

while(true)
{

if(exitFlag) // negotiation is finished stop thread
{
return;
}

try
{
Thread.sleep(30000);
}
catch(Exception e)
{
e.printStackTrace();
}
}
}
}

```

```
catch(Exception e)
{
e.printStackTrace();
}
}
```

```

package g4mfs.impl.gridpeertrust.net.server;

import g4mfs.impl.gridpeertrust.util.InitializationHolder;
import g4mfs.impl.gridpeertrust.util.InitializeNegotiationEngine;

import java.security.cert.X509Certificate;
import java.util.Date;
import java.util.Iterator;
import java.util.Set;
import java.util.Vector;

import javax.security.auth.Subject;
import javax.xml.namespace.QName;
import javax.xml.rpc.handler.MessageContext;

import org.globus.gsi.jaas.GlobusPrincipal;
import org.globus.gsi.jaas.SimplePrincipal;
import org.globus.wsrp.impl.security.authorization.exceptions.CloseException;
import org.globus.wsrp.impl.security.authorization.exceptions.InitializeException;
import org.globus.wsrp.impl.security.authorization.exceptions.InvalidPolicyException;
import org.globus.wsrp.impl.security.authorization.PDP;
import org.globus.wsrp.impl.security.authorization.PDPConfig;
import org.w3c.dom.Node;

import java.util.StringTokenizer;

/**
 * @author ionut constandache ionut_con@yahoo.com
 *
 */
public class InterceptorPDP implements PDP
{
    Vector permittedOperations = new Vector();
    InitializeNegotiationEngine initializeNegotiationEngine = InitializationHolder.initializeNegotiationEng
    ClientManager clientManager = InitializationHolder.clientManager;

    public String[] getPolicyNames()
    {
        return new String[0];
    }

    public Node getPolicy(Node query) throws InvalidPolicyException
    {
        return null;
    }

    public Node setPolicy(Node query) throws InvalidPolicyException
    {
        return null;
    }

    public boolean isPermitted(Subject peerSubject, MessageContext context, QName operation)
    throws MyAuthorizationException
    {
        if(clientManager == null)
        {
            if(InitializationHolder.clientManager != null)

```

```

{
clientManager = InitializationHolder.clientManager;

}
}

String subjectDN = null;
String issuerDN = null;
String clientOperation = operation.getLocalPart();

if(operation.toString().endsWith("negotiateTrust")
|| operation.toString().endsWith("getNotificationURI"))
{

return true;
}
else
{
Set pc = peerSubject.getPublicCredentials();
Iterator it = pc.iterator();
int i = 0;
while(it.hasNext())
{
Object o = it.next();

if(i == 0)
{
X509Certificate[] tab = (X509Certificate[]) o;

subjectDN = tab[tab.length-1].getSubjectDN().getName();
issuerDN = tab[tab.length-1].getIssuerDN().getName();
Date expirationDate = tab[0].getNotAfter();

}
i++;
}

if(clientManager != null)
{
if(clientManager.isAuthorized(subjectDN, issuerDN,clientOperation))
{
return true;
}
else
{
MyAuthorizationException e = new MyAuthorizationException("negotiation exception");
e.setReason("negotiation exception");
throw e;
}
}
else
{

MyAuthorizationException e = new MyAuthorizationException("negotiation exception");
throw e;
}
}

```

```
}

//return false;
}

public void initialize(PDFConfig config, String name, String id) throws InitializeException
{
String value = (String) config.getProperty(name,"operations");

StringTokenizer st = new StringTokenizer(value," ");
String s;
while(st.hasMoreTokens())
{
s = st.nextToken();
permittedOperations.add(s);
}

return;
}

public void close() throws CloseException
{
return;
}
}
```