R | R | Z | N

Leibniz Universität Hannover

Gottfried Wilhelm Leibniz Universität Hannover
Regionales Rechenzentrum für Niedersachsen
Fachgebiet Distributed Virtual Reality
Lehrgebiet Rechnernetze

Masterarbeit
im Studiengang Informatik (M. Sc.)

Implementation of a policy-based authorization for gLite Compute Services

Verfasser:	B. Sc. C. Kunz
Erstprüfer:	Prof. Dr.-Ing. C. Grimm
Zweitprüferin:	Prof. Dr.-Ing. G. v. Voigt
Betreuer:	Dipl.-Ing. S. Piger
Datum:	14. Juni 2007

Hannover, den 14.06.2007
Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Christopher Kunz

Abstract

Grid computing allows the scientific community to share resources across network and organization boundaries, increasing the possibilities for researchers worldwide. The gLite framework is used by the EGEE initiative to build a standardized grid infrastructure for scientific institutions worldwide.

Security is an important aspect of grid frameworks: Not only are confidentiality and integrity necessary, but there are also high amounts of computing power involved that need to be accounted to the ones actually consuming it. Attackers posing as a legitimate grid user could thus consume computing and data resources. With the use of mutual authentication through certificates and delegation of rights by means of proxy certificates, this problem can partially be solved. By obtaining a proxy certificate and its private key, an unauthorized third party could still pose as a legitimate user and consume resources on their behalf.

This thesis presents a solution to this problem by enabling grid users to restrict their own execution rights to the minimum necessary. By embedding an XACML policy in a X.509 extension to their proxy certificate, they can specify a set of Job IDs. The CE will then only allow the jobs associated with these identifiers to be passed to the worker nodes for computation, thwarting an attacker's intent to consume processing resources.

Contents

List of Figures

List of Abbreviations

ASN.1	Abstract Syntax Notation number One
CA	Certificate Authority
CE	Computing Element
CERN	Conseil Européen pour la Recherche Nucléaire, now: Organisation Européenne pour la Recherche Nucléaire
CREAM	Computing Resource Execution And Management
CRL	Certificate Revocation List
CSR	Certificate Signing Request
DAG	Direct Acyclic Graph
DN	Distinguished Name
EEC	End Entity Certificate
EGEE	Enabling Grids for E-SciencE
EUGridPMA	European Policy Management Authority for Grid Authentication
GSI	Grid Security Infrastructure
ICE	Interface to Cream Environment
JDL	Job Description Language
LCAS	Local Centre Authorization Service
LHC	Large Halon Collider
LPDP	Local Policy Decision Point
LRMS	Local Resource Management System

OASIS	Organization for the Advancement of Structured Information Standards
OCSP	Online Certificate Status Protocol
OID	Object Identifier
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
PKI	Public Key Infrastructure
UBP	User-Based Policy
URI	Uniform Resource Indicator
VO	Virtual Organization
VOMS	Virtual Organization Membership Service
WMS	Workload Management System
XACML	eXtensible Access Control Markup Language

Chapter 1

Introduction

1.1 About Grid Computing

The idea of a global computing *Grid*, a geographically distributed supercomputing structure, is not a new one. Sharing of computing resources occurred as early as the 1970s, but only in the last decade evolved into the standardized concept that is currently regarded as *Grid computing*.

Basically, the phrase "computing Grid" stems from a comparison with the power grid which can be seen as an ubiquitous and reliable source of electrical power. However, the actual power plant providing the power to the end user is normally kept hidden from them as it is not an indicative of quality or fitness for a special purpose. Similarily, the idea of a computing Grid is seen as an ubiquitous source of computing and data resources, shared over geographically and organizationally distributed resource providers, available to everyone who wishes to participate.

Subsequently, the increasing digitalization of scientific content, especially the output of experiments and lab procedures, led to a massive increase in data volume. Especially physicists have to analyze large amounts of raw data for single experiments, e.g., in particle physics. Distributing the workload onto multiple computing resources at different organisations became too error-prone without a standardized workflow. Therefore, the so-called "fathers of the Grid" created the Globus toolkit that catered to the requirements on the three-point Grid checklist authored by Ian Foster [Fos02]:

1. *"The Grid coordinates resources that are not subject to a centralized control"* – it integrates between different organizational divisions or entities, taking care of security, accounting and other issues.

2. *"The Grid uses standard, open, general-purpose protocols and interfaces."* In the Globus Toolkit or – more specific to the topic of this thesis – the gLite middleware, established protocols are used to establish inter-component communication, security and accountability.

3. *"The Grid delivers non-trivial quality of service"* so that an effect of emergence is created: the Grid is more useful than the sum of all its parts.

All modern Grid toolkits allow more than just batch processing of individual jobs – that would be yet another multi-user computing system. They incorporate job control, ensure accountability and reliability and distribute the workload across domain boundaries. Typically, domains in current supercomputing are those of one single institution or even one branch at one institution. For Grids, they are aggregated into so-called Virtual Organizations that combine their computing power even though they are usually not part of the same company, scientific institution or even jurisdiction. These Virtual Organizations, or VOs, are composed of resource providers and users – both groups can, of course, overlap.

Although Grid computing has been under intensive scrutiny and development within the scientific community, "computing wall jacks" are still far from being a mainstream end-user product. Scientists remain the largest target audience for the Grid. Various projects and initiatives, the EGEE being one of them, aim at providing organizations like the CERN with a highly scalable, very large computing infrastructure for analysis and distribution of their experimental results. Most notably, the LHC creates a large amount of output during experiments, aggregating to about 15 petabytes of data per year [Jon05]. While data storage and retrieval pose independent non-trivial problems, much of the experimental output is classified and experiments cannot easily be repeated. Thus, data integrity and confidentiality have to be provided (see [Gro06] for an approach to policy-based data access in Grids).

In addition to storing the data, their computational evaluation is the main goal of Grid computing. The raw input has to be transformed into meaningful results via computing jobs. These jobs, executed on as many independent nodes as necessary, will commonly read data, manipulate or even remove it – thus authentication and authorization are necessary. Since jobs incur costs (albeit those might only be virtual

and not actually of any monetary value), they also need to be accountable to a specific user. Using certificates as a time-limited mandatory means of authentication, unauthorized access to the computing resources can be limited, but if credentials are wrongfully duplicated, unauthorized third parties could consume computing resources on a legitimate user's behalf. This thesis presents a concept to limit proxy credential usage by implementing a policy-based self-restriction mechanism.

1.2 About this Thesis

In current Grid solutions, most notably the gLite middleware, jobs are prepared by the submitter on a client machine, often called the User Interface (UI). Using the Job Description Language (JDL), jobs are parametrized, necessary executable and helper files identified and computing instructions are given to the Grid middleware. Additionally, data sources and sinks are defined. After the job description is completed, the job is submitted to the middleware system. Upon submission, each job is assigned one or more job identifiers (or job IDs). Since jobs can contain sub-jobs dependent on them, job IDs can have a parent identifier (or vice versa: A job ID can have "children"). Additionally, a set of independent jobs can be submitted with a single job description.
Every job that is submitted to the Grid is authenticated and authorized using credentials that typically have a relatively short lifespan. Using these credentials, all parts of the job computation chain can validate the user who submitted the job in question, account resources spent and make sure the job input and outcome can only be accessed by that specific submitter.

If a third party gets a hold of the credential that was used to submit a job, they could easily submit more jobs to the Grid and thus consume computing power that they are not being billed for. However, there is a limited timeframe for this interception and unauthorized job submission, as these credentials typically last for twelve hours or less, rendering them useless afterwards. There is no revocation scheme in place for these so-called *proxy credentials* – once issued, they cannot be revoked even if unauthorized use by a third party has been observed.

This thesis will present a method for self-restriction of execution permissions based on user-based policies (UBP). Using the job ID as a unique identifier, the Grid user can specify one or more jobs that may be computed using a specific credential set – no other jobs will be accepted by the Grid management components.

In the second chapter, an overview over existing security measures in the gLite middleware, especially the certificate-based security infrastructure, will be given and the current security situation – including possible vulnerabilities to unauthorized third parties – will be evaluated. Also, the motivation for an execution policy will be shown by detailing possible attack scenarios against users on a Grid, be it from the inside or outside. It will be made clear that in the current gLite infrastructure stealing credentials and submitting unauthorized jobs is theoretically possible – including jobs that jeopardize the worker nodes' security.

In chapter 3, the principle of user-based policies and the descriptive language XACML used to formulate UBPs will be introduced.

Chapter 4 will give a detailed look on job management in the Grid with the job ID being the most important identifier for a Grid job. The peculiarities of Directed Acyclic Graphs in job management will also be discussed.

The fifth chapter will be dedicated to the solution at hand and the current prototype implementation. There, the conceptual reasons and practical implications of implementing the user-based policies will be presented and detailed. Also, an extension to the MyProxy credential repository will be discussed.

In the last chapter, a conclusion and ideas for future work will be shown. The appendices include a comprehensive installation guide for the software developed as a part of this thesis.

Chapter 2

gLite Security Architecture

The term *security architecture* is defined as *"a set of features and services that tackles a set of security requirements and can handle a set of use cases"* [EGEE-Sec]. In order to be sufficient for use in a Grid environment, a security architecture has to handle secure and confidential access to remote resources like worker or storage nodes. Additionally, it must provide protection and monitoring to avoid – or at least notice – exploitation of resources by malicious users or unauthorized third parties.

As with any security concept, the gLite security architecture is under constant development to meet the changing requirements that arise with new projects and the increasing ability of attackers while still maintaining the basic security requirements of availability, integrity and confidentiality.

While availability and integrity of Grid services are out of the scope of this document, confidentiality is the main concern of this thesis. Keeping Grid jobs and all associated information confidential means that they are only passed along to authorized parties that have been authenticated as such. Thus, there are two tasks that have to be completed to ensure confidentiality:

1. Reliable *authentication* must be provided. In the gLite middleware, this is accomplished by a PKI.
2. There must be measures to provide resources only to entities that have been *authorized* to make use of them. Authorization (or "AuthZ") is currently mainly provided by means of Gridmap files or LCMAPS and the LCAS service on the CE.

2.1 Public Key Infrastructure

The GSI (see 2.2 below) relies heavily on a hierarchic Public Key Infrastructure (PKI) to provide a secure means of decentralized authentication. The certificate infrastructure is based on one or more universally trusted Certificate Authorities (CAs) that issue certificates to Grid components and users. These certificates are issued when a Grid participant (or their agent) submits a certificate signing request to the CA and passes along credentials that prove their identity. Upon inspection of this identity proof, the CA issues a document that can be presented to third parties instead of the actual identity proof - a certificate. Third parties who are provided with an entity's certificate inspect and accept it if it bears a trusted CA's signature. Their trust in the CA is projected onto the certificate, giving it a trusted state too.

2.1.1 X.509 Certificates

A *certificate* is a set that consists of a public key, meta information and the issuer's valid signature. The most important standard for certificates in Grids is ITU-T Recommendation X.509 [ITU05]. Based on this recommendation, [RFC2459] details the implementation of PKIs. A certificate declares that an independent trusted authority has inspected its owners identity and verified that the owner is who they claim to be. It can be compared to a federally issued identity card that bears the owner's name and signature.

Similar to a real ID card, certificates can only be useful if issued by an authority that is trusted by all peers. While ID cards can supposedly not be forged due to technical protections in the manufacturing process, a digital certificate is protected by the issuer's cryptographic signature, which (a universal assumption made in public-key cryptography) cannot be forged either.

When a certificate holder wishes to authenticate any information via an untrusted communication channel, they would sign the information to be transmitted using the private key that corresponds to the public key in their certificate. Like any private key, this key has to be kept secret and must not be passed along to third parties. The certificate, however, is being included with the signed information. The communication partner can then use the public key to verify the signature and the CA certificate to verify the sender's certificate.

Thus, a certificate serves two distinct purposes. It is used to authenticate *who* sent something and – combined with a signature on the information itself – that the information *has not been changed* after it has been sent. This ensures accountability for the sender and is a vital part of Grid security.

Certificates provide the owner with a Distinguished Name (DN) that must be unique to that entity in the context of the issuing CA. The CA must never assign an identical DN to any other entity, even after the original certificate has expired. Consequently, it is possible to use the DN as a unique identifier for any persistent entity in a Grid governed by a single CA. However, if several CAs issue certificates for one authentication domain, this assumption is no longer valid. See section 2.1.3 for more detail on CAs.

A certificate issued to a Grid entity by its governing CA is called "End-Entity Certificate" (EEC). An EEC's validity depends on the CA's regulations but for the Grid context it can be assumed to be no less than 12 months. For this thesis, the combination of the EEC and its private key will be called "end-entity credentials".

It is important to note that in a PKI whoever has access to the EEC credentials can impersonate their owner, since they can present the certificate and digitally sign data with the private key associated to it. They can perform the mutual authentication protocol (see below) for the certificate's complete lifetime. Thus, the private key connected to an EEC must never become known to any third party whatsoever.

2.1.2 X.509 certificate extensions

In addition to standard metadata included in certificates (like subject, issuer or the certificate's unique serial number), the third version of the X.509 recommendation defines a possibility to extend the certificate with additional data. Several frequently-used extensions are defined in [RFC2459], section 4.2. These standard extensions are useful for maintenance and revocation purposes, because they allow for the inclusion of CRL distribution points in the actual certificate, the restriction of a certificate to specific purposes and other common tasks.

Apart from these standard extensions, custom ones can be used to further extend the scope of information included in the certificate. These extensions can include arbitrary data and are assigned unique Object Identifiers (OID) as defined in X.680 [ITU02]. OIDs follow a tree-like hierarchy that goes from a root to different branches and each leaf uniquely identifies an attribute. An example for this would be the OID

`.iso.org.dod.internet.mgmt.mib-2.system.sysLocation.0` or its numeric equivalent `.1.3.6.1.2.1.1.6.0` which defines the field for the physical location of a computer system.
Base OIDs are assigned to organisations which can then extend them by appending new branches to them. In the example, the base OID would be `.1.3.6` for the US Department of Defense.

Certificate extensions contain their values in an ASN.1 [ITU02] structure and any extension may only exist once in a given certificate.

As per the RFC, extensions that are marked with the *critical* bit **must** be evaluated by each entity checking the certificate. If a critical extension included in the certificate is not known to the checking entity, the certificate must be rejected as invalid. Non-critical extensions may be discarded if they are unknown.

2.1.3 Certificate Authorities

A certificate issuer or *CA* is the entity that makes sure the subject of a certificate request and the corresponding public key belong together. If any entity wishes to obtain a certificate, they submit a certificate signing request (CSR) to the CA. This request includes the entity's unique identifier – the DN – and a public key. It is signed with the private key belonging to the key included in the request. The requestor must provide some additional identity proof to the certificate authority; for end users this is usually some federally issued photo ID. By comparing the requestor's credentials to the identity information contained in the CSR, the CA certifies that they are who they claim to be. This certification is accomplished by cryptographically signing the requestor's certificate request using the CA's private key. Together with some metadata, this signed request is the certificate.

Each certificate is given an incremental serial number that is unique in the issuing CA's context. This is necessary for maintenance, especially revocation of the certificate.

To verify that a certificate is valid (in the context of the CA that issued it), it can be verified by using the CA's own certificate. This certificate is the so-called *root certificate*. While a CA hierarchy where a CA issues certificates to other CAs is theoretically possible, it is not currently used in the Grid context. There, one CA marks the highest possible level in the trust chain.

In order for any certificate infrastructure to function, the CA must be trusted by all participants of the infrastructure and under no circumstances may its private key be compromised. Certificate authorities that issue certificates for use within the european Grid infrastructure are accredited by the EUGridPMA[1]. There should not be more than one CA per participating country. This limitation rules out the possibility of multiple CAs issuing certificates with identical DNs to different entities under the same authentication domain.

For the purpose of this document, it is assumed that the CA is trusted by all entities within the Grid and that the CA certificate is available for inspection by each entity.

2.1.4 Unauthorized credential access

EECs are usually long-lived, since the administrative process associated with issuing them involves strict checking of credentials, usually not via an automated process. Since private keys for long-lived certificates are obviously to be kept secret, they must never be transferred via an insecure means of communication. It is assumed that all parties adhere to this requirement, so network sniffing is ruled out as an attack vector. However, a number of local attack scenarios can arise from credential theft and forgery.

If an EEC's private key is obtained, the attacker can then proceed to act on the victim's behalf. This will lead to information disclosure and unauthorized resource usage. Since EEC keys are typically stored in the user's home directory on the UI server and protected by the usual operating system measures, an attacker would have to obtain either that user's or administrative privileges on the UI to perform such an attack.

However, since all EEC private keys are encrypted and require a passphrase for decryption, acquiring the private key would not help the attacker – they would have to capture the passphrase too. Setting up a keystroke logging device on the UI for that purpose is not infeasible and cannot be ruled out completely.

Similar to this, an attacker could attempt to obtain the root certificate's key and passphrase by breaking into the certificate storage on the CA. This kind of attack is assumed to not be feasible. The operational requirements in section 4 of [DaG05] suggest this assessment by requiring the CA computer to be protected by cryptographic hardware modules and disconnected from any public networks. Thus, unau-

[1] http://www.eugridpma.org/

thorized duplication of the CA's credentials is highly unlikely.
If the CA certificate is stored on a Grid component in an insecure way, i.e. file permissions are too wide, a malicious user could attempt (and succeed) to overwrite the CA certificate with a forged one. However, this will instantly render all valid EECs useless (they are not signed by the purported CA) and expose the manipulation. Additionally, any difference in the CA certificate would result in it not being trusted on other components of the same Grid, making the manipulation and all manipulated EECs useless. Adding another CA certificate in order to issue a certificate with a duplicate DN (and thus impersonate a legitimate user) is also an option, but not feasible due to the broken chain of trust in that case. The "rogue" CA is not trusted by other Grid components and thus the manipulated certificate is also untrusted. All jobs submited with these credentials will fail.

It can be seen that, given a standards-compliant CA practice, the only weak link in the PKI chain is the EEC. By taking over a UI server, an attacker could obtain all end-entity credentials of that UI's end users. Thus, the attacker could act on the victims' behalves and consume resources that have to be paid for by the victim organization. It is the system administrator's job to prevent intrusions on the UI server; this attack vector cannot be closed within the Grid application. However, if an EEC's private key and corresponding passphrase somehow become known, the corresponding certificate must be invalidated as quickly as possible to limit possible damage. This is done by revoking the certificate.

2.1.5 Certificate Revocation

If for some reason a certificate is compromised and the certificate owner is made aware, they can have the certificate revoked by the issuing entity. This is accomplished with Certificate Revocation Lists (CRLs). These lists contain the serial numbers of certificates which are no longer deemed valid by the CA and cannot be used as a means of authentication any longer. CRLs need to be present on each component of the Grid infrastructure. They are usually fetched in regular intervals via HTTP. Since they need to be trusted by the Grid component, they may only be issued by the CA and need to be signed by it. If a certificate is listed in a CRL, it is to be regarded as untrusted in any given context.

Another solution that facilitates certificate revocation is the Online Certificate Status Protocol (OCSP), which is not currently used in gLite. It is defined in [RFC2560].

2.2 Grid Security Infrastructure

The GSI was originally developed for use in the Globus Toolkit and has been adapted to the gLite middleware. Its motivation is threefold:

- Communication between Grid elements should be authenticated and ideally confidential, so third parties cannot eavesdrop.
- Security shall be provided across organization boundaries, since virtual organizations can be composed of several different physical organizations
- Users must be able to delegate their authorization to the Grid in order to make long-running or parallel computations possible. This is necessary since during the course of a job or a set of jobs, a Grid component might require authorization in order to complete their task (i.e. for data access or to start new sub-jobs).

There is no doubt that transport-layer security and authenticity of information are important – however, both are beyond the scope of this thesis.

X.509 credentials are the main means of authentication in the GSI. For the purpose of delegation and single sign-on, short-lived proxy certificates are derived from the aforementioned end-entity certificates.

2.2.1 Mutual Authentication

To establish an accountable communication link, two peers (named A and B in the following example) need to be sure of the other party's authenticity. Provided that they both have EECs signed by a mutually trusted CA, they can prove their identity to each other with the following protocol:

1. A sends their certificate to B.
2. B checks the certificate, using the appropriate CA certificate.
3. To rule out certificate theft, B sends a random message to A.
4. A encrypts that message with their private key and sends it back to B.
5. B decrypts the encrypted message with the public key obtained in A's certificate and compares it with the original message.
6. The protocol is repeated with B's certificate.

2.2.2 Delegation and Single Sign-On

An essential component for Grid frameworks is a possibility for non-interactive authentication and authorization of Grid users and jobs. This necessity arises, amongst others, from the following use cases:

- A Grid job needs resources from a storage node. These resources are only accessible to an authorized user.
- After submission, Grid jobs pass through a number of components on the Grid and might cross site boundaries during that process. The necessary authentication for this cannot be acquired from the user who submitted the job, as there might no longer be any interaction between the user and the UI.

The concepts of Rights Delegation and Single Sign-On (SSO) are very closely tied together, but serve different purposes. Single Sign-On is primarily used to facilitate the authentication process across multiple components of the Grid, while the delegation of rights is a means to pass along the user's authorization or parts thereof to another party. This thesis is centered around a delegation of rights that is more fine-grained than the current all-or-nothing approach in the gLite middleware, which is an example implementation of the principles outlined in the GSI.
Using X.509 certificates, both single sign-on and rights delegation can be achieved in one step. However, an X.509 certificate alone is neither sufficient for single sign-on nor for rights delegation, because it cannot be used to complete the mutual authentication protocol. To perform that protocol, both the user and the authenticating Grid component need to sign messages using their private key. The EEC's private key must not be transmitted to potentially untrusted third parties since they might be disclosed to unauthorized entities. Revocation and re-issuance of a compromised EEC are not desirable and must be avoided at all costs.

Thus, a short-lived derivative of the long-term EEC should be created - a certificate/key combination that is expendable and does not need to be replaced upon disclosure. These certificates are named proxy certificates and the combination of proxy certificate and private key – along with the EEC and possible intermediate certificates – is considered the *proxy credentials*.

2.2.3 X.509 Proxy Certificates

A concept for creation of short-lived proxy credentials is presented in [RFC3820]. If a user needs to provide authentication credentials for a Grid job, the first choice would be the end-entity credentials. However, sending it along with the Grid job to enable SSO is not possible (see above). Thus, the certificate holder creates a request for a new certificate that has a shorter lifetime than his original EEC and is marked as a proxy in the DN. They then sign this new certificate request with their EEC's private key instead of having it signed by the CA.

The resulting certificate is trusted by other Grid components due to the Chain of Trust incurred. A party that wishes to verify the proxy certificate would use the issuer's public key to do so. This public key is the public key included in the EEC. The EEC can then be checked with the CA's public key which is included in the CA certificate. By successfully validating all parts of the chain, the trust vested in the CA is passed down to the proxy certificate.

With regards to revocation and re-issuance, a proxy certificate is an expendable resource. However, if it is hijacked by an attacker, it can be used to authorize new Grid jobs for the entirety of its lifetime. Moreover, since it is not directly signed by the CA (and the CA has no actual knowledge of its existence), there is no means of revocation for proxy certificates as there is no trusted authority to issue revocation lists. Thus, proxy certificates need to be as short-lived as possible, while being as long-lived as necessary. A lifetime of twelve hours has been widely adopted; after expiry of this lifetime, the proxy must no longer be accepted by Grid components.

2.3 Authorization

After successfully authenticating to the Grid, a user is granted a limited set of capabilities. These capabilities regulate which tasks the authenticated user is allowed to perform on Grid components. Grid components must be able to retrieve a set of capabilities for evaluation, decide if a specific action is authorized and then enforce that decision. Since the component that decides upon authorization is not necessarily the same one that actually enforces it, a distinction is necessary. Authorization components are typically divided into policy decision points (PDP) and policy enforcement points (PEP).

2.3.1 Decentralization

One of the basic principles of the GSI and its successors is decentralization. Since the Grid strives for cooperation between different physical organisations in a VO, many Grid components will be decentralized. Authentication or authorization might not be managed by a central instance, for several reasons:

- Finding a central position within the VO might be difficult – an authorization server is a prime attack target and must be very carefully maintained.
- Centralized structures are prone to becoming a single point of failure in case of network or service outage, and a bottleneck in case of saturation, while not being as easily scalable as a decentralized solution.
- Organizational reasons – a VO participant running centralized authentication and authorization services would require a higher level of trust from the other participants. This is a requirement that is not easily satisfiable.

Decentralized solutions have other weaknesses – the effort required to gather all required information for policy decisions can be higher, and administration of several decentralized vs. a single decision point requires more work.

2.4 VOMS

The Virtual Organization Membership Service or VOMS[2] [Cia05] is a system that manages authorization information within virtual organizations. Essentially, it comprises a database of user roles and capabilities and a means for users to generate Grid credentials that include these roles and capabilities. Based on a user's capabilities, they can then be in- or excluded from usage of resources within that VO. VOMS was developed as a part of the European DataGrid Project.

[2] `http://edg-wp2.web.cern.ch/edg-wp2/security/voms/voms.html`

Chapter 3

Grid job management

3.1 Job workflow in gLite

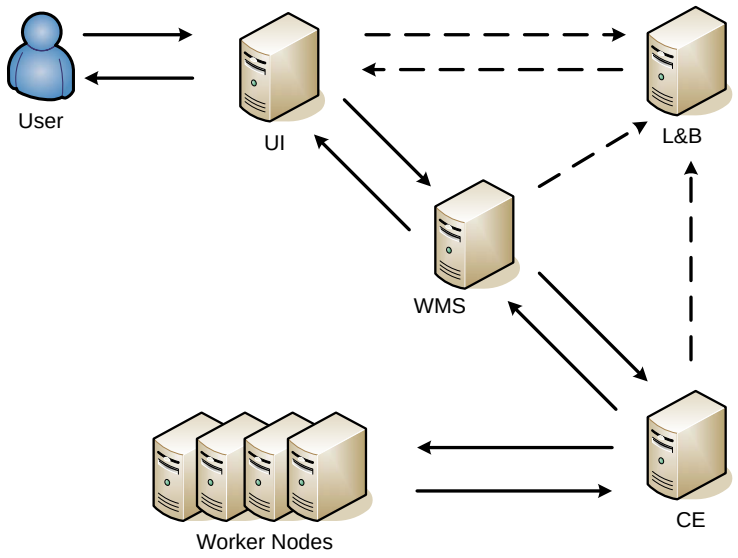

Figure 3.1: Job Workflow in gLite

The gLite middleware uses a multitude of components to process jobs from submission to completion. The basic principle is illustrated in figure 3.1 and can be described as follows:

Prerequisites:

1. The Grid user formulates the job details and requirements in a JDL file.

2. By running `voms-proxy-init`, a proxy certificate with VOMS authorization attributes is generated. Optionally, `myproxy-init` is used to upload a long lived credential to the MyProxy Server (see 3.4).

Job submission and computation

1. The user submits the JDL file, together with their proxy credentials, to the WMS, using `glite-job-submit`.
2. After checking the user's authentication and authorization, the WMS chooses an appropriate CE and forwards the job information and credentials to it.
3. The CE checks the user's authorization (via VOMS attributes) and deploys the job to the associated Worker Nodes.
4. After computation, the WNs returns all job output (contained in the so-called "output sandbox") to the CE.
5. The CE forwards the output sandbox back to the WMS.
6. On user request, the WMS returns the *output sandbox* to the UI (and thus, to the user).

The following sections give a more detailed look at what happens during job submission within specific Grid components.

3.1.1 User Interface

In standard gLite environments, the UI server frontend consists of a set of command-line tools, most notably the `voms-proxy-*`, the `myproxy-*` and the `glite-job-*` families of tools. Jobs generally need to be prepared by the user using the JDL language syntax described in [Pac06]. Job descriptions in JDL format are text files with `attribute = value` pairs that include the command-line instruction to be executed on the WNs as well as various parameters for CE selection and other variables pertaining the job's handling and execution.

Before submitting the job, the user needs to prepare their credentials. When using the MyProxy credential storage, they need to execute `myproxy-init` to create and upload credentials that can be used for credential renewal during the job (see section 3.4). An initial proxy certificate including VOMS authorization extensions is then created with `voms-proxy-init`. This certificate is valid for 12 hours by default and

signed with the EEC's private key. It is saved to a temporary directory, together with its private key, forming the initial *proxy credentials*.

Using the `glite-job-submit` script, the job is then prepared for submission to the WMS. The UI creates a job ID from some static and some random parameters (see section 3.2) and sends this ID to the L&B server. Then, the job – including the job ID and the proxy credentials – is submitted to the WMS. The user is then informed about successful submission and provided the job ID for future reference, especially for status inquiries.

After submission, the user can check their jobs' states with `glite-job-status`. They can manipulate submitted jobs to a limited degree, most notably cancel them (`glite-job-cancel`) and attach to interactive jobs (`glite-job-attach`).

When the job is completed, the user can access its output by issuing the `glite-job-output` command, using the job ID as a parameter to identify one specific job.

Thus, all user maintainable parts of the job lifecycle are managed from the UI server.

3.1.2 Logging and Bookkeeping

Acting as the central instance for persistent information pertaining jobs and their status, the Logging and Bookkeeping service (L&B) composes events that are passed from other components, most notably CE and WMS, to a high-level abstraction, the *job status*. Its operation – similar to logging in conventional computing applications – is transparent to the user and occurs without interaction. However, the UI – and thus also any individual Grid user – is given a public interface to query job information (pull model) or register for notifications (push model).

Upon job submission, every job is assigned a job ID (see below, 3.2) and registered with the L&B. From then on, any event pertaining the job is tracked by L&B; its architecture is **job-centric**. Events come in different classes (such as *Transfer*, *Run*, *Done*) and each class carries specific attributes.

As its name suggests, the L&B service consists of two components that run according to different requirements and specifications. While the logging part, comprised of *producer library*, *locallogger* and *interlogger* services, is designed for fast response and saves event states to disk, the bookkeeping service asynchronously receives events from the interlogger and processes them to the high-level abstraction mentioned before. The authors of [Kre05] assume this architecture to be *suitable*

in the prevailing number of cases while being the most efficient in the erratic Grid environment ([Kre05], p. 6).

As soon as events are transformed into job states, the L&B's public interface (via HTTP or HTTPS) provides the user (or their agent, the UI server) with a possibility to synchronously query (pull) for job information. Additionally, every processed event is matched against a list of registrations for notification by the L&B. If a match is found, a notification for the corresponding status is generated and sent by the L&B to the registered notification listener. Thus, information can also be pushed to any interested parties (e.g. for accounting purposes or for faciliation of automated job monitoring).

The L&B implements the security concept common to all gLite components. Users who wish to query for status information need to authenticate with their respective proxy credentials. In addition to this conventional authentication mechanism, an ACL allows users to share their authorization and grant other users the right to query the L&B for information about their jobs. This can either be done with the subject names of the delegee's certificate or with a VOMS group name.

3.1.3 WMS

The Workload Management System accepts jobs from UI servers and takes all actions necessary for their completion. Like the UI hides most details of job submission from the submitter, the WMS forms an abstraction boundary for the submitting entity and offers limited interaction possibilities to it. The main point of interaction with the WMS is formulation and submission of a Grid job in an appropriate description language (in gLite's case, JDL). It is then the WMS's job to translate this abstract job description into an actual set of resources. The user can affect this conversion to a limited degree by issuing specific resource requirements.
After conversion of the job requirements, the WMS forwards the job to all participating resource providers, especially Computing Elements. It communicates with the L&B to send and receive status information regarding the job and after job completion retrieves the job's results.
After submitting a job, users can interact with the WMS via the UI server and cancel the job, inquire its status or – if the job is already completely processed – retrieve the output sandbox. This is i.a. done with the `glite-job-*` commands mentioned in section 3.1.1.
Internally, the WMS is assembled from a total of seven components, as detailed in

[Pac05]. Those components' interaction is hidden from any external entity, thus a more detailed view is not given in this thesis.
Naturally, authentication and authorization occurs in the same manner as with other Grid components – the submitter's proxy credentials are used for authentication and the user's authorization status is obtained via the `grid-mapfile`, amongst others. In addition to the initial proxy credentials that are submitted together with the job, the WMS has a *Proxy Renewal Service* that detects expiring certificates and – if possible – requests renewed credentials from the MyProxy service (see Section 3.4).

3.1.4 Computing Element

The Computing Element (CE) is the last Grid component that deals with an abstract job description before a Grid job is actually computed. It represents a computing resource to the Grid user and other Grid components. Its optional Web Service interface is employed by end users or a WMS to submit jobs for computation. While direct interaction of job submitters with the CE can be considered a rare use case, submission of jobs via the WMS is the prevailing way of interaction with the CE.

The CE's main function is job management. In current non-SOAP-based gLite environments, the CE receives job information via its gatekeeper service as a HTTP request. The necessary user credentials are also transmitted while a secure connection is established via GSI methods and, together with the job RSL, prepared for resource matching. To accomplish this task, the CE must evaluate a given job JDL file, match the requirements outlined within the job specification to its local resource providers (i.e. only select Worker Nodes with CPU characteristics specified in the JDL file, etc.) and then append the job to its Job Controller. This component of the CE is responsible for submission of the job to the underlying resource management system which in turn is responsible for job execution. However, since

a) users can technically submit jobs without any WMS that authenticates and authorizes them, or

b) A "rogue" or manipulated WMS might submit unauthorized jobs to the CE

the CE authenticates and authorizes each job separately. The authentication decision is – as usual for gLite components – made by verifying the submitted proxy credentials, while the decision if a user is authorized to run a job is made based on several local and remote factors like VO membership, the CE's "opening times" and local

ACL. These factors are defined by the resource providers and need to be enforced on the CE.
To faciliate modularization of these authorization decisions, the CE makes use of a service called LCAS to decide upon them. LCAS (Local Centre Authorization Service) is a modular authorization service that comprises a number of plug-ins to decide if a Grid job can be accepted by the CE or must be denied. These plug-ins currently contain a VOMS plug-in that checks for the right VO membership, a module that checks the proxy certificate's distinguished name and a plug-in to check users against a local ban list. The LCAS is called by the gatekeeper before the job is forwarded any further into the Grid. To faciliate module activation and configuration, a configuration file enumerates the list of LCAS modules used and their arguments, if any. Modules are sequentially called by LCAS and expected to return a boolean authorization decision. If authorization for the job is not granted by any module in the list, the LCAS service as a whole returns a negative decision and the job is not forwarded any further.
In addition to authentication and authorization, the CE is responsible for matching the job submitter's identity to a pool account on the Worker Nodes. This is currently done via a mapping of DNs to user accounts in the grid-mapfile or with another modular service called LCMAPS (Local Credential Mapping Service). This step is performed only after successful authorization by LCAS and thus out of the scope for this thesis.
With the LCAS as an authorization service, the CE acts as a gatekeeper for the Worker Nodes that are attached to it, denying all jobs without sufficient authorization access to the Grid resources.

3.1.5 WN

Worker Nodes are the final element in the gLite execution workflow. Attached to a CE's batch system, they receive jobs for execution. Each job is combined with a WN-local Unix account ID (taken out of a local pool of accounts) to create a sandbox environment for that job's execution. This sandbox account must be cleaned up after job completion and its contents (i.e. the job results) needs to be stored or transferred back to the WMS (via the CE).
The WN or WNs (as selected by the CE) then proceed to compute the job and upon completion, return the results to the CE.

3.2 Job IDs

A central role in job control is taken by the job ID. It is the sole unique identifier assigned to every job (or DAG) that is submitted into the Grid and is essential for status information sent by the UI, WMS and CE.
The job ID is constructed by the UI during the job submission process and reads as follows:
`<scheme>://<authority>/<uniqid>`
or, in an actual example,
`https://lb1.gridlab.uni-hannover.de:9000/DCF8hw_OtxK826QQex76_w`
This string comprises a Uniform Resource Indicator (URI) as specified in [RFC3986]. Its scheme is usually `https://` which corresponds to HTTP over SSL/TLS.
The authority part consists of the hostname and port of the Logging&Bookkeeping server responsible for that UI, as defined in the UI configuration.
The part denoted as `<uniqid>` in the example is a string that is also determined by the UI, but guaranteed to be unique. Internally, the job ID is created by the function `glite_lbu_JobIdRecreate()` that composes it from the given parameters and adds the `<uniqid>` which consists of the IP address, current process ID, a random integer and the current time in seconds since the Epoch. This unique string is then hashed with the MD-5 algorithm and converted into a Base-64 string.
With the combination of a random integer and the Unix timestamp, the probability that two different jobs receive the same job ID is negligible. Thus, the complete string can be regarded as a unique identifier for the job that is about to be submitted.
A user-based policy governing job execution needs very specific targetting. It is of no use to establish a "wild card" system that allows for an unspecific set of job IDs to be included in the policy, since this would defeat the purpose of self-restriction. Since it was established in the last paragraph that the last part of a job ID is unique, wildcards could only be used to restrict execution based on the other parts of the job ID - but neither scheme nor authority serve any meaningful purpose for job identification.
Each of the targets of an execution policy needs to be resolvable to one specific job by the PDP and PEP. The job ID – guaranteed to be unique – is one such valid target identifier for UBPs.

3.3 Job Types

There is a number of different job types as outlined in [Pac06]. Apart from normal jobs, there is a possibility to define dependent or independent collections ob jobs that would – since the number of total jobs is bigger than 1 – be assigned more than one job ID. An execution policy for user self-restriction needs to take that into account.

However it will become clear in the next paragraphs that the different job types can roughly be divided in two groups:

- Singular jobs receive only one job ID.
- Job sets receive one job ID per sub-job and one parent job ID.

Thus, a solution that considers DAGs during policy generation would also cater for partitionable and collection jobs.

3.3.1 Simple jobs

The most straightforward job type defines a one-step job that is submitted to the CE, computed on a set of WNs and then returned. There are no additional sub-jobs that directly depend on this job and have to be catered for; neither is the job an umbrella for a collection of independent jobs.

This job type is the most common and will be the main concern for this thesis.

3.3.2 Direct Acyclic Graph

Using JDL, it is possible to define a set of jobs that are interdependent. These jobs can be represented as a directed graph with a node for each job and an edge between two dependent jobs. Since the execution order of this job graph is significant, the edges have to be directed. A job is only started when all previous jobs it depends on completes successfully ([Pac06], p53). The job description graph needs to be acyclic, since circular dependancies between jobs would result in an infinite loop and thus in a neverending job execution.
With regards to job submission, each sub-job of a DAG and the DAG itself are assigned a job ID. Thus, for the 5-node graph presented in 3.2, 6 job IDs would be issued.

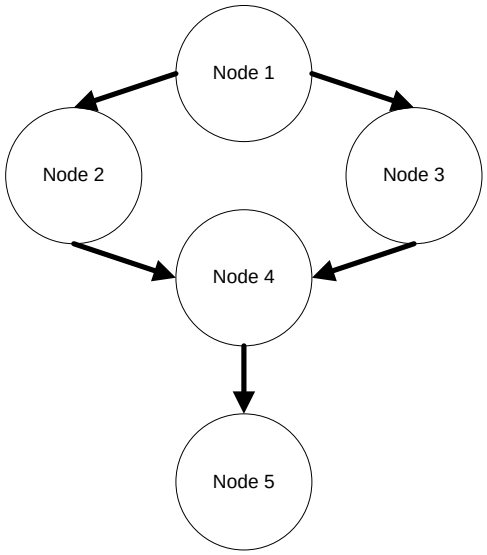

Figure 3.2: A Direct Acyclic Graph

In the example, the job named "Node 1" would be computed first, and after its completion "Node 2" and "Node 3" would be started in parallel. They, in turn, both need to complete successfully for the job called "Node 4" to be started. Upon completion of this job, the final node, "Node 5", is processed.

Thus, a DAG can only complete successfully if all of its node jobs complete successfully. Failure of one node results in failure of the DAG.

3.3.3 Other job types

Collection

A collection is a set of independent jobs that need to be handled as one single request. Similarly to the handling for DAGs, a collection of N sub-jobs is assigned N+1 job IDs upon submission. Regarding UBPs, their behavior can be considered identical to that of a DAG.

Partitionable

A partitionable job is composed of a set of independent sub-jobs that can be computed in parallel. All of these jobs have to be completed successfully for the partitionable job to complete. Thus, a partitionable job is essentially a DAG with one

starting node, one ending node and N parallel intermediate nodes, one for each job partition. Each of these nodes depends on the starting node and the ending node depends on each intermediate node's completion.

Others

There are a number of other job types which are of no concern to the solutions presented in this thesis: *Interactive jobs* where the input and output streams are directly forwarded to the UI, *checkpointable jobs* that can be stopped at specific points and continued later, and *parametric jobs* that take a numeric parameter and increment it, generating multiple jobs that have different input/output parameters. These job types are not in common use and will be disregarded for this thesis. However, it is assumed that interactive and checkpointable jobs will only receive one job ID, while management of parametric jobs is similar to DAG handling.

3.4 Credential renewal with MyProxy

In the Grid, many jobs will run for longer than twelve hours, while the submitter might not want to give it further attention until the result is ready for retrieval. After twelve hours however, a normal proxy certificate expires during computation of the job, forcing the workload management to pause the job until credentials are renewed by the original submitter. Under the paradigm of single sign-on and credential delegation, this is not desirable.
To achieve SSO even for long-running jobs, the submitter can create another certificate with a significantly longer lifetime than the usual 12 hours for proxy certificates and then delegate this certificate to a service that is known to be trustworthy. The MyProxy [MyPr06] is such a service. Maintained by the user's home organization (and thus implicitly trusted), the MyProxy server stores derived certificates with a lifetime of typically one week (7 days). The Grid user creates these certificates by deriving a proxy certificate from their EEC and uploading the resulting credentials to the MyProxy server.
After sending their 7-day proxy credentials to the MyProxy server, the user will proceed to submit a job, using an initial 12-hour proxy certificate they created with i.e. `voms-proxy-init`. In order to utilize the MyProxy facility, the job's JDL file has to be modified to include a MyProxy server name. The workload manager handles the job as usual and passes it on to the CE. Additionally, it stores the certificate expiry

date in the renewal database.
When, during the course of one job, the certificate has reached 75 per cent of its lifetime, the WMS contacts the MyProxy server to inquire a renewed certificate. Using mutual authentication, the WMS is authenticated to the MyProxy server and vice versa. To prove that the WMS is really in possession of the currently valid proxy certificate, it is presented to the MyProxy server. Using an internal ACL, the MyProxy server additionally compares the WMS's FQAN to a list of FQANs that are allowed to retrieve and/or renew certificates.
The WMS creates a new certificate request and sends it to the MyProxy server which signs it with the private key belonging to the 7-day certificate stored in the repository and passes the renewed certificate back to the WMS. The WMS can then send the updated certificate to the Computing Element and to any other parties that require valid certificates for the job (i.e. storage elements).
All current versions of MyProxy contain support for VOMS and the appriopriate attributes are added to the new certificate request before signing using VOMS API functions.

Chapter 4

Initial Considerations

4.1 Motivation

Usage of low-value proxy certificates limits the time window for a hijacker to the maximum lifetime of the certificate but that can still be enough to incur costs or otherwise abuse the certificate. Thus, a more fine-grained solution for the restriction of rights has to be found.

4.1.1 Current situation

Currently, job submission in gLite environments is handled on the UI server, while credential storage is typically outsourced to the MyProxy service. If a user suspects a job to run longer than twelve hours (the lifetime of any normal proxy certificate), they need to pass their credentials to MyProxy (see section 3.4).

In addition to uploading a certificate for renewal purposes, the user creates a proxy certificate that will be used for the first twelve hours of the job's runtime and includes VOMS extensions. This is usually done with the `voms-proxy-init` command. After creating the required proxy certificate, the user submits a job, typically described in a JDL document, using the `glite-job-submit` script. The user's proxy credential are used for authentication. The job is then assigned a unique Job ID (or a set of Job IDs if it is a DAG) and passed to the Workload Management System (WMS). The WMS passes the job on to a Computing Element (CE) which in turn sends it over to the Worker Nodes (WN) that actually compute the job.

Computing Elements use the LCAS to determine if a job may be computed on their associated worker nodes. They use the VO attributes in the certificate passed along

with the job to check if computation is authorized by VO membership or roles. With the default set of authorization plugins, LCAS can also check a list of banned or allowed users based on certificate DNs or allow computation only during specific "opening hours"'. If the proxy certificate is about to expire while a job is still running, it can be renewed using MyProxy and the protocol described in section 3.4.

4.1.2 Attack scenarios

Attacks on a Grid infrastructure can be carried out in a multitude of ways. Attacks that hinder usage of the Grid (Denial of Service or any other destructive attack) are out of the scope of this document. The only type of attack that this thesis deals with is unauthorized resource usage after a successful privilege escalation. Two types of attackers are to be considered:

- *Current members of a VO* that wish to compute jobs, but not be billed for them.
- *External parties* that do not have access to the Grid, but wish to compute jobs.

Current members of a VO that wish to obtain another Grid users' credentials to compute jobs in their name do only have one advantage over external attackers: They already have access to the Grid in order to submit and manage their jobs. Apart from that, both groups have the same privileges. Taking into account the fact that on large UI servers a lot of individual user accounts exist and the notion that some of these accounts have weak passwords, the position of an external attacker becomes even more similar to that of a rogue individual within the organization. It is also likely that both groups have the same knowledge about the Grid's infrastructure since the gLite architecture is described in documents that are publicly accessible. Thus, merging those two groups into one for abstraction reasons shall be permitted.

Attackers cannot set up a wire tap between any two Grid components, since all network traffic is secured by SSL/TLS with strong encryption. Thus, sniffing network traffic in order to obtain valid user credentials is not feasible – since private keys are never transferred over the network, even a working wiretap would not lead to credential disclosure. Thus, in the following scenarios, it must be assumed that an attacker successfully obtained local administrative privileges on a Grid component and has full read access on the component's file system.

Another obvious approach is the MyProxy server, which is however assumed to be safe against attacks.

There are several other components of a gLite Grid that might be susceptible to attacks, but only those that store user certificates are interesting:

1. **The UI** server stores all EECs for its users. Obtaining a large number of CA-signed certificates and the corresponding keys would be the ideal prerequisite to unauthorized job submission. However, the certificates' private keys are encrypted and the encryption passphrases are assumed to be safe from unauthorized access. In addition to the EECs, the UI also stores currently-used proxy certificates in a temporary directory. These unencrypted proxies are only secured by the permissions set on file system level and can be copied by the attacker.

2. **The WMS** carries another set of proxy certificates that are functionally equal to the ones stored on the UI. A successful attack on the WMS would result in the attacker obtaining a number of proxy certificates that can be and have been used to submit jobs.

3. **On a CE**, the proxy certificates are also stored, since they need to be passed to other services and the WNs. Thus, attacking the CE to retrieve certificates is also a possible vector. However, the proxy certificates on the CE are marked as "limited"' and cannot be used to derive new proxies.

4. **Each WN** also carries the user's proxy credentials and is possibly the most susceptible to attacks by malicious third parties. Since jobs cannot be checked for undesired effects, it is generally possible to use a Grid job for an attack against the WN and to gain administrative privileges by executing exploitative software. Additionally, it cannot be ruled out that proxy credentials are leaked to third parties as a result of flawed post-job cleanup routines. However, the derived proxy credentials on the WN are marked as "limited" and cannot be used to submit new grid jobs.

Thus, there are four distinct entry points for attackers who wish to obtain a valid certificate, and it must be assumed that an ambitious attacker would succeed in copying a proxy credential set without the owner's consent. Since the proxy credentials include all necessary authentication information as well as VOMS attributes, the attacker is then able to impersonate the victim and submit jobs to the Grid. They can only do so within the credential's lifetime but on large Grids, twelve hours worth of computing power are still an attractive goal. If the MyProxy service is used by the victim, credentials could be renewed to give the attacker a time window of up to a

week to compute their jobs. On top of that, an attacker could create malicious Grid jobs to gain administrative privileges on the worker nodes. The victim user or users – who can be uniquely identified using their credentials – would then face bills for their alleged resource usage or even criminal charges for their alleged break-in on the worker nodes.

Since a gLite-based Grid is usually made up of a lot of heterogenous components that – in addition to vulnerabilities in the underlying operating systems – must be considered vulnerable to outside or inside attack, the problem of credential disclosure cannot be solved by the GSI. It is likely that components in the Grid is susceptible to compromise, so the only practical remedy is damage control by limiting certificate usage.

It is clear that the certificate's lifetime – although relatively short – is not a sufficient measure to limit certificate abuse. It is necessary to limit certificate usage beyond this factor. It is in the users' foremost interest to limit abuse of their credentials as far as possible. The most straightforward way to execute this limit would be to not pass along credentials at all and have all authentication happen at the UI server. This obviously defeats the purpose of SSO and is thus not feasible. Limiting the certificate usage by shortening its lifetime is also only a minor improvement since – although the attack window would decrease in size – the principal problem of unauthorized usage would not be solved.

Limitation of rights should occur directly in the certificate, as proxy credentials are the key resource for authentication and authorization in the Grid. Using a certificate extension that limits the number of jobs to compute with any given proxy certificate would be a first intuitive approach, but – aside from implementational and conceptual issues – would still only limit the potential abuse to a minimum of one job computed without authorization. With a user-based policy that includes the Job ID of the job to be computed , the abuse potential decreases to zero: even if the attacker obtains the proxy and even if the attacker is able to abort the job the proxy was created for, they still cannot change the Job ID embedded in the certificate. They can also not engineer another job to be given the same ID, since Job IDs are unique and not changeable by the user.

Thus, a Grid user should be able to impose a restriction on their proxy certificate that limits the certificate to usage only with the current job or DAG. This restriction shall be implemented using an open standard and transported in the certificate itself.

4.2 Interoperability

In [EGEE-Sec], the basic principles of interoperability in the EGEE Grid environment are laid out as follows:

- *Be modular* and implement new functionality as a module that can easily be incorporated into existing frameworks
- *Be agnostic* so that new and different ideas and concepts do not interfere with the underlying architecture
- *Be standard*; make use of standardized protocols and languages wherever possible.

The solutions presented in this thesis are based on these paradigms as far as possible. The user-based policies are to be implemented using the open standard XACML, while maintaining the PDP/PEP paradigm. Policy decision and enforcement will be implemented in an LCAS module on the Computing Element, keeping the process of introducing UBPs modular. By not making the usage of user-based policies for job control mandatory for the user and by other means of implementation, the solution will also be agnostic enough to not interfere with existing solutions, especially not with existing solutions that involve UBPs.

4.3 Basic concepts

4.3.1 PDPs and PEPs

Originally introduced in [RFC2904], Policy Decision Points (PDP) and Policy Enforcement Points regulate authorization decisions. The original definition distinguishes between Remote PDPs and Local PDPs (LPDP) – this distinction is omitted for this thesis. It is assumed that a policy decision point can always obtain the required information, be it over a network or by using local resources.

Policy Decision Points in Grids

When a Grid user wishes to perform a privileged action, they are required to have sufficient authorization. The Grid component in question forwards the proposed action to the PDP which decides upon it. The PDP gathers all necessary information

and then makes a decision that is passed back to the requesting entity. In a decentralized structure like the Grid, this process often involves multiple steps and multiple authorization rules that can even contradict each other. There can be global, i.e. Grid-wide permissions that are further restricted by more specific component-local rules, or vice versa. Therefore, rule combining algorithms have to be introduced. There are three distinct rule combining mechanisms in use:

- *Allow-Overrides*: If any rule in the rule set allows the requested action, it is granted.
- *Deny-Overrides*: If the ruleset contains any rule that denies access, access is denied in any case.
- *First-Applicable*: The first rule in the ruleset that is authoritative for the requested action is used for the decision. Rule evaluation stops after that rule.

After gathering and evaluating the supplied ruleset, the PDP makes an authoritative decision (which can be one of `Allow`, `Deny`, `Indeterminate` and `NotApplicable`) and returns it to the PEP. While the first two decisions' meanings are obvious, the latter require further explanation. An indeterminate decision is returned by the PDP if the policy's target could not be determined; if the policy target did not match with the actual target, a decision of `NotApplicable` is returned.

Policy Enforcement Points

The PEP is the requesting party and the actual point of entry for rule decisions. When a user attempts an action that requires privileges, he does so at a PEP. The PEP collects all evidence that is necessary to decide on the request and sends that evidence to the PDP, together with the request. The PDP processes the information presented to it and decides upon the request. Consequently, the decision is returned to the PEP. The PDP's decision is authoritative for the PEP which then has to implement it by either allowing the user to perform said action – or by denying it.

4.3.2 Message Sequences

While in the generalized view presented in [RFC2904], PDP and PEP can be in different administrative realms, for the purpose of this thesis, they are in the same VO and thus share a common, although virtual, home organization. However, there

obviously needs to be a common communication protocol between PEP and PDP for delivery of authorization evidence, user requests and authorization decisions. The authors of [RFC2904] have defined three sequence types for that protocol:

Agent Sequence

In this sequence, the PDP acts as an agent (hence the name) between the user and the PEP. It first receives the request to be authorized, computes a result and then forwards the result to the PEP, together with the request. The PEP acts on the request and replies to the PDP, which sends a return message to the requestor.

Push Sequence

In the Push Sequence, the PEP and PDP do not communicate with each other, but relay all communication through the user. Before submitting a resource request, the user first retrieves an authorization token from the PDP. The user then passes the request, along with the token, to the PEP which fulfills the request. The push model can be employed in environments where the user has access to both the PEP and PDP, but they cannot reliably communicate with each other.

Pull Sequence

In the Pull Sequence, the user's point of communication is at the PEP, which is asked for a resource by the user. The PEP forwards that request, along with the necessary evidence, to the PDP. The PDP decides on the request and returns its decision to the PEP, which in turn evaluates the decision and acts accordingly.

4.4 User self-restriction of execution privileges

As established in 4.1.1, the current situation only allows for a very coarsely grained security model. Using the VOMS-issued attribute certificates, Grid resources can verify a user's VO membership and bestow predefined privileges onto that user. Thus, users who do not belong to the right VO are denied access. Within the VO, all users are given privileges based on their set of roles and capabilities in the VOMS database. The combination of VO, group, role and capability for a given user are called that user's Fully Qualified Attribute Name or FQAN (see [AFVC04] for a

detailed definition). There is, however, no inherent mechanism that allows users to deliberately limit their credentials' usage after they have submitted them into the Grid.
Proxy credentials are created with a limited lifetime of usually 12 hours or less, rendering them useless after this timespan has been exceeded. Thus, abuse is only possible within a narrow time window, which is still more than sufficient for attackers to obtain and abuse proxy credentials.
In the context of this thesis, i.e. regarding job submission, possible actions by the hijacker are as follows:

- Submission of additional computing jobs on behalf of the original users. This would lead to *unauthorized resource consumption* on the Grid and administrative consequences.
- Abnormal termination of jobs that have already been submitted to the Grid, incurring a *denial of service* against the original user.
- Submission of specially manipulated jobs that exploit local software faults on the Worker Nodes and provide the attacker with escalated local privileges on the WNs.
- Submission of crafted jobs that exploit faulty post-job cleanup to gather other information from the WNs (like data remaining in the pool users' home directories).

Since the hijacker poses as a legitimate Grid user, all consequences affect this user – including possible legal or contractual penalties. Thus, it is in the user's best interest to limit credential usage as far as possible while not making their interaction with the Grid more complicated. Although a user should be able to trust all members of their VO, they will still want to limit abuse potential when submitting their credentials from a realm that is in their home domain (the UI) into one that is potentially out of their reach (WMS and following components).

However, since execution of a Grid job can not be reasonably limited on the VO level, any additional restrictions need to be established by the user themselves and enforced by the Grid. A user needs to be able to limit their credentials' scope of usage to the bare minimum of what is necessary to execute a given job.

By specifying this set of rights, the user defines a subset of their proxy credentials' privileges which is significantly smaller than the time- and VO-limited subset that

was previously defined by "conventional" proxy certificates. In the scope of this thesis, there is only a small set of possibilities for user-based restrictions on short-lived credentials. The following options are imaginable:

- Limiting the number of operations that is carried out using the certificate. This limit would only be enforcable if all operations are logged to a central entity or if the certificate itself contains a count of all operations carried out with it or its derivatives. While the former might be the case, the latter is infeasible since the certificate cannot be modified after creation. However, even one additional unauthorized action performed with a certificate is not desirable and undesired side effects (like job resubmission after initial failure) could lead to false positives.
- Limiting the nature of the operations that may be carried out with a given set of proxy credentials to a list of allowed binaries or job types is also not feasible: An attacker could engineer jobs in a way that they match these criteria, while still being unauthorized, e.g. rename an executable or preparing "trojaned" versions of permitted files.
- Limiting the usage to only a given set of operations that are specified by a unique identifier included in the policy.

It has been shown that the first two options can be ruled out. The third option requires a unique identifier for each operation that can be carried out with a set of proxy credentials and such a unique identifier already exists in the form of the job ID. Thus, the job ID is the primary target for the kind of user-based policy that is being introduced in this thesis.

4.5 User-based Policies for job execution

4.5.1 Elements of UBPs

For the PDP to be able to decide upon an authorization request, it first needs said request, consisting of:

- The requestor and proof of their identity.
- The target of the request – i.e. a job ID.

- Optionally, a mode of operation for the request, like `start`, `cancel` or similar.

Such a mode of operation is possible: A user might want to make their jobs not cancellable or similar. However, this functionality is not a critical component for implementation and thus left for consideration in future work.

After receiving the request and all necessary proof, the PDP needs to make an authorization decision based on all active policies. These can stem from a central policy storage (see [LKS03]), decentralized structures and local resources. For user-based policies, the policy content is delivered as an extension to the user's proxy certificate. Each policy contains a set of one or more rules with (at least) the following components:

- The rule object, i.e. execution privileges.
- Since policies are usually issued for a well-defined set of applicants, the rule subject needs to be part of the rule. For a UBP, this is implicitly taken care of with the underlying PKI which identifies the rule subject via its certificate's distinguished name.
- (Optionally, see above) For each distinct mode of operation, an allow/deny directive.

Based on the combination of all applicable rules, the PDP makes a decision that is being forwarded to the PEP.

4.5.2 XACML

As outlined in the previous section, PDPs and PEPs are usually not in the same location and thus need a standardized means of communication for authorization requests and decisions. Since the components involved in this communication are assumed to be from different software vendors, yet need to process each others' input, the contents of such communication needs to be standardized as well. The Extensible Access Control Markup Language (XACML), as proposed by OASIS[1], is an XML-based descriptive language specially developed for modelling of authorization policies.

The decision to use XACML for the encoding of UBPs in this context stems from a number of reasons. Mainly, the gLite paradigm *be standard* advocates the usage of

[1] `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml`

open standards for inter-component communication between Grid components. Additionally, XML-based protocols (like SOAP/WSDL) are already used for much of the network communication inside the gLite middleware and further convergence of gLite and Web Services is intended. On the implementational side, XML documents can easily be validated and there is a sufficient number of libraries for processing of XML documents.

XACML can not only be used for policy description, but also for requests and replies that are communicated between PDP and PEP. Since for this thesis, PDP and PEP will both be on the same Grid element (namely the CE), there is no need for expressively formulated authorization requests; only policies need to be formulated as XACML documents.

4.6 The Grid's interest in enforcement

As mentioned in 4.4, the user has a strong motivation to self-restrict their credentials. However, it might be argued that administrators of Grid components do not necessarily see a reason to invest additional work to accomodate a concern for security that is being voiced by users from a completely different organization.

This argument does not hold true for a number of reasons. The foremost reason is that security mechanisms are usually introduced on a VO-wide basis and not by single members of the VO. Thus, all participants would be bound by formal agreements to modify their components to include PDPs and PEPs for enforcement of UBPs.

Another reason is of a more technical nature and takes into account that malicious users can abuse Grid credentials not only for computation, but also for intrusion on the worker nodes. The administrator responsible for all WNs will want to protect these relatively open elements of the Grid (many pool accounts, many unknown processes and jobs, all unknown to the administrator) with all available measures - including UBPs.

Computing Elements and Worker Nodes are commonly under the same administrative domain and administrators concerned with the WNs' security are also responsible for maintaining the CE. They will most likely implement UBPs on the CE in order to increase security on the WNs.

Accountability is a third concern: It is a known shortcoming of current PKI-based systems that hijacked proxy credentials can be abused. With this in mind, any victim

of such abuse will argue that they cannot be held responsible for the damage since they had no possibility to limit their credentials' usage. Thus, WN owners will not receive reimbursement for the resources consumed by means of certificate abuse.

Thus, implementation of the PDP and PEP mentioned in the next chapter is not only in the user's interest, but also in the interest of CE and WN administrators.

Chapter 5

Implementation

5.1 Requirements

The software implemented for this thesis exclusively deals with execution restrictions in the gLite middleware. In a previous thesis [Gro06], a similar solution for data access policies was developed. The extensible concept introduced in that thesis is a requirement for the current scenario as well as interoperability. The goal of this thesis is the realization of a fully functional prototype that proves the concept and meets the functional requirements set forth in the following sections. However, the prototype is not intended for production use.

5.1.1 Functional requirements

The following *functional requirements* are to be met by the software developed for this thesis:

Policy creation and distribution

- The software developed must be compatible to the currently-installed version 3.0.0 of the gLite middleware.
- Users must be able to restrict their job execution rights by choosing to include a user-based policy in a proxy certificate.
- For normal jobs, only the job ID of that job must be included in the policy.
- For DAGs and other collections of jobs, all job identifiers must be included in the policy.

- The UBP must be distributed to all Grid components that are concerned with job execution.

PDP and PEP

- The existing infrastructure has to be extended so that a UBP is evaluated and honored.
- It must not be possible to execute a job whose ID is not included in the UBP.
- For compatibility reasons, credentials without a valid UBP must be accepted with a *default-accept* paradigm that allows each job to be computed.
- It must be possible to configure the PDP and PEP to make a valid UBP mandatory for job execution.

MyProxy

- The MyProxy credential repository used for proxy renewal must include the XACML execution UBP extension in renewed certificates.
- The MyProxy server should also include the XACML data policy extension in renewed certificates.
- Support for XACML UBPs must be configurable.

5.1.2 Non-functional requirements

The following *non-functional requirements* are to be met by the software developed for this thesis:

Policy creation and distribution

- The implementation should adhere to the basic principles outlined in 4.2.
- The process of policy creation and inclusion should be possible without additional steps for the user.

PDP and PEP

- The implementation should adhere to the basic principles outlined in 4.2.
- The PDP should always return a permissive or restrictive response and never return indeterminate responses.
- The software should be programmed in a way that makes bypassing the PEP as difficult as possible.

MyProxy

- Inclusion of XACML UBPs in renewed proxies should be transparent for the end-user.

5.2 General considerations

5.2.1 Critical vs. noncritical extension

In Section 2.1.2 it was established that X.509 certificate extensions can be marked as "‘critical". If an extension is marked as critical, it *must* be evaluated by any entity that wishes to verify the certificate. If the extension is unknown to the verifying entity, the certificate as a whole cannot be accepted.
For the environment at hand this means the following: If the certificate extension used for a XACML UBP is marked as "‘critical", all grid components that encounter the extended proxy certificate must have knowledge of the extension’s meaning. This is not desirable, since the vast majority of the components that are part of a grid job do have no point of contact with the presented UBP. This is particularly true for any storage or data components, but also for e.g. the L&B.
Since each and every grid component would need to evaluate the policy, every component would need to be modified to check for a valid XACML policy. This greatly increases the implementation effort and is prone to errors and therefore highly undesirable.
Thus, the X.509 certificate extension carrying the execution policy is *not marked as critical*.

5.2.2 Object identifier

X.509 extensions are assigned unique Object Identifiers or OIDs which designate the position of this leaf in the object tree. The institution governing this thesis, Regionales Rechenzentrum für Niedersachsen, has already been assigned a base OID by IANA, `1.3.6.1.4.1.18141`[1]. This base has been extended by a branch for grid usage by appending `.3.100`. Based on this extended base OID, different identifiers for grid-specific extensions are appended. The data policy extension introduced in [Gro06] has been assigned the sub-ID `.1` and the execution policy that is the topic of this thesis was assigned sub-ID `.2`. To faciliate versioning of the extension contents, a version identifier `.1` has also been appended.
Thus, the complete OID for the execution policy extension is `1.3.6.1.4.1.18141.3.100.2.1`.

5.2.3 Push, Pull or Agent sequence?

As established in section 4.3.2, three different sequence models can be used for policy transport and decision. In the current situation, an unambigious classification of the workflow is not possible since the implemented methods are a mixture of several approaches. In the proposed layout, the authorization request, namely the request for the CE to compute a specific job, is forwarded to the PEP along with the authorization evidence, suggesting a push model. However, the push model as outlined in 4.3.2 states that communication between PDP and PEP is always relayed through the user who is presented with an authorization ticket to present to the PEP. This is clearly not the case, since PDP and PEP are so closely coupled that communication with the requestor is not necessary at all.
The closest match to the layout of the policy framework in this thesis is the *pull model* because it best represents the workflow within the CE. The CE's gatekeeper (CEA) component would then be referred to as the PEP, while the LCAS plugin for policy evaluation assumes the PDP role. However, the pull model does not allow for authorization proof – in this case the execution policy – to be pushed to the PDP or PEP, making a classification difficult.

[1] http://www.iana.org/assignments/enterprise-numbers

5.2.4 The XACML policy

As established earlier in this thesis (see section 4.5.2) user-based policy will be formulated in XACML. The request / response syntax of said language will not be used since it is only employed as a policy description language, not for a communication protocol.

The policy in prose

Before formulating a policy in the chosen description language, it is helpful to define the targets and rules in plain english. For the purpose of this thesis, the execution policy is a *whitelist* that only contains one item: the ID of the job that is coupled to the policy. This job ID is the only one for which permission to execute should be granted.
Thus, the policy contents can be summed up in one sentence: *Permit execution of the given job ID and no other.*

Policy Sets

A valid XACML document consists of one or more policy sets. These policy sets are denoted by the `<PolicySet>` tag. The tag contains a parameter to denote the policy combining algorithm. This algorithm specifies how the evaluation of the distinct policies that are contained within the policy set is combined into a single policy. In the current context, only one policy will be evaluated at all – thus, the corresponding argument, `PolicyCombiningAlgId` is set to the value of "'first-applicable". In addition to the rule combining algorithm, a document-wide unique identifier is assigned to the policy set via the attribute `PolicySetId="gLiteUserPolicy"`. The complete opening tag for the policy set reads as follows:

```
<PolicySet PolicyCombiningAlgId="identifier:rule-combining-algorithm
    :first-applicable" PolicySetId="gLiteUserPolicy">
```

In the previous listing, as well as all following code blocks, namespace elements, XML processing information and prefixes are ommitted for the sake of brevity and legibility.

Policies

Each policy set contains n policies, with $0 \leq n$. Each distinct policy describes distinct actions for a resource token and is given a unique identifier via the `PolicyId` attribute. Again, the rule combining algorithm is set to `first-applicable` with the corresponding attribute-value pair:

```
<Policy PolicyId="ExecutionPolicy1" RuleCombiningAlgId="identifier:
    rule-combining-algorithm:first-applicable">
```

Targets

Policies apply a set of *rules* to a *target* to describe permitted or forbidden actions. The rule set and the resource set are constrained in the appropriate tags, `<Target>` and `<Rule>`. The set of resources that forms the policy's target is bracketed in the `<Resources>` tag set with one `<Resource>` element per distinct resource.
In the current environment, the only resource governed by the policy is a job ID, which is a string value. Since the goal is to only allow a job that has the exact same job ID that is denoted in the execution policy, an exact matching is necessary. In XACML, this matching is described by the `<ResourceMatch>` tag that takes an attribute to identify the resource matching algorithm, `MatchId`. In the policy set developed for this thesis, both strings need to be identical, mandating use of the `string-equal` matching identifier. Nested in the `<ResourceMatch>` tag is a tag for the actual value of the attribute that requests should be matched against as well as its type, both in a separate tag. Both are typed with the `DataType` attribute. The complete resource block would typically look like this:

```
1  <Target>
2   <Resources>
3    <Resource>
4     <ResourceMatch MatchId="function:string-equal">
5      <AttributeValue DataType="string">https://lb1.gridlab.uni-
           hannover.de:9000/DCF8hw_OtxK826QQex76_w</AttributeValue>
6      <ResourceAttributeDesignator AttributeId="resource:resource-id"
           DataType="string" />
7     </ResourceMatch>
8    </Resource>
9   </Resources>
10 </Target>
```

The target attribute element contains a job identifier as created by the WMS UI, since it is the target resource. In other works, i.e. in [Gro06], data resources are contained in the resource attributes.

Rules

In addition to the target resources governed by a policy, it contains a number of rules that define the restrictions or permissions imposed by the policy. Each rule is contained within a `<Rule>` tag that is assigned an identifier and the desired effect. Since there is only one rule in the ruleset used for the policy defined in this thesis, the rule ID is set to a generic value of "‘Execution" to denote that the rule is concerned with execution privileges.
Since the policy defines an execution whitelist, the rule effect is set to "‘Permit". If for some reason execution of the job ID given as the target should be denied, the rule effect would have to be set to "‘Deny". An XACML rule also needs a target that is matched by it. The target for any rule is a set of one or more actions that are regulated by the rule. While the data policy defined in [Gro06] would include target actions such as `read` or `write`, the only restrictable action in the current context is `execution`. Thus, the set of actions only contains one element inside the `<Action>` tag. This attribute – contained in an `<AttributeValue>` element – is compared using the function specified in the `<ActionMatch>` element's `MatchId` attribute. As for the target resource, the policy instructs the PDP to perform a string comparison between the contents of the `<AttributeValue>` element and the intended action passed by the PEP. If both values are equal, the effect specified in the opening `<Rule>` tag is applied to the request. The rule section of the XACML policy is included in the listing below.

```
1  <Rule Effect="Permit" RuleId="Execution">
2   <Target>
3    <Actions>
4     <Action>
5      <ActionMatch MatchId="function:string-equal">
6       <AttributeValue DataType="string">execution</AttributeValue>
7       <ActionAttributeDesignator AttributeId="action:action-id"
            DataType="string" />
8      </ActionMatch>
9     </Action>
10   </Actions>
11  </Target>
12 </Rule>
```

However, it should be noted that in the current context, the rules are not actually evaluated by the PDP. XACML is not used in its primary function as a policy language, but as a standards-compliant way to encode a list of restricted resources in plain text. The PDP evaluates the policy only for a list of job identifiers, disregarding the rule portion and applying a "‘default deny" logic to all job IDs that are not contained

in that list. The rule section of the policy is retained for conformity reasons and to faciliate further work on user-based execution policies.

5.3 Overview over necessary modifications

A number of modifications have to be made in order to include an XACML execution policy in users' proxy certificates. These range from the insertion of said policy in a custom certificate extension to implementation of PEP and PDP on the appropriate components. Additionally, the MyProxy server must be modified to automatically copy the XACML UBP from an expiring proxy to the new proxy that is issued to the WMS's Proxy Renewal Service (see Section 3.1.3).
The concept of UBPs mandates at least three necessary components that need to be embedded within the gLite job submission workflow:

- The user-based policy itself (see 4.5 and the previous section, 5.2.4)
- A Policy Enforcement Point (see 4.3.1)
- A Policy Decision Point (see 4.3.1)

To minimize compatibility issues and adhere to the principles outlined in section 4.2, the general workflow is to be kept unmodified; new functionality should be added in a modular fashion.

5.3.1 UBP creation

As the name suggests, a user-based policy is dependant on interaction with the grid user. Since an end user's only interaction with the Grid occurs on the UI server (see Chapter 3 for a closer look at the gLite job submission workflow), UBPs must be constructed and attached to the job there. Moreover, the UBP must be verifiable and must not be changed after creation. Thus, it must be signed with the user's proxy credentials. These requirements already narrow down the amount of possibilities for UBP creation to a very short window between proxy certificate delegation and job submission and determine the UI as the sole alternative for this.

5.3.2 PEP

The user-based policy presented in this thesis governs job execution and is necessary to protect worker nodes from unauthorized workload (and users from unwarranted billing). To achieve this goal, enforcement of this policy must occur as "‘close" (in terms of the job workflow) as possible to the protected entities.
The Policy Enforcement Point for the execution policy could either be placed on the *WMS*, the *CE* or the *WN* itself. However, two of these options can be ruled out.
The WMS is not the correct place to implement the execution policy because in terms of workflow steps, it is too distant from the protected entity. The WMS could belong to a different entity in the VO that is not at all interested in protecting the WNs from additional workload since it is not being held accountable for it. Moreover, jobs can theoretically also be submitted to the CE directly by using its Web Service interface - using a WMS for job submission is not mandatory. Enforcing the policy decision on *the WN* itself is regarded as too late in the workflow chain. As soon as the job has been passed into the CE's batch system, operating system level AuthZ mechanisms come into place instead of Grid-specific ones. From the Grid's point of view, the batch system is a sink/source without its own logic. All decisions regarding the job are made before adding it to the CE's queue, thus making the CE the only valid option to implement a PEP.
Based on these considerations, it was decided that a PEP for execution policy enforcement is best placed on the CE and should be implemented there.

5.3.3 PDP

In general, the Policy Decision Point for the user-based execution policy could be placed on any grid component, since XACML contains a syntax model for requests and responses. However, the requirements and constraints regarding the PEP in the previous section also apply to the PDP. In addition to these, implementing another communication channel to a remote component would induce additional security issues on the network level. Additionally, performance would decrease due to network overhead and possible congestion issues. Due to these reasons, outsourcing the PDP to a completely new component is also not a desirable option. Deploying the PDP on the same component as the PEP is an obvious solution that adheres to the imposed restrictions and avoids implementation of remote communication channels.

5.3.4 Summary

As pointed out in the previous sections, the PDP and PEP will be deployed on the Computing Element, while the user policy itself will be created on the UI. Thus, changes to the job submission workflow will be held as minimal as possible while making all necessary additions in a modular fashion. The updated job workflow is shown in Figure 5.1. Changes are denoted in red, while unmodified parts of the workflow remain in blue.

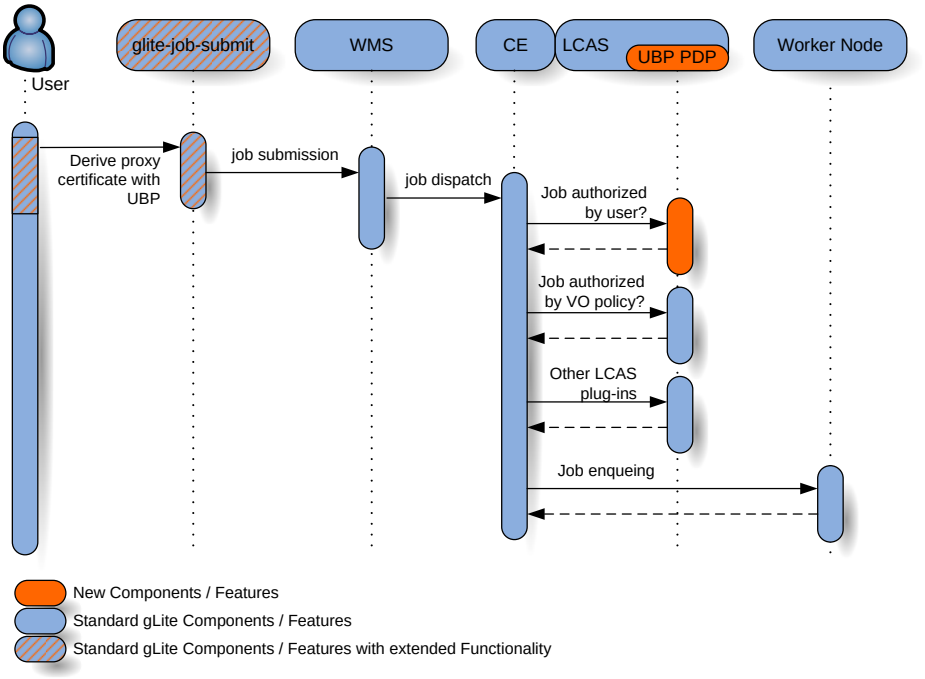

Figure 5.1: Modified job submission workflow

5.4 Modifications on the UI

The concept of UBPs requires an active decision made by the user, thus the starting point for the modifications made for this thesis is the UI server. There, it is necessary to modify the user's proxy certificate upon creation or – if that is not possible – extend the certificate chain by one additional proxy certificate that includes the execution policy.

The author of [Gro06] has decided to modify the utility currently used for proxy

certificate creation, `voms-proxy-init`. As noted in 3.1.1, this tool derives a proxy certificate off the user's EEC and includes the appropriate VOMS attributes. The modified version also adds a data policy to the certificate. The resulting proxy certificate is valid for 12 hours and saved to a temporary directory on the UI, using a fixed naming scheme (typically `/tmp/x509_uXXX`, with the creator's unix UID as an identifier). The user then proceeds to submit a job using `glite-job-submit`, using this precreated certificate.
At first sight, it seems tempting to duplicate this approach and modify `voms-proxy-init` to attach the execution policy to the proxy certificate. However, this is not feasible since at the time this tool is executed, there is no job context present yet. This is intended behavior; a set of proxy credentials created by `voms-proxy-init` should be usable for submission of more than one job. Furthermore, the job ID is not available until shortly before the actual job submission, ruling out the proxy initialization tool for inclusion of a UBP.
The next step in the job submission workflow is usually execution of `glite-job-submit` to actually submit the job into the grid. This python script takes all measures outlined in 3.1.1, especially obtaining a job ID and using the proxy credentials for submission of the job. Thus, it is a better candidate for modification than `voms-proxy-init`. In fact, it is the only tool that is available for modification since submission of the job (and the accompanying credentials) is irreversibly completed after `glite-job-submit` has terminated. Thus, the necessary modifications were made to this script.
In detail, the following additional functionality was necessary:

- Command-line option to include a UBP
- Preparation of an execution policy that includes the job ID(s)
- Inclusion of the execution policy in a proxy certificate
- Submission of the grid job along with the modified certificate

5.4.1 Implementation environment for glite-job-submit

The `glite-job-submit` script is written in the Python scripting language. Since modularity is a key concern, additional functionality should be implemented using existing modules for that language. For the creation and extension of certificates, an interface to the OpenSSL X.509 library[2] was needed. The only existing Python module to offer such an interface, pyOpenSSL[3], must be considered inactive and

[2] http://www.openssl.org/docs/crypto/x509.html
[3] http://pyopenssl.sourceforge.net/

contained a number of shortcomings that makes its use for this thesis infeasible.
One of the advantages that Python offers is the ease of extension: C programs can be used as python libraries with very little effort and their methods can then be used to extend Python's native functionality. By using the well-known C API for OpenSSL, all functions regarding X.509 are available to such a glue module. Several modules that `glite-job-submit` uses for other purposes are already implemented as C modules with a Python compatibility layer. Thus, it was decided to implement the additional function in a similar fashion.

5.4.2 Optionality

To ensure that the user can choose to include or not include a UBP in her proxy certificate – a requirement as per section 5.1 –, the command-line option `--with-execpol` was added to `glite-job-submit`. If this option (without any arguments) is appended to the job submission command, the appropriate execution policy for the job is prepared and included upon submission. No further user interaction is necessary.

5.4.3 Policy creation

For a valid execution policy, an XACML document has to be created on the fly by `glite-job-submit` before job submission, but after creation of one or more job identifiers. Since this policy is quasi-static, it can be created using only Python's native string manipulation functions from a pre-created template. The only part in the XACML file that actually changes is the section containing the Job ID(s). Using a template for the basic XACML document and another one for each `<Resource>` section, the document can be assembled. As established in 5.2.4, the policy's rule section is of no concern to the PEP and does not need to be changed. However, it is retained for future use or reference.

```
1 <Resource>
2   <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
        string-equal">
3     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#
          string">https://lb1.gridlab.uni-hannover.de:9000/
          DCF8hw_OtxK826QQex76_w</AttributeValue>
4     <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:
          xacml:1.0:resource:resource-id" DataType="http://www.w3.org
          /2001/XMLSchema#string" />
5   </ResourceMatch>
6 </Resource>
```

5.4.4 Derived proxy with UBP

During job submission with `glite-job-submit`, a job identifier is created and coupled to the job, which is then submitted. Both of this happens in a relatively short window, making timing essential for policy creation. The policy can only be created after job identifiers are known to the Python script (in the variable `jobid.jobid`), but must be ready before the job is submitted. Thus, the section of the script directly after job ID assignment was modified to call the external module for policy creation and certificate extension.

The external module, compiled as a shared portable library, reads the user's original proxy certificate (created by `voms-proxy-init` or `myproxy-get-delegation`) and creates a new certificate request (CSR). This request's DN is the original proxy's DN, with one additional `CN=proxy` field. The CSR's validity is set to the proxy's validity minus one minute since a certificate's lifetime is limited by the signing certificate's validity and not adhering to that limit could cause compatibility problems. In addition to the CSR, a new set of keys with a key length of 512 bit (the same as for the certificates created by `voms-proxy-init`) is created, using the operating system's random number generator.

In the next step, an ASN.1 structure is created and added to the CSR as a noncritical extension with the OID established in 5.2.2. This structure contains the previously prepared UBP – see the previous section for details on the UBP's creation. In the last step, the external module uses the private key from the original proxy credentials to sign the completed certificate request. The new certificate, its private key and the complete previous certificate chain is written to a file in PEM format. This file's name adheres to the naming convention of `/tmp/x509up_uXXX`, but adds an additional `.restricted` suffix to denote that the proxy certificate in that file contains a user-based policy. This restricted proxy file is deleted by the `glite-job-submit` script after job submission. Compatibility with other tools that delegate proxies is maintained – the modified glite-job-submit script can also parse the certificate chain output from `myproxy-get-delegation`. Since the complete certificate chain is included in addition to the restricted proxy certificate, the chain of trust can easily be inspected by any grid component.

After job submission, the restricted proxy certificate is no longer necessary for administrative purposes. All further interaction with the Grid, such as status or sandbox retrieval, can be performed using the original or any other proxy certificate derived off the user's EEC. To avoid unnecessary information leakage and resource consumption, the restricted proxy is removed after job submission.

5.4.5 Job submission

The process of submitting a job into the Grid is unchanged from the previous version of `glite-job-submit`. However, to ensure that the correct certificate is used for submission, the original proxy (contained in the file `/tmp/x509up_uXXX`) is temporarily renamed and the modified proxy certificate replaces it. This is reverted directly after the job has been submitted. The modified proxy is retained for further reference, especially for interaction with helper tools like `glite-job-status`.
After the job has been submitted, `glite-job-submit` provides the submitter with a message indicating successful submission. This message is slightly modified to include a reference to the UBP and the location of the restricted proxy certificate.

5.5 Modifications on the CE

The inclusion of user-based policy marks the user's wish to restrict the rights they delegate into the Grid. However, this deliberate self-restriction must be checked and enforced before the job is forwarded to the worker nodes for computation. Policy decision and enforcement are to be implemented on the Computing Element – the reasons for this decision are outlined in sections 5.3.2 and 5.3.3 respectively. With the LCAS, the CE already employs a modular service to issue authorization decisions and the paradigm of modularity suggests that the implementation of a new PDP and PEP be done using this service. It was thus decided that the UBP PDP and PEP were to be implemented as an LCAS module.

5.5.1 LCAS module for policy decisions

As mentioned in 3.1.4, the LCAS is called by the gatekeeper service and provided with the necessary information to authorize or deny a job, especially job meta information and the user's credentials. In order to successfully authorize a job based on the UBP discussed in this thesis, it is necessary that LCAS receive the job identifier issued by the WMS-UI. Only if this ID is known, it can be compared to the ID contained in the UBP and access can be granted or denied. It was assumed that the job ID must be known on the CE as soon as job information is forwarded to it by the WMS, since logging and bookkeeping events cannot be generated without it.
However, this assumption proved to be false. The job RSL forwarded to the gatekeeper does not contain any job identifier, especially not the one issued by the UI

(and subsequently included in the UBP). It is only after successful authorization by LCAS that the job ID is retrieved with the CondorG service. This is clearly too late in the processing chain on the CE to be of any use for an LCAS module. In the new, web-service based gLite environment version 3.2, this situation might change, but these changes are not yet implemented.
Thus, the original concept is not realizable using LCAS and an alternative had to be found. After some consideration, it was decided to change the acquisition of authorization proof to accomodate the possibilities available to an LCAS module. This approach provides, although not as secure by concept as the one originally proposed, still a considerable increase in security to Grid users who wish to restrict their credentials' usage.

An alternative approach to job state restriction

As outlined in the previous paragraph, there is currently no way to retrieve the job ID before LCAS authorization, so another foundation for the authorization decision had to be found. Since limiting job execution precisely to the job identifier or identifiers given in the UBP is thus not feasible, the goal of this alternative approach is to limit the abuse potential of hijacked proxy credentials to the minimum possible under the given conditions and to prepare the ground for future implementations.
In the current PKI-based environment with default lifetime values, hijacked proxy credentials offer the attacker a 12-hour time window to submit new jobs and modify running tasks. Limiting this time window to a significantly smaller size can be considered an advantage over current implementations. A new approach takes this and the available information on the CE into account.
With the job identifier included in the UBP, it is possible to query the logging & bookkeeping server to retrieve further information regarding that job. It is especially possible to retrieve proof of the job identifier's existence and to assert that the proxy certificate has not been recycled for submission of new jobs. To clarify the approach, the normal workflow for job submission and the flow of information has to be reviewed.
A user-based policy containing a job identifier is created upon submission of an actual job into the Grid. It cannot be created prophylactically and used later. Thus, the existence of a XACML UBP within a proxy certificate is a definite proof that the job regulated by this policy has been submitted into the Grid.
Upon submission and after each state change, logging information is sent to the L&B server which maintains a state log for each job. State changes are logged with details

including the time of the change and the reason for the change, if available.
It must be assumed that at some point of the job processing chain, an attacker can obtain a set of proxy credentials and use them for job submission. These credentials are derived off the original proxy which includes a user-based policy. An attacker who wishes to submit new jobs using a hijacked set of proxy credentials cannot remove the UBP, so they have to submit the job (which is then assigned a new job identifier) together with the credentials that include the policy. A such illegitimate job will invariably be submitted at a later time than the first, legitimate job since the attacker needs to obtain a copy of the proxy credentials from some grid component. Meanwhile, the original job is still being processed and changes states according to the diagram seen in figure 5.2.

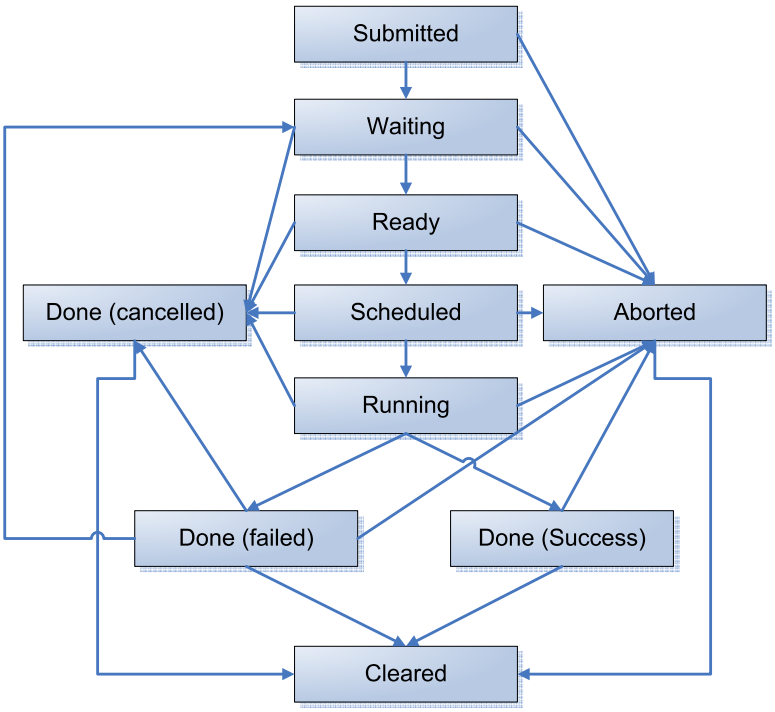

Figure 5.2: gLite job state machine

During the course of processing, the job's state changes from "Submitted" to "Scheduled" and eventually enters the "Running" state when it is computed on the worker nodes. After running, the job enters the state "Done" with a substatus field of either "failed" or "success". Has the job not completed successfully, can it either be resubmitted or cancelled by the user or be aborted by the CE. In all cases, the output sandbox can be retrieved by the user, changing the job's state to "Cleared".
It should be noted that as soon as a job has entered "Running" state, it cannot reenter

that state later without a resubmission. This state change is caused by appropriate user interaction (i.e., a resubmit command from the WMS-UI) and (like all other state changes) logged at the L&B.
As can be seen from the diagram, the "Done" status is ambigious and can have a total of three substates: The "failed" and "success" states denote the result of the actual computation of the job, while the "cancelled" substatus is entered after the user has cancelled the job. As soon as a job has been in either of the states "Done - cancelled" or "Done - success", it has (in either way) been terminated and cannot reenter the Grid.
To determine if a newly submitted job makes use of "recycled" or hijacked proxy credentials, the LCAS module compares the status information of the job ID included in the UBP with an allowable list of job states and attributes. The new job will always arrive at the LCAS with a job state of "Ready". If the L&B query for the job ID from the XACML policy yields a result that shows the job already was in a more advanced state at some prior point in time, it is assumed that something is wrong. The only legitimate reason that a job that already went through a part of or the full job state diagram is a job resubmission. If the job state log does not indicate that the job has been resubmitted after failure, it is certain that the certificate is being abused for submission of additional jobs.
In detail, the LCAS module undertakes the following steps to identify illegitimate submission of additional jobs:

1. Retrieve the job identifier contained in the XACML UBP.
2. Check if the job identifier is known at the L&B server.
3. Check if the current job status is "Ready".
4. Query the L&B server to see if the job ever was in a more advanced state than "Ready".
5. If so, check if the job has previously been resubmitted.

These actions are combined as outlined in the flow chart in figure 5.3. In detail, the reasons for the authorization decision are as follows:

- If the job identifier extracted from the XACML policy is not known at the L&B server that is responsible for it, it is either fabricated or it is so old that all information regarding it has already been purged from the L&B. In either case, the job in question is guaranteed not to be the job that is identified by that

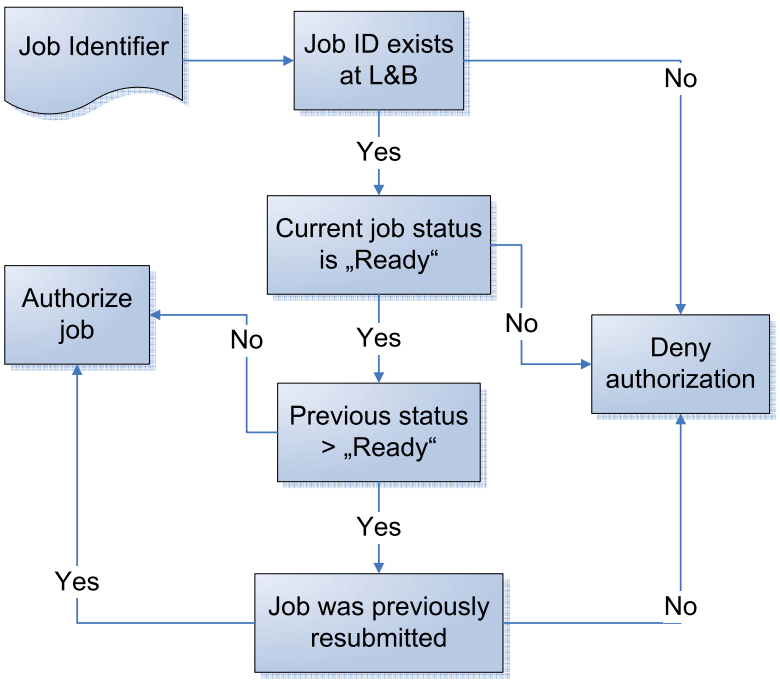

Figure 5.3: Decision flow of LCAS module

job ID. If a very old job identifier is recycled, the proxy certificate's lifetime will in most cases have expired already, ruling out this attack vector since no successful authentication is possible with an expired certificate

- Any job that arrives at the CE does so in the "Ready" state. If the job identified by the UBP extension is not in that state at the time the L&B is queried, both jobs cannot be identical.
- If a job has already passed the "Ready" state and entered "Running" or "Done" states, there is only one legitimate case where it can re-enter the "Ready" state and reappear on the CE: If it failed previously and was resubmitted. Only if such a resubmit event is found on the L&B, the job is authorized.

Implementation details

The LCAS source distribution contains the full source code for an example plugin. This example was used as a basis for the LCAS plugin that acts as a PDP and PEP. Any LCAS plug-in consists of 3 essential functions:

- An initialization function that is called upon startup and sets module-specific variables based on the options specified on the command line.
- The actual authorization function is called after startup and passed all available information. It is expected to return either a LCAS_MOD_FAIL value if the authorization fails or LCAS_MOD_SUCCESS if it succeeds. All actions necessary to reach an authorization decision are performed from inside this function.
- After module execution, a shutdown function is called to clean up memory or file handles and prepare the module for shutdown.

The module initialization function was extended to accomodate an option to make an XACML policy mandatory. This option is a requirement as per 5.1 and the appropriate option is added to the plugin's entry in the `lcas.db` file as the string `-mandatory-policy`. If this option is set, a valid XACML policy *must* be present in the user credentials or authorization is denied.
The main work is done within the function `plugin_confirm_authorization()`. Here, the first step is to acquire the user's credentials via a call to the Globus GSI API. The credentials are then saved to disk – including the current (limited) proxy certificate, the private key that belongs to it and the complete certificate chain. The certificate chain is then iterated to find the XACML UBP extension. If none is found, the module either issues a LCAS_MOD_SUCCESS decision if the policy was not mandatory or denies authorization if the `-mandatory-policy` option was set.
If more than one certificate in the chain contains an XACML policy extension of the same type, a negative authorization decision is issued immediately, since there is no legal use occurence of that behavior.
After one XACML policy has been extracted from the respective certificate extension, the job identifier is queried using the XPath functions of the LibXML library. The XPath expression used to find the job ID is
`//xacml:Policy[@PolicyId=\"ExecutionPolicy1\"]/xacml:Target//xacml:AttributeValue/child::text()`.
The logging and bookkeeping C API is then used to retrieve status information for the job identified by that ID. Since the L&B server requires proper PKI authentication, the previously assembled GSI credentials are presented. After retrieving the job status, another query is sent to the L&B, asking for a list of all states that the appropriate job has passed already. This array is then iterated by a helper function to determine if it contains any illegal state codes. If that is the case, the module issues

a LCAS_MOD_FAIL decision.
The module shutdown function has not been modified from the example implementation. All memory allocations and file handles are freed directly after usage.

PDP and PEP in the LCAS

In this example implementation, the role of the PDP is taken by the LCAS plug-in for policy evaluation, while the caller, the LCAS service, is the PEP. While the module only delivers an authorization decision, its enforcement is left to LCAS itself, effectively allowing or denying the job.
It could also be argued that LCAS as a whole acts as a PDP and the service calling LCAS (in the current environment, the edg-gatekeeper daemon) is the PEP that enforces policy decisions made by LCAS. For the current observations however, the policy evaluation module is considered the PDP and LCAS is considered the PEP.

Handling of job collections

The current approach to user-based restriction unfortunately does not work for any collection of jobs (i.e. any job type that is assigned more than one job identifier) due to conceptual reasons. In general, it is feasible to include each job identifier in a separate rule during policy creation and have the LCAS module issue a job status inquiry for each rule encountered. However, there is no way to make a meaningful authorization decision based on this information because job states can and will differ between different subjobs belonging to the same "super-job".
A simple example shall clarify this issue. A user has submitted a very simple DAG that consists of 5 nodes, as seen in the illustration below:
The jobs B, C and D run in parallel and are only started after job A has completed. Job E is started after all other jobs have been run successfully. As established in 3.3, the DAG shown in the illustration would be assigned 6 job IDs and thus, the policy embedded in the limited proxy certificate would include 6 rules to be evaluated by the LCAS module. Whenever a job arrives at the LCAS and the policy evaluation module is started, it has 6 different job identifiers that are candidates for identification with the job that is currently decided on. Since jobs are enqueued on the CE in a serial way and job A and E are executed out of band from the other jobs, a subset of the jobs might already be running or even done at the time the policy including their identifiers is presented to the LCAS again.

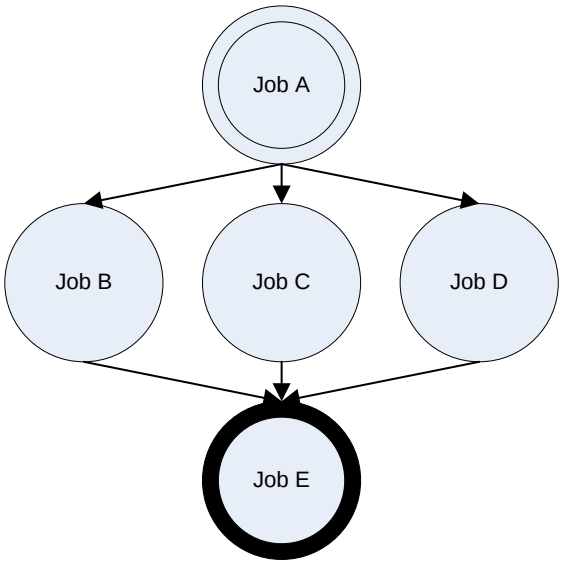

Figure 5.4: A simple DAG job

Thus, there is no way to differentiate a legitimate DAG job that is associated with limited credentials from an illegitimate job that recycles credentials already used for computation of another job. No proof encountered by the LCAS module is not reliable enough to merit an authorization decision.

In newer gLite versions, CREAM-based Computing Elements might be able to provide enough information, i.e., a job identifier, together with the job request.

5.6 Modifications to MyProxy

The MyProxy service can be used for credential renewal if jobs run longer than the initial proxy lifetime. If the WMS detects that a set of credentials are about to run out, it requests a new set at the MyProxy server responsible for the current job. According to its protocol specification outlined in 3.4, the MyProxy server then creates a new set of credentials and passes them back to the WMS. However, all custom extensions including the VOMS attribute extension and also the extensions that contain XACML policies, are not included in the in these credentials.

5.6.1 Why modify the MyProxy server?

On the WMS side, the `glite-proxy-renewd` (gLite Proxy Renewal Daemon) service handles proxy renewal and requests updated credentials. It also handles creation of a VOMS attribute certificate after receiving an updated credential set. It could be considered to have the proxy renewal daemon handle XACML policy inclusion too – however, the decision was made to modify the MyProxy server for this purpose. The reason for this decision is the fact that since the proxy repository contains a lot of sensitive information in form of delegated user credentials, it is often kept "close" to the users within their home organization. Generally, it enjoys a higher level of trust compared to the WMS and therefore is the preferred place for the necessary modifications.

5.6.2 Authentication and Authorization in MyProxy

The MyProxy server listens on a TCP port and authenticates incoming connections using the Globus GSI API. In addition to this transport-level authentication, every incoming request for a new delegation must be separately authorized and authenticated. This is done in two steps:

1. The requestor's certificate DN is compared to a configurable list of allowed credential retrievers. The request is only permitted if a match is found.
2. If provided, the MyProxy passphrase is checked against the passphrase connected to the credential in the repository. Alternatively, a second certificate is accepted as proof of the requestor's authorization to renew a credential.

If a credential is about to be renewed, the certificate chain belonging to that credential is transmitted. The topmost certificate acts as an authentication device, while the expiring proxy is presented as a proof of authorization. This part of the MyProxy protocol can be used for the purpose of this thesis.

5.6.3 Modifications to the source code

In a first step, the MyProxy routines for credential renewal were modified to traverse the certificate chain and check for the existence of an X.509 user policy extension. If found, the contents of that extension is then saved to a temporary file on the MyProxy server for later use.

The functions for proxy creation and signature employed by MyProxy make use of Globus GSI API functions, which posed a problem in the context of this thesis. While the Globus API offers a wealth of utility functions for credential and proxy manipulation, there is currently no way to add a custom extension to a certificate request before signing it. Therefore, the functions `globus_gsi_proxy_sign_req` and `globus_l_gsi_proxy_sign_key` (which is called by the former) were copied from a current (Version 4.0.4) Globus source code repository and included in the MyProxy SSL utility file `ssl_utils.c`. Additionally, several header files were also copied and included in the appropriate C source code files to provide an interface to the Globus GSI API's internal data structures.
Within the MyProxy source code, the function `ssl_proxy_delegation_sign` is responsible for signing a set of newly-created proxy credentials. It was changed to call the modified version of `globus_gsi_proxy_sign_req`.
That function was then modified to check for existence of the temporary file created earlier in the certificate renewal process. If that file exists, a user policy must have been found and should also be included in the new credential set. The modified signing function then adds the policy extension to the prepared CSR and the certificate is signed as usual, creating an updated credential set.

5.6.4 Configuration

As it was a requirement to make all policy-related operations optional, the MyProxy server configuration was extended by a new boolean directive, `renew-xacml-policy yes/no`. Only if that directive is set to "yes" are user policy extensions included in renewed credentials.

5.6.5 Data policies

As a side effect of the implementation work done as a part of this thesis, the data policy extension presented in [Gro06] is also recognized by the modified MyProxy server and included in updated credentials.

5.7 Modifications to the build environment

The modifications to the gLite build environment have been held minimal to minimize the impact on current installations. All software implemented for this thesis

can be transparently deployed on existing gLite 3.0.0 installations. The modified `glite-job-submit` python script and the accompanying policy creation module can be copied to the UI server; the LCAS plug-in can be compiled and deployed on any CE that contains the usual gLite libraries and header files. Thus, no changes had to be made to the current build environment and users can easily integrate the developments presented in this thesis in their current infrastructure.

Chapter 6

Conclusion and Outlook

6.1 Summary

This thesis presented an approach to user self-restriction via XACML policies embedded in proxy certificate extensions. To establish the general concept, it started with a short introtduction to PKI in general and the gLite security architecture in particular.

It then went on to discuss the basic concepts and components of the gLite job execution workflow and possible attack vectors. These attack vectors and possible ways to avoid them were given more thought in the fourth chapter that also established the policy concept and delivered some initial considerations regarding the implementation. It also established the most well-suited Grid components for policy decision and enforcement.

Finally, the implementation and architectural challenges that arose are presented in chapter five. The two parts of the prototypical implementation are presented here – a modification of the gLite job submission script that makes use of an extension to create and embed an XACML policy in a X.509 proxy certificate, and an LCAS module that evaluates this policy and issues an authorization decision to allow or deny a Grid job execution. A third modification concerns the MyProxy server, which is now capable of including user-based policies in renewed proxy credentials.

The software developed for this thesis is contained in source code form on the CD-ROM accompanying it and can be used on any standard gLite 3.0 environment. Comprehensive installation instructions are provided in an appendix to the thesis.

6.2 Improvements to the gLite security situation

The solution presented in this thesis solves one major problem in the current gLite environment: unauthorized duplication of proxy credentials on a grid component and their reutilization is no longer feasible. In standard gLite environments, a set of proxy credentials obtained by a third party can be used for job submission during the rest of its lifetime which usually is about 12 hours. If the solution presented is implemented on the WMS-UI and the CE, no proxy credentials obtained from a WN or other component can be re-used. This stems from the fact that proxy certificates contain an XACML policy with the job ID for the job they were originally submitted with. If these credentials are reused by a malicious third party that obtained them from a compromised Grid component, they will be denied authorization on the CE since their job state cannot match the designed requirements.
Whenever proxy credentials are transmitted to a grid component, they accompany a Grid job. That job's status is set to "running" as soon as the job has passed the CE's authorization layer, thus denoting the fact that the actual computation is the next part in that job's lifecycle. Proxy certificates are not modifiable after issuing and thus the job identifier contained therein cannot be modified either. If a malicious third party has obtained a set of proxy credentials from a grid component, one of the certificates contained in the credential chain contains a policy with the job identifier associated with these credentials. Upon arrival of a job on the CE, the LCAS module developed as a part of this thesis checks that identifier's current processing status as well as its status history. If that history includes a "running" event, the newly arrived job is assumed to be submitted with recycled credentials and thus not authorized.
Taking this behavior into account, the requirements regarding job restriction laid out in 5.1 can be considered as fulfilled. Effectively, there is no longer a way to "steal" credentials from a grid component and reuse them for new job submission. Even if an attacker is able to obtain the restricted credentials for a job before that job is passed on to the CE, he could not put them to any use since there is already a job "on the way" to the CE that includes those credentials. In order to prepare submission of their own job, the attacker might try to manually cancel the legitimate job using `glite-job-cancel` before it reaches the CE, but a such action would result in the job entering the "done (cancelled)" state. A second job using the same limited credentials would not be accepted by the LCAS module on the CE (as mentioned in 5.5.1).
It seems highly unlikely that an illegitimate job that is submitted after a legitimate one, but using the same user credentials, would arrive at the LCAS prior in time

to the former job. However, since the gLite CE offers a Web Service interface to directly submit jobs to it without an intermediate WMS, this attack vector is theroetically possible.
In addition to the enhancements mentioned above, the modifications made to the MyProxy server ensure that an unrestricted set of credentials is never available on the Worker Nodes, futher hindering credential theft.

6.3 Future work and research

The solution presented suffers from a technical problem that is not easily solvable in the current gLite middleware version and should be considered for future development.

6.3.1 Unavailability of job identifiers in the working context

While working with the edg-gatekeeper and LCAS, it became clear that the job identifier is not part of the information passed from the WMS to the Computing Element upon initial submission of a job. Only selected RSL information as well as the job submitter's proxy credentials are transmitted to the CE. It is only after job authorization that the identifier is acquired through the CondorG context. This circumstance posed a central problem in the implementation of the PDP and PEP and there currently seems to be no solution to it without resorting to extreme measures that would change the CE architecture.
In future gLite versions, the current CE layout will be replaced by the CREAM/ICE component infrastructure that brings a closer interconnection between WMS and CE. In the proposed architecture, the ICE (Interface to Cream Environment) component on the WMS will communicate with its counterpart on the CE, the CREAM (Computing Resource Execution And Management). In this context, more information that currently transmitted to the gatekeeper can be provided, possibly including the job identifier necessary for the originally proposed solution to work.

6.3.2 Job collections and DAGs

As pointed out in 5.5.1, handling of job collections, i.e. jobs that are assigned more than one job identifier, is not reliably possible with the current implementation. This is due to the fact that although collections can be represented by an appropriate num-

ber of rules and policies, the LCAS module can make no reliable authorization decision. It is not reliably possible to determine a job's identifier by evaluating the user policy and L&B status information as outlined in 5.5.1.
Thus, the current implementation can only be used for jobs that are assigned one (1) job identifier upon submission.

6.3.3 Persisting attack vectors

The remarks in 6.2 already introduced a possible attack method against the solution presented in this thesis. This attack makes use of the different job submission possibilities on the CE. In addition to the standard gLite workflow of accepting a job by way of a WMS, the CE also offers a Web Service interface to submit jobs directly. In this use case, there is no intermediate WMS that could induce delays in job processing.
An attacker that was somehow able to obtain valid restricted proxy credentials before a job reached the CE could then submit their own jobs directly to the CE and "outrun" the original, legitimate job, which is still being processed on the WMS. The illegitimate job, being the first carrying the XACML UBP, would then be authorized by the LCAS module while the original job, arriving at a later time, would be denied authorization. This is due to the fact that at the time the second job arrives, the illegitimate job would already be in the "running" or even a later job state and the LCAS module's authorization mechanism would detect credential reuse while examining the second job.
The abuse potential for this kind of attack can be considered minor since for successful exploitation, a number of prerequisites need to be satisfied. First, the attacker needs to find a possibility to acquire valid credentials as early as the WMS – a potentially more secured component than the easy to attack Worker Nodes. Then, they need to use these credentials to submit a job within a relatively short time window using a UI server that is accepted by the Computing Element. Only under these conditions could a successful exploit occur.

6.4 Conclusion

This thesis has presented a review of the Grid Security Infrastructure, giving special attention to job execution. It has identified a weakness in the current security infrastructure and discussed measures to close this hole. These measures then resulted in

a working prototype implementation for user-based policies.
Due to infrastructural problems, this implementation could not adhere to the original theoretical approach, but another way to achieve the higher goal of user self-restriction was found and implemented in an LCAS module. Thus, the solution presented in this thesis has contributed to an increased security in the gLite environment.

Appendix A

Installation Instruction

A.1 Installation on the UI

A.1.1 Required libraries

The software developed for this thesis contains a modified `glite-job-submit` script and a C module that is used for policy creation. This module is delivered in source code form and must be compiled prior to use. As it makes use of the OpenSSL X.509 API for certificate generation, this library must be present as well as the Python libraries used for linking against that script language. These libraries can be installed via the apt tool:

```
apt-get install openssl-devel
apt-get install python-devel
```

On a standard Scientific Linux release 3 system, these commands install the libraries and header files in the locations used for the software's compilation and installation scripts. It might be necessary to adapt paths or other parts of the scripts to the target environment.

A.1.2 Installation

Before compilation, all files contained in the archive `glite-job-submit-xacml.tgz` should be unpacked with the command

`tar zxf glite-job-submit-xacml.tgz`.

After unpacking and changing into the directory `glite-job-submit-xacml-policy`, the C module can be compiled using the included shell script:

```
sh compile.sh
```

The compiler will create a shared object named `createproxy.so` that is called from within `glite-job-submit`. A modified job submission script (based on the gLite 3.0.0 CVS version) is included in the archive. The file `tpl.xml` that is also included in the archive is required for operation - it contains an XACML policy template. All three files, `glite-job-submit, createproxy.so` and `tpl.xml` must reside in the same directory during operation.

A.1.3 Usage

The additional functionality contained in the script can be invoked by appending the `--with-execpol` option to the script command line:
`./glite-job-submit --with-execpol [further job options]`
If this option is included, the job submission script prepares restricted credentials. It requires valid user credentials in the usual path (`/tmp/X509up_uXXX`). These proxy credentials can be from a `voms-proxy-init` or `myproxy-get-delegation` call.

A.2 Installation on the CE

A.2.1 Requirements

It is assumed that before attempting to install the software provided as a part of this thesis, all elements of a standard gLite Computing Element are operational, especially the edg-gatekeeper and LCAS components.
Apart from the standard libraries that are already installed as a part of the gLite CE, the LCAS module for policy evaluation requires the following libraries:

- OpenSSL 0.9.7 or higher
- LibXML

It must be made sure that the L&B API libraries are also installed (any CE installed via CERN's apt sources fulfills this requirement) and that a valid `globus_config.h` file is found. Otherwise, compilation might fail.
The required LibXML can be installed from apt with the following command:
`apt-get install libxml2-devel`
The OpenSSL library is installed from apt by issuing this command:
`apt-get install openssl-devel`

A.2.2 Development Environment

The PDP developed for policy evaluation is contained within an LCAS module. This module can be compiled on any CE that has a current (CVS tag `GLITE_3_0_0`) source checkout of the LCAS. Since it makes use of LCAS's module API, the policy evaluation module requires a full LCAS source tree which can be checked out from the anonymous gLite CVS repository. This is achieved with the following steps:

1. Set the CVSROOT environment variable:
   `export CVSROOT=:pserver:anonymous@glite.cvs.cern.ch:/cvs/glite`
2. Check out the base gLite 3.0.0 sources:
   `cvs co -r GLITE_RELEASE_3_0_0 org.glite`
3. Check out the gLite security source tree:
   `cvs co -r GLITE_RELEASE_3_0_0 org.glite.security`
4. Check out the LCAS:
   `cvs co -r GLITE_RELEASE_3_0_0 org.glite.security.lcas`

After checking out all necessary sources, the build environment must be prepared with the `ant` tool:
`ant -f org.glite.security.lcas/project/glite.security.csf.xml`
Building the environment can take from several minutes to several hours.
After the build environment has been prepared, the LCAS source distribution needs to be prepared for compilation by copying M4 macros into the project directory and calling a bootstrap script (instructions as per `org.glite.security.lcas/INSTALL`):

```
cd org.glite.security.lcas
cp ../org.glite/project/*m4 project
./bootstrap
```

A.2.3 Installation

Now, the tar.gz file that includes the LCAS module can be copied into the org.glite.security.lcas directory and unpacked there. Its files will be unpacked into the `examples` subdirectory and overwrite the existing example module.
`tar zxf lcas-module-xacml.tar.gz`
After unpacking the module, it can be compiled by changing into the directory and calling the "make" command:

```
cd examples
make
```

In a standard gLite environment with the required libraries (see the paragraph above) installed, the compilation will result in the LCAS module being built as a dynamic library into the `.libs` subdirectory. After successful compilation, the library and a number of symbolic links to it need to be copied into the LCAS module directory:

```
cp .libs/*.so /opt/glite/lib
```

After copying the library and its symbolic links, the installation is completed and the LCAS module can be configured.

A.2.4 Configuration

After the plugin has been copied to the gLite module directory, it must be enabled in the LCAS configuration. The main configuration file that regulates plugin usage is `/opt/glite/etc/lcas/lcas.db`. By adding a line to this file, the policy evaluation module is activated:

```
pluginname=lcas_plugin_example.mod,pluginargs="mandatory-policy"
```

In the example above, the plugin is configured to make an XACML policy inside the proxy certificate mandatory. If it should be optional, the appropriate line would read like:

```
pluginname=lcas_plugin_example.mod,pluginargs=""
```

After enabling the plugin in the LCAS configuration, the gLite CE needs to be restarted:

```
/etc/init.d/gLite restart
```

After restarting, the LCAS module is activated and will be called by the LCAS as soon as a job needs to be authorized.

A.3 Installation of MyProxy

The CD-ROM contains a modified version of MyProxy 3.2. While the MyProxy version distribued with gLite 3.0 is considerably older, this version was determined to work in the gLite testbed environment at RRZN and is recent enough to contain a number of bugfixes. The current version 3.9 of MyProxy cannot be compiled in a gLite 3.0 build environment because it links to a version of OpenSSL that is newer than the one contained in gLite 3.0.

A.3.1 Prerequisites

For the build process to complete successfully, the gLite and Globus libraries and header files for a standard gLite 3.0 environment must be installed. The build flavor used for installation, e.g., `gcc32dbg` or similar should also be known.

A.3.2 Installation

The source code for MyProxy is contained in a gzip-compressed tar archive and does not need to be uncompressed. It should be copied to a convenient location on the target system. In that location, the build and installation process is started with the following command:
`/opt/gpt/sbin/gpt-build -force -verbose myproxy-3.2.tar.gz gcc32dbg`
If a different flavor, e.g., `gcc32` or a pthr flavor is used, the last argument to the command line must be modified accordingly.
After the build was successful, the MyProxy server needs to be started (if there was no MyProxy server previously installed on the machine) or restarted (if MyProxy was previously installed) with
`/etc/init.d/myproxy start` and `/etc/init.d/myproxy restart`, respectively.

A.3.3 Configuration

User policy support in the modified MyProxy server is configurable via the main configuration file `/etc/myproxy-server.config`. To enable policy renewal, the following line must be added:
`renew-xacml-policy yes`
After changing the configuration file, the MyProxy server must be restarted.

Appendix B

Sample documents

B.1 XACML execution policy

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
       xmlns:xacml="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       PolicyCombiningAlgId="identifier:rule-combining-algorithm:first-
       applicable" PolicySetId="gLiteUserPolicy" xsi:schemaLocation="
       urn:oasis:names:tc:xacml:2.0:policy:schema:os http://docs.oasis-
       open.org/xacml/2.0/XACML-CORE/schema_files/access_control-xacml
       -2.0-policy-schema-os.xsd">
3    <Target />
4    <Policy PolicyId="ExecutionPolicy1" RuleCombiningAlgId="identifier
         :rule-combining-algorithm:first-applicable">
5      <Target>
6          <Resources>
7                  <Resource>
8            <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:
                 function:string-equal">
9              <AttributeValue DataType="http://www.w3.org/2001/
                   XMLSchema#string">https://lb1.gridlab.uni-hannover.
                   de:9000/DCF8hw_OtxK826QQex76_w</AttributeValue>
10             <ResourceAttributeDesignator AttributeId="urn:oasis:
                   names:tc:xacml:1.0:resource:resource-id" DataType="
                   http://www.w3.org/2001/XMLSchema#string" />
11           </ResourceMatch>
12         </Resource>
13         </Resources>
14       </Target>
15     <Rule Effect="Permit" RuleId="Execution">
16       <Target>
17         <Actions>
18           <Action>
19             <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:
                   function:string-equal">
20               <AttributeValue DataType="http://www.w3.org/2001/
                     XMLSchema#string">execution</AttributeValue>
```

```
21                  <ActionAttributeDesignator AttributeId="urn:oasis:
                        names:tc:xacml:1.0:action:action-id" DataType="
                        http://www.w3.org/2001/XMLSchema#string" />
22                </ActionMatch>
23              </Action>
24            </Actions>
25          </Target>
26        </Rule>
27      </Policy>
28    </PolicySet>
```

Appendix C

Supplementals

C.1 Contents of the CD-ROM

The CD-ROM supplied as a part of this thesis contains the following items:

1. This thesis in PDF and TeX format
2. The source code for a modified version of glite-job-submit in a tar.gz archive
3. The source code for an LCAS module for execution policy decision and enforcement in a tar.gz archive
4. Source code for the modifications made to the MyProxy proxy renewal service on the WMS

Bibliography

[AFVC04] Á. FROHNER, V. CIASCHINI *VOMS Credential Format* `http://edg-wp2.web.cern.ch/edg-wp2/security/voms/edg-voms-credential.pdf`

[Cia05] V. CIASCHINI ET AL. *VOMS User's Guide* EGEE document identifier EGEE-JRA1-TEC-571991 `https://edms.cern.ch/file/571991/1/voms-guide.pdf` (2005).

[DaG05] DAVID GROEP. *Profile for Traditional X.509 Public Key Certification Authorities with secured Infrastructure.* `http://www.eugridpma.org/guidelines/IGTF-AP-classic-20050930-4-0.pdf`

[DiM05] A. DI MEGLIO. *EGEE Developer's Guide For The gLite EGEE Middleware.* `https://edms.cern.ch/document/468700` (2005).

[EGEE-Sec] EGEE JRA 3 *Global Security Architecture for Web and Legacy services* EU Deliverable DJRA3.1 (2004). `https://edms.cern.ch/file/487004/1.1/EGEE-JRA3-TEC-487004-DJRA3.1-1.1.pdf`

[EUGridPMA05] EUGRIDPMA *Minimum CA requirements.* EuGridPMA (2005). `http://www.eugridpma.org/guidelines/EUGridPMA-minreq-classic-20050128-3-2.pdf`

[Fos02] IAN FOSTER *What is the Grid? A Three Point Checklist* `http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf` (2002).

[FoKe04] I. FOSTER, C. KESSELMAN. *The Grid: Blueprint for a New Computing Infrastructure, Second Edition.* Morgan Kaufman Publishers (2004). ISBN: 1-558-60933-4

[GLI06] EGEE ENABLING GRIDS FOR E-SCIENCE IN EUROPE. *EGEE > gLite > Quality Assurance: SLOC*. http://glite.web.cern.ch/glite/project/sloc.aspx (29.09.2006)

[Gro06] RALF GROEPER *Policy-based Authorization for Grid Data-Management* (2006).

[ITU02] ITU-T. *Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation*. ITU-T Recommendation X.680 (2002). http://www.itu.int/rec/T-REC-X.680-200207-I/en

[ITU05] ITU-T. *Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks*. ITU-T Recommendation X.509 (2005). http://www.itu.int/rec/T-REC-X.509-200508-I/en

[Jon05] BOB JONES *CERN and its approach to Grids* https://edms.cern.ch/file/654356/1/BobJones_UNESCO_HP_Sept2005.ppt (2005).

[Kre05] ALES KRENEK ET AL. *Logging & Bookkeeping User's Guide* EGEE document identifier EGEE-JRA1-TEC-571273 https://edms.cern.ch/document/571273/1 (2005).

[LKS03] M. LORCH, D. KAFURA, S. SHAH *An XACML-based Policy Management and Authorization Service for Globus Resources* Proceedings of the Foruth International Workshop on Grid Computing (2003).

[MyPr06] BOARD OF TRUSTEES OF THE UNIVERSITY OF ILLINOIS. *MyProxy Credential Management Service*. Board of Trustees of the University of Illinois (09/05/06). http://grid.ncsa.uiuc.edu/myproxy/

[OAS06] ORGANIZATION FOR THE ADVANCEMENT OF STRUCTURED INFORMATION STANDARDS (OASIS). *Language (XACML) Version 2.0 Policy Schema*. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-policy-schema-os.xsd

[Pac05] F. PACINI *WMS Service User's Guide* EGEE document identifier EGEE-JRA1-TEC-572489 https://edms.cern.ch/document/572489/1 (2005).

[Pac06] F. PACINI *Job Description Language Attributes Specification* EGEE document identifier EGEE-JRA1-TEC-555796-JDL-Attributes-v0-8 `https://edms.cern.ch/file/555796/1/EGEE-JRA1-TEC-555796-JDL-Attributes-v0-8.pdf` (2006).

[PWFK02+] L. PEARLMAN, V. WELCH, I. FOSTER, C. KESSELMAN, S. TUECKE. *A Community Authorization Service for Group Collaboration.* `http://www.globus.org/alliance/publications/papers/CAS_2002_Revised.pdf`

[RFC2459] R. HOUSLEY, W. FORD, W. POLK, D. SOLO *Internet X.509 Public Key Infrastructure Certificate and CRL Profile.* Request for Comment 2459, Network Working Group (1999). `http://www.ietf.org/rfc/rfc2459.txt`

[RFC2560] M. MYERS, R. ANKNEY, A. MALPANI, S. GALPERIN, C. ADAMS. *X.509 Internet Public Key Infrastructure: Online Certificate Status Protocol - OCSP.* Request for Comments 2560, Network Working Group (1999). `http://www.ietf.org/rfc/rfc2560.txt`

[RFC2748] D. DURHAM, J. BOYLE, R. COHEN, S. HERZOG, R. RAJAN, A. SASTRY *The COPS (Common Open Policy Service) Protocol* IETF Network Working Group Request for Comment 2748 (2000). `http://tools.ietf.org/html/rfc2748`

[RFC2904] J. VOLLBRECHT, P. CALHOUN, S. FARRELL, L. GOMMANS, G. GROSS, B. DE BRUIJN, C. DE LAAT, M. HOLDREGE, D. SPENCE. *AAA Authorization Framework* Request for Comments 2904, Network Working Group (2000). `http://www.ietf.org/rfc/rfc2904.txt`

[RFC2906] S. FARRELL, J. VOLLBRECHT, P. CALHOUN, L. GOMMANS, G. GROSS, B. DE BRUIJN, C. DE LAAT, M. HOLDREGE, D. SPENCE. *AAA Authorization Requirements.* Request for Comments 2906, Network Working Group (2000). `http://www.ietf.org/rfc/rfc2096.txt`

[RFC3820] S.TUECKE, V. WELCH, D. ENGERT, L. PEARLMAN, M. THOMPSON. *Internet X.509 Public Key Infrastructure (PKI): Proxy Certifi-*

cate Profile. Request for Comments 3820, Network Working Group (2004). http://www.ietf.org/rfc/rfc3820.txt

[RFC3986] T. BERNERS-LEE, R. FIELDING, L. MASINTER *Uniform Resource Identifier (URI): Generic Syntax* IETF Network Working Group, Request for Comment 3986 (2005). http://www.ietf.org/rfc/rfc3986.txt

[VA07] VARIOUS AUTHORS *gLite Security Best Practice* http://rss-grid-security.cern.ch/glite.php (2007).

[Wel05] V. WELCH. *Globus Toolkit Version 4 Grid Security Infrastructure: A Standards Perspective*. http://www.globus.org/toolkit/docs/4.0/security/GT4-GSI-Overview.pdf (2005).