DataGrid

SECURITY DESIGN

Document identifier:	**DataGrid-07-D7.6-0112-0-1**
EDMS id:	**344562**
Date:	**15/05/2002**
Work package:	**WP07: Security**
Partner(s):	**CERN**
Lead Partner:	**CERN**
Document status:	**DRAFT**
Deliverable identifier:	**D7.6**

Abstract: Consistent model of the DataGrid authentication and authorization schemes.

Delivery Slip

	Name	Partner	Date	Signature
From				
Reviewed by	Moderator and reviewers			
Approved by	PTB			

Document Log

Issue	Date	Comment	Author
0-1	2002-04-15	First draft	Ákos FROHNER
0-2	2002-05-16	Doc id., use cases	Ákos FROHNER

Document Change Record

Issue	Item	Reason for Change

Files

Software Products	User files / URL
Word	DataGrid-07-D7.6-0112-0-2-SecurityDesign.doc

CONTENT

1. INTRODUCTION

1.1. OBJECTIVES OF THIS DOCUMENT

Based on the D7.5 Security Requirements document we intend to describe a consistent model for authentication and authorization in the European DataGrid tools and services. The model should be applicable to every work package service. It is also desired that our solutions will be compatible with other grid projects.

1.2. APPLICATION AREA

1.3. APPLICABLE DOCUMENTS AND REFERENCE DOCUMENTS

Applicable documents

[A1]	Security Requirements And Testbed 1 Security Implementation

DataGrid document can be found by going to the DataGrid web page at
(**http://eu-datagrid.web.cern.ch**) and selecting the relevant work package.

Reference documents

[R1]

1.4. DOCUMENT AMENDMENT PROCEDURE

1.5. TERMINOLOGY

Definitions

Glossary

2. EXECUTIVE SUMMARY

If necessary, this is one or two pages executive summary. It contains an adequate description of the conclusions or results.

3. OVERVIEW

A beginning is the time for taking the most delicate care that the balances are correct.

In the Grid projects the biggest challenge is to face the scalability problems. We have to deal with large virtual organizations spanning over several countries and thousands of machines, while trying to make the whole system quick, flexible, responsive and robust.

To achieve responsiveness and improve robustness, we should make local decisions, based on information available locally. To achieve flexibility and optimize the large-scale performance we should make global decisions, based on information available in the distributed system. To find the balance between these approaches we have to consider the information flow: when and where is a piece of information updated and when and where should it be published.

This document we will focus on security, but issues discussed here are in close connection with every piece of the DataGrid project. It affects the internal logic of several services and even the architecture and interaction of some other pieces.

The basis of the Grid security infrastructure is the Public Key Infrastructure (PKI). Once a web of trust is set up (trusted certificate authorities, etc.), it can be used to prove the validity and integrity a piece of information without contacting the issuer of this data. We will use it to transfer authentication and authorization information.

Once this information is sent to a site it can be validated “offline” reducing the dependency on the source of the information – it will improve the speed and robustness, because we will not have to contact it every time. By setting constraints on the validity we can trigger the expiration of authentication and authorization information. This will help to keep the overall system up-to-date.

We have to set these validity constraints (e.g. expiration time) carefully, thus it would satisfy the security requirements, but not generate too much traffic in the overall system.

To understand the overall structure we have to see the process in details:

- Authenticate a user at a service
- Gather additional information associated to the user or the actual session (e.g. group membership, role, time – see [A1] for details
- Gather additional information associated to the protected service or object (e.g. file permissions)
- Get local policy applicable to the situation (e.g. temporarily disabled user)
- Make an authorization information based on the identity and the additional information

To give a scalable solution we have to express the information flow: when and where is a piece of information updated (also created and deleted) and when and where is it used. *Chapter 4* describes the authentication process, *Chapter 5* the gathering of user related information and *Chapter 0* the rest of the information sources. *Chapter 7* discusses the lifecycle of the principals in the DataGrid project giving information about the update schedules of the information associated with them.

In this document we don’t intend to discuss *accounting*, *auditing* and *non-repudiation*, because these issues are closely related to the logging and monitoring of services. We do not intend to discuss *confidentiality* in detail, because it can be achieved by careful setting of permissions, if the information is available at the decision making point (e.g. from where does the user log in). We also do not intend to discuss *integrity* in detail, because this should be solved at service/application level.

4. AUTHENTICATION

4.1. OVERVIEW OF GRID SECURITY INFRASTRUCTURE

The Grid Security Infrastructure (GSI) is Globus's implementation of the GSSAPI. GSSAPI is defined in the Internet RFC 2743 [R2]. It is based on asymmetric cryptography used in a "Public Key Infrastructure" (PKI). Asymmetric cryptography allows users to communicate securely without the need for a prior confidential channel to exchange the encryption key. Exploiting features of a specific class of mathematical challenges that are easy to create but virtually impossible to solve (like factorising large prime numbers), end-entities generate a complementary set of keys: a "private key" that will be kept secret and a "public key", that is broadcast to the world. Data encrypted with the public key can only be deciphered with the private key (and vice versa). Thus data confidentiality, message integrity and non-repudiation can be achieved with the two keys of the key pair.

A PKI is used to uniquely bind an identifier to a specific public key together in a "Certificate". The identifier can represent any entity: a human being, a host on the Internet or a Grid service. Anyone wanting to communicate to another entity on the Grid can obtain their certificate and use the public key contained in it to send messages that can only be read by the original owner – who has knowledge of the private key needed to decipher the message. But the sender must first be sure that the intended recipient is indeed the holder of this private key. Therefore a trusted third party digitally signs the certificate: the Certification Authority (CA). The CA certifies with its signature that the identifier contained in the certificate is a truthful representation of the identity that possesses the associated private key. The CA's digital signature is again based on public key cryptography.

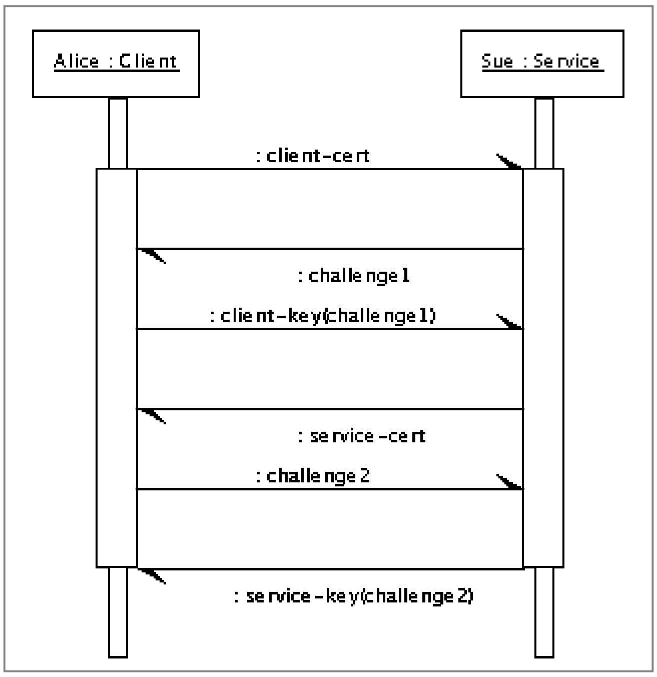
The Grid Security Infrastructure (GSI) is based on public key cryptography. Its primary motivations are (quoted from http://www.globus.org/security/):

- the need for secure communication (authenticated and perhaps confidential) between elements of a computational Grid;
- the need to support security across organizational boundaries, thus prohibiting a centrally-managed security system;
- the need to support "single sign-on" for users of the Grid, including delegation of credentials for computations that involve multiple resources and/or sites.

Every user and service is identified by a certificate encoded in the X.509 format, which contains:

- a *subject name*, which identifies the user;
- the subject's public key;
- the identity of a Certificate Authority (CA) that has signed the certificate to certify that the public key and the identity both belong to the subject;
- the digital signature of the Certification Authority (CA) certifying that the public key belongs to the user.

4.2. MUTUAL AUTHENTICATION



Two parties may authenticate one another using mutual authentication, if they trust the CAs that signed each other's certificate. If they do trust the other's CA, they will then have copies of the other CAs public keys.

To mutually authenticate, the first person (Alice, client) establishes a connection to the second person (Sue, service). To start the authentication process, Alice gives Sue her certificate. The certificate tells Sue who Alive is claiming to be (the identity), what Alice's public key is, and what CA is being used to certify the certificate. Sue will first make sure that the certificate is valid by checking the CA's digital signature to make sure that the CA actually signed the certificate and that the certificate hasn't been tampered with. (This is where Sue must trust the CA that signed Alice's certificate.)

Once Sue has checked out Alice's certificate, Sue must make sure that Alice is really the person identified in the certificate. Sue generates a random message (challenge1) and sends it to Alice, asking Alice to encrypt it. Alice encrypts the message using her private key, and sends it back to Sue. Sue decrypts the message using Alice's public key. If this results in the original random message, then Sue knows that Alice is who she says to be (at least she is in possession of the private key to prove it).

Now that Sue trusts Alice's identity, the same operation must happen in reverse. Sue sends to Alice her certificate; Alice validates the certificate and sends a challenge (2) message to be encrypted. Sue encrypts the message and sends it back to Alice, and Alice decrypts it and compares it with the original. If it matches, then Alice knows that Sue is who she says to be.

At this point, Alice and Sue have established a connection to each other and are certain that they know each other's identities.

Note that GSI uses the Secure Sockets Layer (SSL) for its mutual authentication protocol.

4.3. SHORT-TIME CERTIFICATES

The main advantage of the PKI authentication is that the parties only have to know the public key of the other party's CA to validate its certificate. In a typical system the trusted certificates authorities are known before any authentication happens, thus their public key can be cached at every site. It means the authentication can be established without contacting any online third party service.

The system has only one weakness: if someone steals the private key, then he will be able to claim the stolen identity and prove it successfully in a mutual authentication process. Once the loss of a private key is discovered it can be revoked (the signer CA puts it on the list of revoked certificates – CRL), however contacting the CA for the revocation list at every authentication will eliminate the main advantage of the system.

To overcome this limitation we use the following solutions:

- the certificate revocation lists are cached (open issues are in 4.6)

- authorization system for immediate actions (e.g. disable a certain user)
- *short-time certificates*

Short-time certificates are normal certificates with a very short expiration time, typically 12-24 hours. This expiration time is the typical update period of a certificate authority, thus it makes no sense to revoke such a certificate, because the CRL would be probably update only after the expiration time of the certificate. This practical, *best-effort* solution eliminates the need for the complex CRL update mechanism and also minimizes the impact of stolen private key (especially if the loss is not marked).

The limitations of short-time certificates:

- they cannot be used to sign or encrypt a document, because they are not registered at any place
- they can be difficult to use with services, because it would probably need a restart there

How to get this short-time certificate?

- get a long-time from a certificate authority and generate locally (Globus)
- store a long-time certificate in a server and generate it there (MyProxy/OCR)
- get it directly using another authentication method (kx509)

4.3.1. GSI Proxy Certificate

The usual approach is to get a long-time (1-2 years) certificate from a certificate authority and generate a short-time certificate (called *proxy certificate*) locally using the GSI *delegation* mechanism (see 4.4). This is the currently supported mechanism using the *grid_proxy_init* command. This method implies the local storage of the real certificate, which is convenient for users using a single machine or a central filesystem.

The identifier of the new certificate indicates that it is a proxy ("CN=proxy" string is added to the original identifier). This new certificate is signed by the owner, rather than a CA.

4.3.2. Online Credential Repository (OCR)

In order to create proxy certificates, the user must have access to her private key. Often, this problem is solved simply by storing the password-encrypted key in a file in the user's home directory. This simple approach puts the burden of key-management on the user, who may not be able or willing to effectively protect her key from compromise or loss. Some users need to access the Grid from many independent devices; the secure distribution of the private key to all these devices would be difficult. Sometimes a user needs multiple credentials to access different services, which only increases her burden.

An online credential retrieval system (like MyProxy, see http://dast.nlanr.net/Projects/MyProxy/) stores the users' credentials (certificates and private keys) in a central repository, and automatically issues proxy certificates on the users' request. This centralized credential database considerably simplifies the tasks of both the security administrator and the user base, while at the same time improving the security of the whole system.

The OCR (On-line Credential Repository) may also help when a job takes an unexpectedly long time to finish. The proxy certificates of long-running jobs may expire before the job finished execution. By contacting the OCR server and authenticating with the nearly expired credential, the job may request a proxy with extended lifetime without user intervention (WP1, see Security for Resource Management and Related Services at http://lindir.ics.muni.cz/dg_public).

The planned improvements in the credential repository:

- standardised protocol for normal and administrative commands
- performance and reliability improvements on the server side (database backend and replicated services)
- support for various authentication schemes (e.g. one time password, Kerberos)

4.3.3. Direct Generation

For organisations with well-established authentication systems it can be a viable approach to reuse their existing methods. They could reuse their existing user registration processes to register new certificate requests and distribute the signed certificates, just as they do it with simple user accounts.

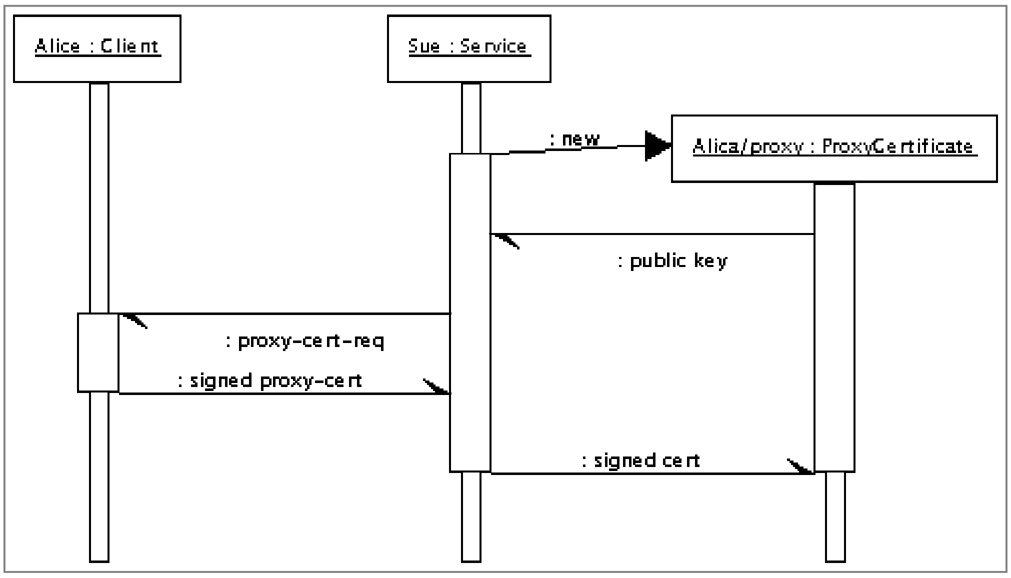
A more efficient way is to trust their existing authentication systems to associate identity to on-line certificate requests. In this case the user basically generates a normal certificate request, but only for a short-time certificate. The user authenticates this certificates request using a local mechanism (e.g. by giving a valid username and password). An on-line certificate authority checks this local authentication and signs the request with its private key. This way the user gets its short-term certificate without a long-term one.

Such an on-line certificate authority should be trusted to sign only short-term certificates.

The idea was implemented using Kerberos as the local authentication mechanism at the Michigan University: they have created a special on-line certificate authority (KCA), which turns kerberos tickets into short-term certificates (see http://www.citi.umich.edu/projects/kerb_pki/).

4.4. DELEGATION

The current version of SSL/TLS does not include a method of credential delegation. Hence, Globus included a mechanism in GSI, which is based on creating a new user proxy certificate over the SSL/TLS channel.

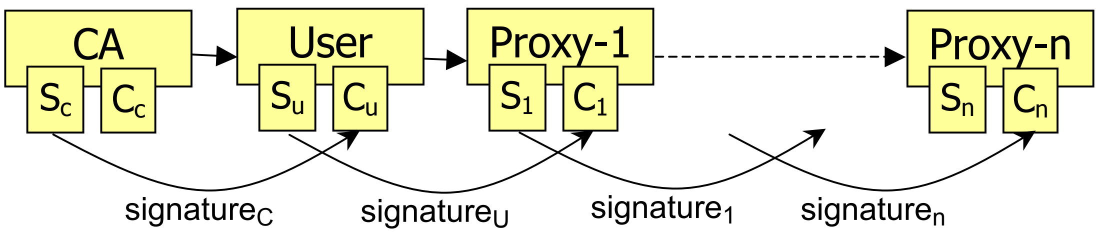


The delegation process is basically an on-line certificate signing procedure, where the user acts as a CA and the service (where the credentials are delegated) acts as the user. To see the process in details we describe a situation where Alice (client/user) wants to delegate her credentials to a remote service (Sue), for example to access her files from that remote computing element.

At the beginning of the delegation process Sue generates a new (proxy) certificate locally. The subject of this proxy is the same as the subject of Alice's original certificate (what she used for the authentication), with "/CN=proxy" added to the name. The typical life of this new user proxy certificate is 12 hours.

Sue sends a certificate request (public key and identity information) to Alice, who checks the content and signs it with her private key. Alice sends back the signed certificate to Sue, who can then use it to utilize Alice's delegated rights. The most important fact in this complicated process that the private keys will not cross the network, even if the communication is encrypted!

If Alice doesn't want to delegate her full identity then they have to negotiate a *restricted proxy certificate.* It is a proxy certificate, with a restriction section added after the identity section. This restriction has to be negotiated before the first step, thus Sue could put it in the new certificate and in the certificate request. The content of such restrictions are application specific.



The text above described only one delegation, however the user's identity can be delegated many times as she goes around in the system. This would process would create a delegation chain, which have some important aspects to note:

- A delegation chain can be validated only if all the certificates are present. In a simple case the certificate of a CA is distributed to every host, but the certificate of the users are not! The certificates, above the current delegated certificate (C_u, C_1, … C_n-1), has to be passed to the service in an authentication (see 4.2).
- A delegated certificate cannot have more right then the certificate above. It means the restrictions are applied to all the subsequent delegated certificates and the expiration time of a delegated certificate cannot be longer then any of the certificates above it.

The first aspect is mostly an implementation detail, but it is significant to note that delegation can make the validation of a certificate very complex.

The second aspect is an important restriction, since we start from a short-time certificate, thus in a few delegation step we might end up with certificates, which last only for a few hours, while we want to run jobs for weeks. To overcome this limitation we can introduce renewable certificates (see 4.3.2).

4.5. CERTIFICATION AUTHORITIES

Note that the policy on which CAs are to be trusted within DataGrid is determined by the WP6 CA Coordination group, and is not carried out by the SCG.

It has been decided to limit the number of CAs considered trustworthy for user and system authentication.

The WP6 CA Coordination Group is in charge of making recommendations to the Testbed 1 (TB1) administrators on which CAs are to be trusted. The group has analysed and discussed the Certificate Policies and the Certification Practice Statements (CP, CPS) published by the CAs – which describe their operative procedures and security practices – with particular attention to the mechanisms used to authenticate the users' requests. The results are summarized in two matrices: *CA Feature Matrix* and *CA Acceptance Matrix* (http://www.cs.tcd.ie/coghlan/cps-matrix/cps-matrix.html).

The following CAs have been included for TB1 authentication:

- CERN,
- Czech Republic (CESNET),
- France (CNRS),
- Ireland (TCD),
- UK (GridPP),
- Italy (INFN-CA),
- Portugal (LIP),
- Netherlands (NIKHEF),
- Nordic Countries (NBI),
- Russia (Moscow Universities),
- Spain (IFAE).

The distributed CA model has been validated using cross-domain submission of jobs on the Grid in May 2001.

The web of CAs has nothing to do with the virtual organisations. Having a certificate signed by one or other CA will not tell us anything about VO membership. These are independent factors.

4.6. CERTIFICATE REVOCATION LIST

The update of certificate revocation lists must happen for every configured certificate authority regularly, but it must not be a bottleneck in the whole system. Currently we simply pull the list from the publication URL (at the CA) at every system, which trusts a particular CA.

The problem with this approach is that it will not scale to 10.000 of machines and irregular update periods for the CRL.

To scale up to 10.000 of machines we will have to use a hierarchical update mechanism.

To handle irregular updates the ideal solution would be "push" mechanism: every interested site subscribes to the CA and when the CRL is updated, it is also sent to every interested site. This update must be asynchronous, to gracefully handle network failures. However such an asynchronous update may even fail, so we would probably have to use the pull method as well.

The simplest (and proven to be scalable) implementation would be a mailing list for the "push" model and a newsgroup for the "pull" model. A mailing list is asynchronous and can handle large number of subscriptions. The news system has a well-established update model for large number of "readers" without a single central server.

idea: create PAM modules to make it automatic and hide it from the user

open issues

- long running jobs, but short lifetime certs -> MyProxy extension from WP1
- Forwarding and copying certificates

Authentication at various services:

- grid services: GSI
- web server: SSL (problem with proxy cert)

- mail/list: signature on the mail
- Login?
- File systems?

Areas to investigate: Kerberos, Windows, MS-Passport, Web-Services security

4.7. SATISFIED REQUIREMENTS

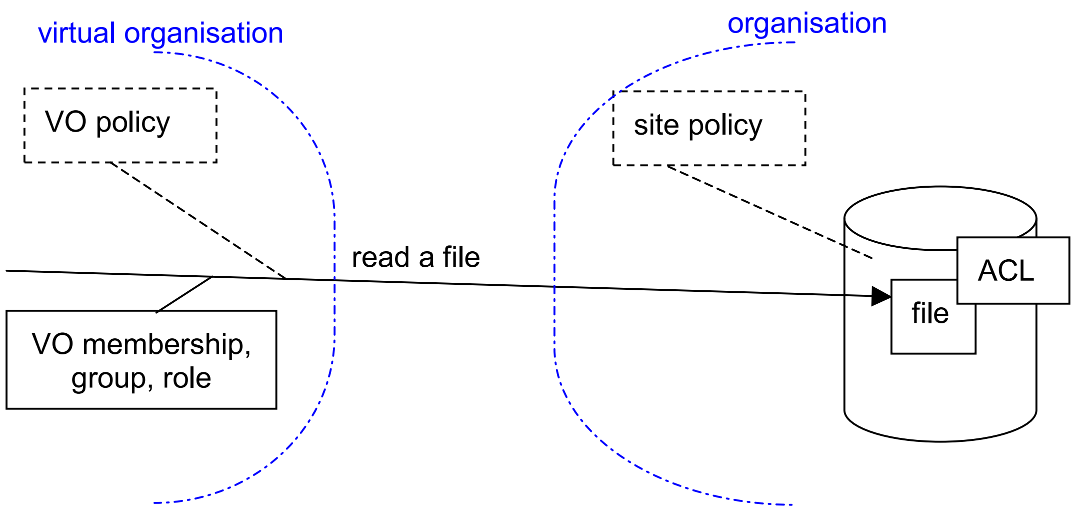
The satisfied requirements from D7.5

5. COMMUNITY MEMBERSHIP MANAGEMENT

Once a user (or any other principal) is authenticated we come to the phase of authorization. In an authorization decision there are typically three main groups of information are used:

- Information associated to the user or the actual session.
- Information associated to the protected service or object.
- Local policy applicable to the situation.

In this chapter we focus on the first set, but here we give a description on where are they managed.



The attributes associated with a user are typically organisation or group membership information, a specific role assigned to the user or any additional attribute, like if she has signed the policy agreement of the virtual organisation. Most of this information is managed by the real organisations.

Permissions associated with protected objects are for example the read and write permissions on a file. This kind of information should be kept close to the protected object to avoid inconsistencies and be able to scale up, even if fine-grained access control is required.

To handle global decisions we have to provide a access point for the administrators of the virtual and real organisations. The most straightforward way is to let them create policies, which are applicable to individual authorization decisions. Such global decision can be to temporarily disable a user, who was abusing a service.

5.1. MEMBERSHIP INFORMATION IN GENERAL

At a given resource or object we would like to specify the set of users, who can make a certain operation, e.g. reading a file. To be able to handle large number of users, we have to group them somehow (not to put every user in an ACL) and test for the membership of a group, when a given user accesses the object.

In a UNIX like system the group membership information is assigned to a user, when she logs into the system. The login process authenticates the user, and then looks up the various directory services (passwd file, NIS, NIS+, LDAP, etc.) for group memberships. The set of groups – where the user is a member – is assigned to the initial process in kernel space, thus this information can be trusted. During the session every permission check is based on this structure without looking up the group membership services.

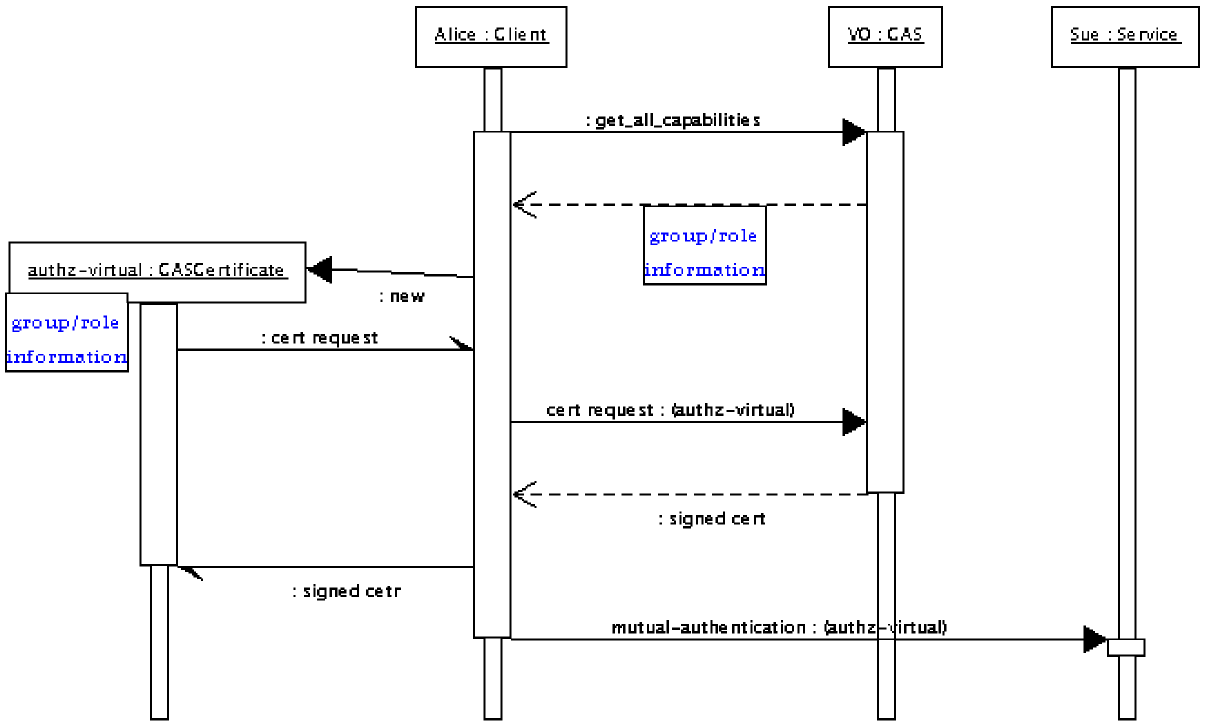
In AFS the authentication process (Kerberos) is separated from the group membership lookup, which happens when a file is accessed. AFS provides its own group service in the protection server (ptserver), which provides mapping of Kerberos principals to AFS users and groups.

In Windows 2000, during the Kerberos authentication, the membership information is also gathered. A Windows 2000 KDC collects the user's security identifier (SID) and the security identifiers of the groups she belongs to, and places this list into the Kerberos ticket's authorization data field. The content of the tickets are protected by cryptography methods, thus this information can be trusted. During the session every permission check is based on the list of SIDs, once a user successfully authenticated itself to the service using the standard Kerberos method.

For the membership management in the grid we would like to combine the power of "offline" validation provided by PKI, with the methods above: at the beginning of a session the grid user acquires its group membership information, which will be stored in a restricted certificate (the restrictions are the groups and roles). The content of this restricted certificate would be signed by the membership service of a VO, thus the services could trust this information, given they trust the signature of the membership service. During the session every permission check would be based on the list of groups in this restricted certificate.

5.2. HOW IT WORKS?

The mechanism behind this kind of authorization is similar to the authentication system: a trusted service (CA in authentication; CAS in authorization) signs a set of information (user's identity; user's groups) with its private key, thus its validity can be checked at the services using the public key of this service. This new certificate will be basically a restricted proxy certificate, which is "delegated" to the user (see 4.4 for details):



Since this is a delegation the same rules are applied: the lifetime of the authorization certificate can not be longer then the lifetime of the credential it is based on, which is typically 12-24 hours (proxy certificate or Kerberos ticket).

In the restrictions section (x509.3 extensions) the following information shall be placed:

- DN of the original user, for accounting, auditing and logging purposes at the services or for fine-grained, user level authorization.
- VO identifier, which could be the DN of the issuer service. It would sign that the user is a member of this virtual organisation. It will also let the user to pass multiple authorization credentials (belonging to different VOs) to a service, which could choose the appropriate one.
- Sequence of the groups, where the user is a member – all of them.
- Sequence of the roles, what the user requested. Some roles should be requested explicitly by the user (e.g. administrative roles), maybe with some additional authentication.
- URI of the issuer membership service. This would provide a "callback" reference for a Computing Element if it runs a long job and the authentication and authorization credentials has to be renewed.

The encoding of the role and group membership information is an open issue.

One opinion (Cal, Andrew) is to preserve the "type" information by encoding the group and role information into a structure, which could be represented in XML for example like this:

```
<group>
<vo>/O=Grid/OU=DataGrid</vo>
<groupname>CMS</groupname>
<group>

<role>
<vo>/O=Grid/OU=DataGrid</vo>
<rolename>replica-admin</rolename>
<role>
```

(of course it would be ASN.1 encoding in a x509.3 certificate)

It would only affect the handling of this information in the services. This additional "type" would let a service to handle users, groups and roles separately. Also it would make the handling code more complex in most cases.

Therefore the other opinion (me ☺) is to treat groups and roles (and whatever comes later) as first class principals in our authorization system and assign them a unique DN in the namespace of the virtual organisation:

```
/O=Grid/OU=DataGrid/Group=CMS
/O=Grid/OU=DataGrid/Role=replica-admin
```

It would simplify the handling of this information and also the management interface.

As a unification of these opinions, we can call these pieces (user's DN, VO's DN, group and role name) *capabilities* and make distinction later only if it is necessary.

This membership service would effectively replace the virtual organisation's LDAP server by providing the same information in a more secure format. Actually the current LDAP server might become the backend database of this service.

5.3. CAS

The *Community Authorization System (CAS)* from the Globus team does almost what we expect from a membership service, but it does a bit more. The CAS server internally maintains its own database of users and groups, but it also maintains information on protected objects, providing the very last step of the authorization as well. It basically means that a service gets a “yes” or “no” decision every protected object, so its only purpose is to apply the decision.

We believe this approach have some problems:

- Doesn’t scale well, if millions of files have to be protected at hundreds of sites.
- May loose consistency with the protected objects: for example a file server looses the connection with the CAS server and a given file is deleted and later added by another user (with the same name). If the original owner meanwhile asked for a write-permission, it would be granted for 24 hours, although the file is already owned by somebody else. A *dumb* file server would still accept the original owner.
- No place for local decisions and permissions: for example a local administrator might *deny* access for a certain user for abusing a service. A local administrator might also *allow* access for local users, who are not member of any virtual organisation.

In summary we would need *less* features in CAS, it should simply put the VO and group membership information into the created certificate. It should also put the DN of the original user into this certificate.

The good news is that we could start experimenting using the existing CAS server by placing pseudo objects into its database; quote from Laura Perlman:

> You can actually get this functionality now, without any modifications to CAS, if you consider "claiming membership in a group" to be an action:
>
> 1. Once, on the CAS server, create a service type (say, "edg_group_service") and add the allowable actions (say, "claim_group_membership" and "claim_role_membership") for that service.
>
> 2. For each group that you want to create:
>   - create a new object inside the CAS associated with the group
>   - create the new group
>   - grant "edg_group_service/claim_group_membership" permission on that new object to the group
>   - populate the group within the CAS as usual
>
> Then when you're evaluating the ACL, instead of asking "is this user a member of group XYZ", you'd ask "does this user have the right to assert membership in group XYZ".

The described solution would provide us the possibility to start working on the authorization solution in the services. We could then wait for the proper solution from the CAS team or write our own version.

5.4. COMPOSITIONAL COMMUNITIES

Looking at the other side of the membership service we could find similar problems to the protected objects. A virtual organisation is a composition of real organisations (or their subgroups) providing a common interface for the unified set of resources for all of their users. But where do these users come from? They are (typically) members of the real organisations, where they are already registered and placed in various groups.

We should avoid the replication of this membership information:

- To scale up to large organisations with ten thousands users.
- Keep better consistency with the current database.

(Of course these are not hard constraints, since ten thousand records can be replicated easily and the membership databases are usually updated only on a daily schedule, but we should keep in mind that here could be a problem in the future.)

5.4.1. Organisational Membership Service

The easiest way of avoiding the replication is to provide the membership service by the organisation itself. The local management can put a thin layer in front of their user management database, which implements the protocol of the membership service and encodes the internal membership information in the virtual organisation's format.

This thin layer would be based on the local authentication method, because it would gather the membership information from the local database (group file, NIS, NIS+, LDAP, Active Directory, AFS ptserver, etc.).

This service could even provide the first short-time certificate (see 4.3.3 for a Kerberos solution) for the user solving the management problem of long time certificates.

5.4.2. Mapping

The organisation group names has to be mapped into the virtual organisation's common group names.

For example the LHC virtual organisation would have two real member organisations with similar group structures:

- CERN: CMS and ALICE groups
- INFN: g-cms and g-alice groups

The members of these groups would be naturally mapped into the CMS and Alice groups in LHC.

Both CERN and INFN could configure their local services to put these names into the issued membership information.

Problem 1: What happens if any of them participate in another virtual organisation, where the groups should be mapped to other names?

One possible solution is to add this new configuration to the local service and let the user select the appropriate one when requesting an authorization certificate. Every organisation has to modify its own mapping, which participates in the new VO.

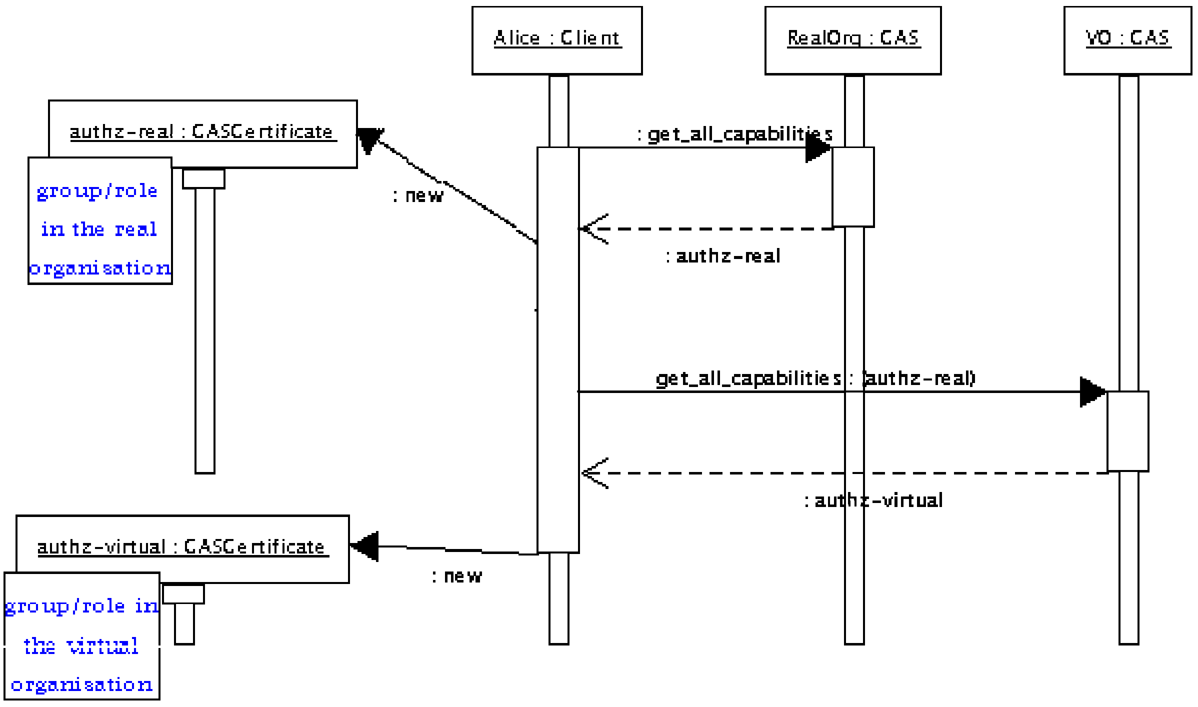
Problem 2: In a virtual organisation every service has to accept the authentication and authorization certificates signed by the users' certificate authorities and membership services. If a new organisation is added to the existing VO, then every organisation has to update its own trust database to include the certificates of the new one.

One possible solution is to provide this list (or the URIs of the certificates) at a central place in the VO, thus every member could update its database based on this reference.

There is another solution for these problems: bringing back the membership service, but with a different purpose. It should provide

- a mapping from various organisational groups to the common group names, and
- serve as a central point of trust – the services has to trust only this certificate.

Using this central mapping service, the "login" to the virtual organisation becomes a two-step method:


Alice : Client
RealOrg : CAS
VO : CAS
authz-real : CASCertificate
group/role in the real organisation
: new
: get_all_capabilities
: authz-real
get_all_capabilities(: authz-real)
: authz-virtual
authz-virtual : CASCertificate
group/role in the virtual organisation
: new

The detailed steps of these certificate delegations are in 5.2 and 4.4.

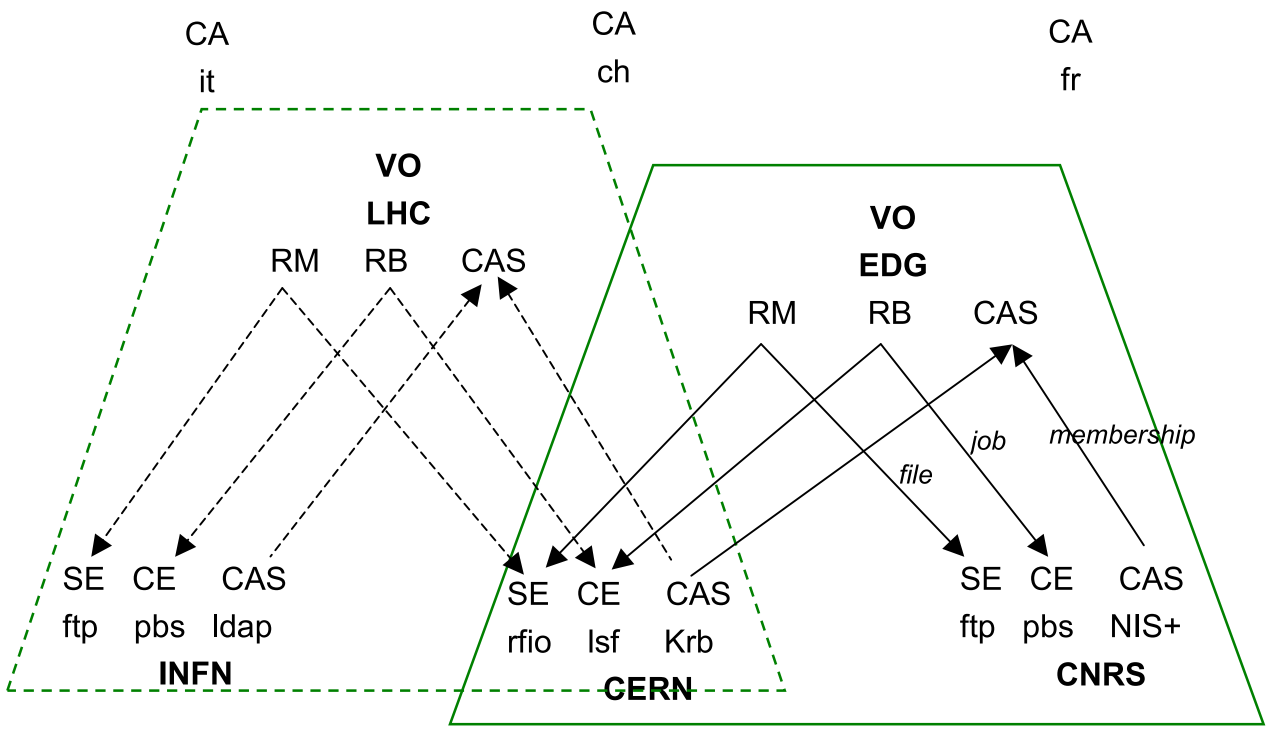
The user first gets a certificate at the local membership service, which adds the local group names and roles to the restrictions. The user presents this certificate to the virtual organisation's membership service, which accepts the information (the certificates of the local membership services are registered as trusted), maps the group and role names to common names and signs the new certificate. All members of the virtual organisation would accept the new one.

The central services might provide additional functionality over simple mapping functions by maintaining a private database for additional users, groups or roles.

This service could also provide a place for user-managed groups.

5.5. EXAMPLE

All services in this section are fictitious. Any resemblance to actual solutions, working or planned, is purely coincidental.



A detailed example is given in this figure: three real organisations (INFN, CERN, CNRS) are working together in two virtual organisations (LHC, EDG).

INFN uses LDAP to store the user information and Kerberos to authenticate their users. Their users have long-term certificates, which are also published in the LDAP database. A user may get its membership information by creating a short-term certificate (grid_proxy_init) and contacting the local membership service. Since the long-term certificates are published in the LDAP database, the service can associate it with the real user and look up the groups. The user would turn then to the LHC membership service and obtain a certificate for this virtual organisation. This certificate can be used at the resource broker to submit a job into INFN's PBS or CERN's LSF system.

CERN uses Kerberos to authenticate its users and the AFS protection server to assign groups to them. CERN users have only short-time certificates, which they may obtain through the kx509 service. This service also queries the protection server's group database and places the result into the restricted certificate. The user could use this certificate at two virtual organisation's membership service to obtain VO specific credentials. A user may obtain both VO specific credentials and use them at the appropriate services. The user could use the local storage element with the LHC, EDG and even with the CERN credential.

CNRS uses NIS+ to manage its users, so a user have to authenticate to the local membership service using a password, even if she has a long-term certificate. To simplify this process they create a special PAM module, which obtains this grid membership certificate at login. After logging into the system the user has to log in to the virtual organisation by contacting its membership service. With the VO credential the user could replicate a file from the CNRS storage element to CERN's mass storage system.

5.6. SATISFIED REQUIREMENTS

The satisfied requirements from D7.5

6. ACCESS CONTROL

In the last phase of the authorization the system has to gather the information associated to the protected service or object, for example read and write permissions on a file, or stop and resume permissions on a job.

In a closed environment these attributes are simply associated with the object (like file permissions), but in the grid these are partially moved to the level of the virtual organization to be able to impose higher-level decisions. For example a storage element cannot allow modifications to a local copy of a replicated file, except through the replica manager, which is able to replicate these changes to every copy.

The easiest solution would be to move the barrier up to the level of the virtual organization and treat the system as a closed environment. This way the global decisions could be imposed by overriding the security system with some special privilege (like root can read anything in a UNIX). The problem with this approach is that virtual organizations might overlap at certain sites, thus we have to be able to separate their global decisions, and not to allow overriding each others security control ("root" of one VO cannot read a file from another VO).

The duality of these requirements affects the major groups of the services: job control (WP1, WP3 and WP4), file management (WP2, WP5) and even to networking (WP7).

6.1. ACCESS CONTROL IN GENERAL

At a given object we would like to specify the set of users, who can make a certain operation. The size of the set of users may vary from one or two fixed entries to an unlimited list. The operations are usually chosen from a fixed set, since they have to be implemented in a service, although a straightforward implementation leaves space for further extensions.

If the authentication process already assigned membership information to the user, then we don't have to worry about those details, so we can treat users, groups and roles as capabilities and put their identifiers in the set of users.

In a UNIX like system files can have only three different "capabilities" assigned: one user, one group and the others (which effectively means all the users, who can access the system). All of these entries can have permission for only three operations: read, write and execute. These operations have different semantics depending on the protected file (e.g. "read a directory" means listing its content). There are three slots left for extensions, which are usually used for the set-user-id, set-group-id and temporary-dir privileges. In some modern filesystems this simple access control can be extended by the more general access control list (e.g. JFS, XFS, ext2/3, Solaris ufs).

There are other objects, which have the same access control mechanism applied: System V. IPC tools, named pipes, named network sockets.

A process (job) has only one element in the set of users: the owner of the process, who can stop, resume or kill it. Anyone can monitor the status of a process, who has access to the system. The owner of the process also the *security credential* assigned to it: the process can act in the name of the user – delegation of access rights without any restriction.

The networking side has a very simple access control: under 1024 only the root user can open up a port, but above that limit anyone.

The root user, who can pass over every access control check, solves the concept of global operations.

various bits – should be cleaned up and reorganized by operating systems:

The concept of access control lists is also used in the filesystem of VMS (as an extension to the standard permissions), in NTFS (uses only ACL) and in AFS (uses only ACL, but at directory level).

Some systems separate the authenticated and not authenticated users: VMS and NIS+ gives an extra slot in the standard set of permissions (4 slots instead of 3); AFS puts them into two pseudo groups (system:anyuser and system:authuser), which can be used in any ACL entry.

Some systems give security check override permissions for all or certain operations: in AFS the members of system:administrators group can modify any ACL; in VMS and Windows NT a user or a process may get a *privilege* to override a certain check (e.g. *backup priv* overrides the check for the read permissions); in AIX it is called a *role* (e.g. *backup role*).

In Java one can assign detailed permissions to every port (listen, accept, open), which could allow opening a port under 1024 even for an average user.

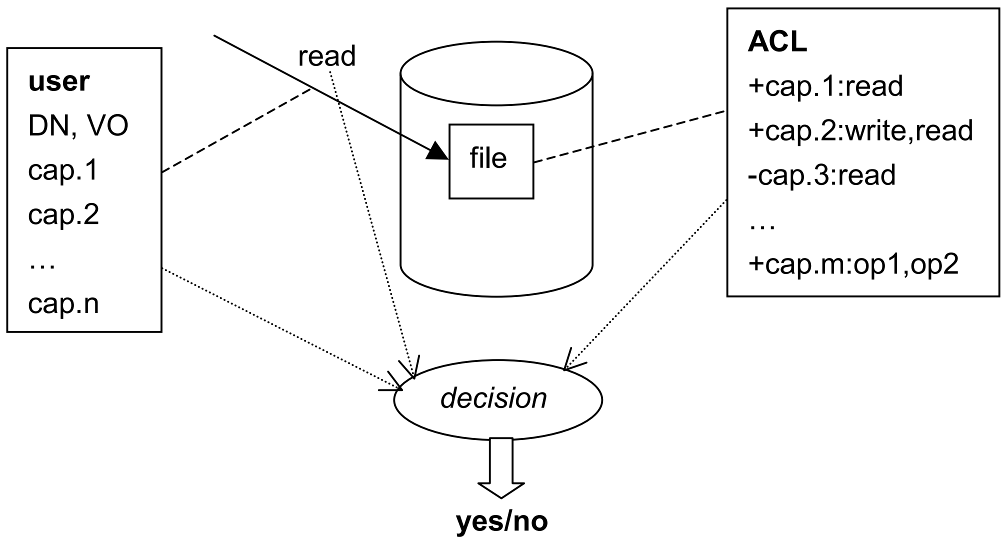
For the access control in the grid we would like combine flexibility with simplicity: we use only access control lists (no standard set of permissions) with capabilities (user, group or role).

Global operations are granted by assigning special roles to users in the VO's membership service and adding this role to every protected object in a VO (in a storage element we can not allow *read* for another VO's *backup role*).

The set of allowed operations are depending on the corresponding service, but there must be at least the *administer*, which enables a user with the appropriate capability to modify the ACL on the object.

6.2. HOW IT WORKS?

When a user is authenticated and got her credentials from a community membership service, a set of *capabilities* will be placed in the restricted certificate. These capabilities describe the user's original identity, the user is a member of a given group or a user plays a given role in the organization. Contacting a service the user's certificate is validated and the capabilities are extracted for the authorization system (→ *list of capabilities*)



The user requests a certain *operation* on a protected object. The service will fetch the access control list (ACL) associated with the object (→ *entries in an access control list*) from a service specific location. In a storage element it can be an associated file or attribute on a file, but it can be also stored in a separate database.

An entry in the ACL can be either *allow* (+) or *deny* (-) entry for a matching *capability* with the associated list of *operations*.

To be able to define global ACL entries (such as "everybody can read this file") the user's capability list is extended with two special elements: anyone and authenticated user. Their encoding can be:

```
/O=system/DN=anyone
/O=system/DN=authenticated
```

Open issue: If the order of the entries in the ACL is significant it can express more complicated cases, although its representation might be difficult. If the ACL entries are stored in a relational database, their ordered lookup is not guaranteed by default. The management of this order also complicates the API, so we must show important use cases in support of this approach. – for now we assume the order is not significant.

The capability "matching" can be defined as simple string comparison, since the "user, who is the member of G group" is encoded in placing G group's capability (either a special DN or a structure) in the user's capability list and in an ACL entry.

One capability of the user's credential matches an ACL entry, if the user's capability matches the capability in the entry (i.e. equals) and the requested operation is in the list of the operations in the entry. The request is granted if any of the user's capabilities matches an *allow* entry and none of the capabilities matches a *deny* entry. It could be expressed in the following formal algorithm:

```
granted = no
for every user-cap in list-of-user-capabilities
      for every entry in access-control-list
            if entry.capability = user-cap and
               operation is-in entry.list-of-operations then
                  if entry.allow then
                        granted = yes
                  else
                        return no
                  end if
            end if
      end for
end for
return granted
```

Of course the decision making process could apply a more sophisticated algorithm with caching intermediate results or reordering the search according to typical cases.

The decision making process could also use other information, such as the policy of the real or virtual organization or the local site.

6.2.1. Access Control List

There are a few things, which can be defined without looking at the actual services.

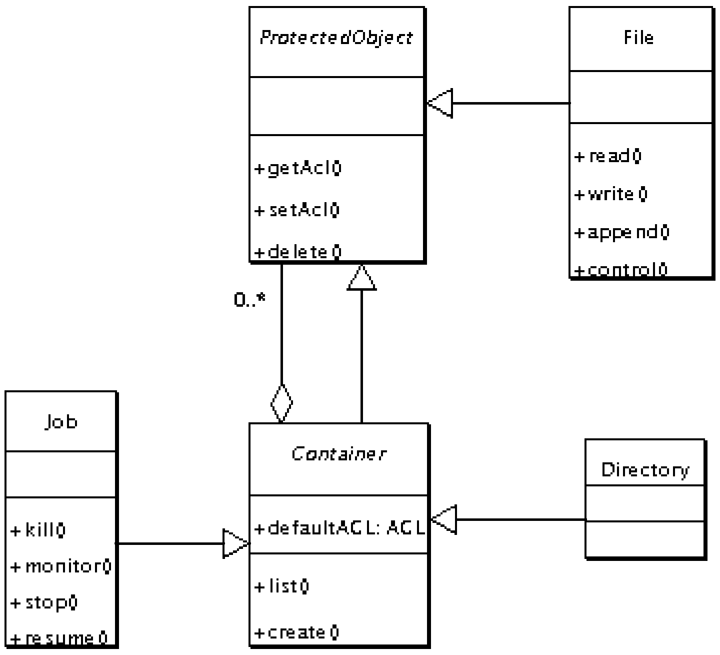
An ACL must have a *setacl* operation, which is the permission to modify the ACL itself. When someone (→ *user*) creates a new object with the associated ACL, then there should be at least the following initial entry: *+user:setacl*

It might happen that the only administrative user deletes its own permission. Therefore it is desired that a special role in every service (e.g. *security-administrator*) could override the check for this permission for the locally managed objects. This functionality can be easily added by a service specific policy (see 6.2.2).

An ACL must have a *getacl* operation too, which is permission to read/get the ACL itself. It is often desired that only the administrators could read the access control list, but it could be relaxed that every authenticated user or everyone may read it.

A protected object should have a *delete* or *remove* operation, which permits the removal of the object itself, but not the objects it might contain.

In a standard filesystem it is often an operation on the enclosing directory, not on the file itself, but it is necessary for the SE-RM semantics to be able to control this on file level.



If an object can contain other objects – it is a *container* – (e.g. files in a directory, processes under a parent process), then it should have a *new* or *create* operation. It is the permission to create a new object inside the current container.

A container must also have a *list* operation, which permits the listing of the contained elements. By default everyone with *new* permission should have this permission as well.

If such an operation is allowed, then the object should have a special ACL list with the *default entries*. This list should be applied on the new objects. If the new objects may aggregate other objects, then the default ACL list should also be copied to the new object.

If there is no *default ACL* list, then the new object simply inherits (actually it is copied) the entries from the enclosing object.

The creator of this new object should always have administrative right, regardless the default ACL.

TODO: have a look at the POSIX semantics

6.2.2. Policy

pattern+ACL, pattern to select the applicable service/object, e.g.: '/path/**':-:Joe:read Joe cannot read any file under /path (recursive)

6.2.3. GAA-API

GAA-API looks like a good candidate to hide the decision making process in the access control. It also has hooks to download policy objects, which could be used later.

Beside the GAA-API there are other simple authorization interfaces, like the AZN-API. We might consider using one of them in favor of the simplicity.

GAA-API is being implemented by the Globus team in connection with the development of CAS. Depending on our usage of CAS service we might have to modify this code.

There is not known implementation of the GAA-API in Java.

6.2.4. VO-level Roles – Mutual Authorization

To impose global decisions on the level of the virtual organization, we have to assign permissions to the corresponding services (e.g. resource broker, replica manager).

Since a service is made of plenty of individual servers at the sites of real organizations, it will have plenty of certificates proving the identity of the individual servers. Managing all of these certificates in the accessed objects would be cumbersome, so we should aggregate them under a single identification, under a single role.

It means that a server process has to acquire its restricted certificate from the community membership service, if it wants to operate with objects associated with the service.

As a side effect we have two benefits:

- The client may authorize (accept) a server based on its role – *mutual authorization* beside our mutual authentication.
- If a server is compromised (or we are just precautious) its authentication certificate can be easily replaced, because only the membership service has to be updated – it is the only service, which checks the server's key and then it assigns a role signed with its own key.

The sad side that we will have to modify the services to make this initial authorization and also make sure that they are registered in the virtual organization's membership service.

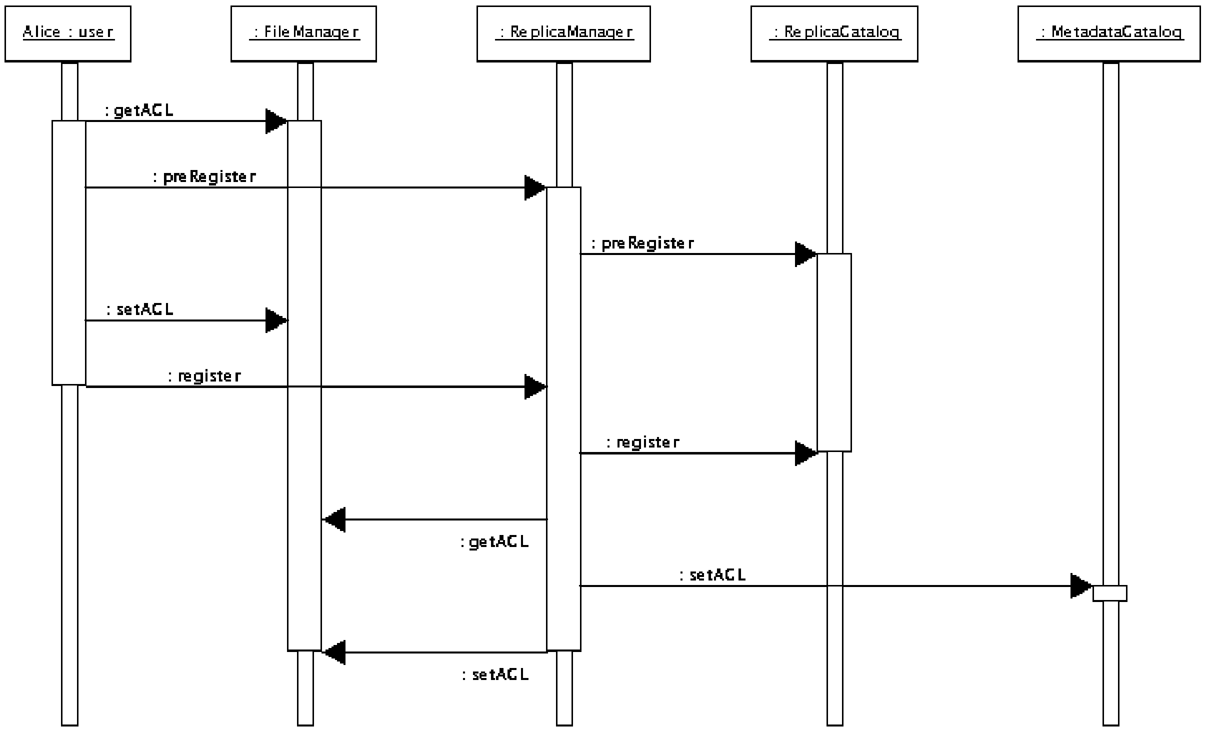
starting a new server:

1. generate a server cert-req and sign it with the site's private key
2. configure the URI and certificate of the VO's membership service and start the server
3. server connects to the membership service – checks its identity
4. membership service checks its identity by validating the site's signature and challenging the server's certificate (site must be registered)
5. membership service assigns the appropriate role
6. server creates a new certificate with this role and sends this request
7. membership service signs the authz cert.
8. the server starts a new connection by presenting its new cert

6.3. FILE MANAGEMENT

For file access, the intention is to rely on the Replica Manager to perform authorization for grid files and for the SE to perform authorization itself for "private" files.

Every file in an SE have an ACL, but if it belongs to a VO, then there would be one entry for the RM (full administrative permissions) and "read" entries for the earlier entries, which had at least "read" permission.



Let's see the process of registering a file in the replica manager:

1. Alice asks the current ACL of the file – *getACL* (+Alice:read,write,admin)
2. Alice asks the replica manager service if the file can be registered and gets its *RM-role* – *preRegister*
3. If it will be registered, then Alice gives administrative permission to the *RM-role* in the Storage Element – *setACL* (+Alice:read,write,admin; RM-role:admin)
4. Alice registers the file – *register*
5. The replica manager registers the file – *register*
6. The replica manager gets the current ACL from the Storage Element and stores it in the Metadata Catalog – *SE.getACL, MC.setACL* (+Alice:read,write,admin; RM-role:admin)
7. The replica manager removes the modification permissions from the user in the Storage Element – *SE.setACL* (+Alice:read; RM-role:admin)

It means the SE will still maintain authorization even for grid files. It is useful when the replica manager is not accessible, because with these special ACL the read operation would work.

At the registration the ACLs are changed and the VO level access rights will be controlled by the replica manager (stored in the metadata catalog). From this point the ACL can only be modified only through the RM, which would provide this functionality in its interface.

In an SE and RM the ACLs are in the metadata of a file: in the RM it is in the replica metadata catalog, in an SE it is in a local database (in case of a mass storage system) or in the local filesystem (e.g. /grid).

Replica Manager/Spitfire: it would simplify the authorization code, because they would not have to deal with group/role management. They already base their authorization on certificates, so it would not cause major modification in the design and the implementation.

6.3.1. Operations

TODO: get the list of operations from Jens

6.3.2. Compatibility with FTP

There is no support in FTP for ACL, so we will not be able to modify them through this protocol. But this is not the only limitation of FTP. One cannot transfer file metadata, ask about the free disk space, preallocate some disk space or do other storage specific operations, like locking or pinning. Since the functionality of a storage element is much richer, than the functionality of an FTP server, it has to implement its own interface, so it can also add support for ACLs as well.

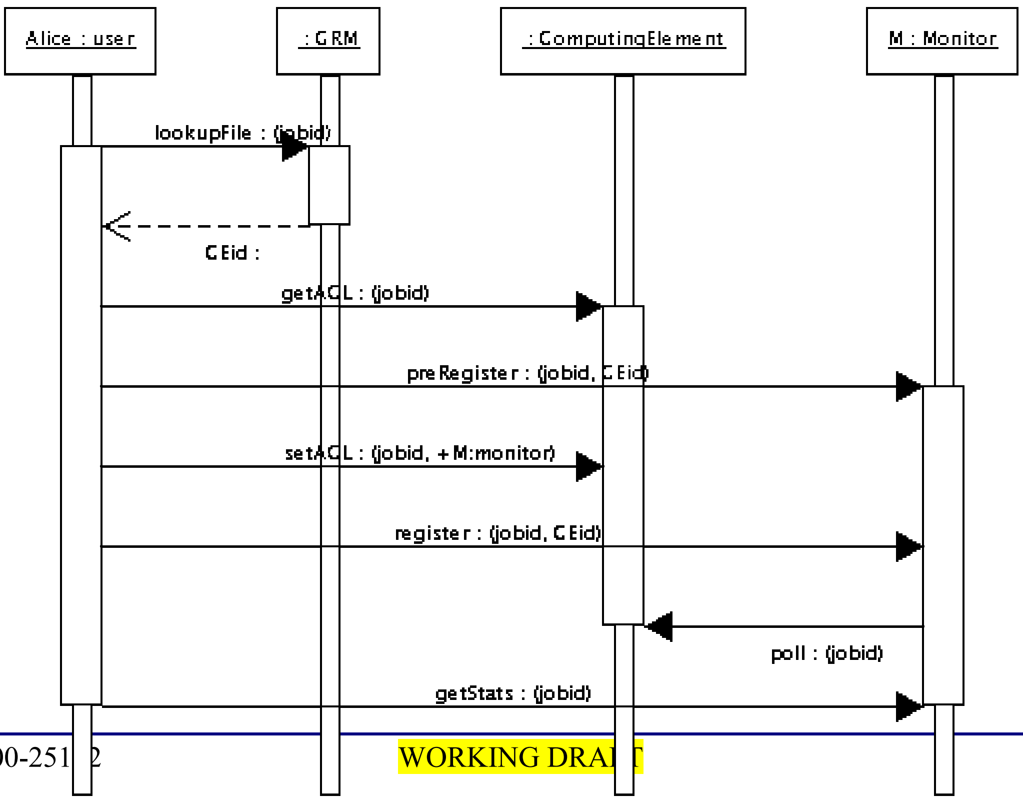
The default behaviour of the ACL is the same as the current permissions in a filesystem: if one has a permission to create a file in a directory

6.4. JOB CONTROL

Operations: start, stop, resume, kill, monitor, administer, credentials

When a user submits a job, she would assign stop, resume and kill permissions to the resource broker.

Credentials: management of credentials associated with a job. With a simple job only the user could assign its proxy certificate to the job. For long running job these credentials have to be replaced before their expiration time, thus the user may assign the permission to a trusted service to renew and replace these certificates.



Monitor: this permission would allow the monitoring of the status of this job. A user may monitor the job directly or use the event collecting service to aggregate some details. In the later case the user would assign monitoring permission to the job for the event service role the permission to get the aggregated date from the monitoring service would be managed by that service.

It should be similar to the replica manager – storage element interaction.

6.5. NETWORKING

Operations: connect, accept, listen, transfer

The speciality of the network operations that the user has to get network permissions on both end of the connection. The effective permission would be the intersection of the permissions on both ends.

These permissions could be checked before submitting a job, thus a resource broker could make appropriate decisions. For example:

1. A-end: connect, 10Mbit (just outgoing connection – probably NAT); B-end: accept, connect, 100Mbit – the intersection means, that the resource broker has to initiate the connections from A-end and these connections will not go faster than 10Mbit
2. CE-end: connect; submit-end: accept, connect – if an interactive job is submitted then the connection has to be initiated from the job, back to the submit host.

One might add other metrics as well, like encryption possibility.

6.6. SATISFIED REQUIREMENTS

The satisfied requirements from D7.5

open issues:

Enforcement

mapping into the local system

7. LIFECYCLE OF OBJECTS

7.1. CERTIFICATE AUTHORITY

7.2. VIRTUAL ORGANISATION

7.3. GROUP AND ROLE

7.4. RESOURCE AND SERVICE

In a grid authentication process both parties are identified by certificates. In a typical use case a user connect to a service and requests an operation. The user usually identifies itself by a (time limited) proxy certificate, which is signed by the user's real certificate, which was issued by the CA of the user's virtual organization (CA -> user-cert -> proxy-cert). The service currently identifies itself by the certificate issued for the host. If there are several services running on the same host (probably in different security domains, aka different pseudo users), they have to share this common key.

The common key doesn't allow the identification of an individual service and it also has to be replaced for all services, if one gets compromised. It would be a good idea to have separate certificates for each service.

7.4.1. CA based solution

The CA of the virtual organization issues certificates for each service on each host.

Pro:

- it works with the current clients
- the cert/CRL handling is at the CA Contra:
- dynamic installation and stopping of services at hosts induces a lot of traffic at the CA

How: issue a certificate with service@hostname SN, e.g. gdmp@testbed.cern.ch

7.4.2. Host based solution

The host receives a certificate, which can be used as a mini-CA to sign certificates for the services.

Pro:

- a host may dynamically install or stop services without further administrative contact with the CA
- host certificate is only accessed by root, thus it would be "more protected"

Contra:

- the clients probably have to be modified to check the full certificate chain (similar to the problem of checking proxy certificates)
- the host acts as a CA for long term certs, thus it has to handle a CRL to revoke compromised certificates

How: issue host certificates with the ability to issue service certificates. Using the X509 name constraints this would then allow the maintainer of the host to issue service certificates for that host only. The SN should be similar to the CA based solution, service@hostname.

For example a host SN=/O=Grid/O=CERN/OU=cern.ch/CN=testbed001.cern.ch and a GDMP service the new SN=/O=Grid/O=CERN/OU=cern.ch/CN=gdmp@testbed001.cern.ch

The host certificate should have the following constraints:

```
      BasicConstraints {            (see RFC 2459/4.2.1.10)
            cA=TRUE
            pathLenConstraint=0
      }
      NameConstraints {             (see RFC 2459/4.2.1.11)
            permittedSubTrees {
                  { base=<CN> }     (e.g. base=testbed001.cern.ch)
            }
      }
```

Open issues:

- lifetime of the service certificate (default: same as host cert)
- CRL for the service certificate issued by the host (distributed by each host or centrally by an organization?)

more information on certificates in RFC 2549 <ftp://ftp.isi.edu/in-notes/rfc2459.txt>

7.5. USER

7.6. PERMANENT OBJECTS

8. ANNEXES

8.1. ENTITY, GROUP AND ROLE REPRESENTATION

We might give unique names to roles and groups by prefixing the DN of the VO's CAS service. This way a WP2 group and an RM role would be:

/O=Grid/OU=DataGrid-VO/CN=cas/Group=WP2

/O=Grid/OU=DataGrid-VO/CN=cas/Role=RM

It would simplify the generation of a CAS certificate and the syntax of an ACL entry as well. The CAS credential would merely contain a sequence of DNs:

```
/O=Grid/OU=CERN/CN=XYZ                # user's DN
/O=Grid/OU=DataGrid-VO/CN=cas         # VO membership
/O=Grid/OU=DataGrid-VO/CN=cas/Group=WP2    # group membership
/O=Grid/OU=DataGrid-VO/CN=cas/Role=RM      # role
```

If someone contacts this service with a CAS credential containing such a sequence, this list can be appended to the original one, giving the possibility to express multiple VO memberships in a single credential.

And the ACL would be pairs of DN and operation:

```
<entry> <level>read</level> <dn>/O=Grid/OU=CERN/CN=XYZ</dn> </entry>
<entry> <level>read</level> <dn>/O=Grid/OU=DataGrid-
VO/CN=cas/Group=WP2</dn> </entry>
```

(It was previously:

```
<person> <level>read</level> <dn>/O=Grid/OU=CERN/CN=XYZ</dn> </person>
<cas> <level>read</level> <dn>/O=Grid/OU=DataGrid-VO/CN=cas</dn>
<group>WP2</group> </cas>
)
```

It would also give us smooth transition using gridmap files: a group or role

DN could be mapped in a specific userid.

8.2. ACCESS CONTROL LIST

8.2.1. XML Representation

8.2.2. ACL API