Authentication and autorisation prototype on the microgrid for medical data management

Ludwig Seitz, Johan Montagnat, Jean-Marc Pierson, Didier Oriol, Diane Lingrand

Authentication and authorisation prototype on the µgrid for medical data management*

Ludwig Seitz[1], Johan Montagnat[2], Jean-Marc Pierson[1], Didier Oriol[1], Diane Lingrand[2]
[1] *LIRIS FRE CNRS 2672, INSA de Lyon, France, http://liris.cnrs.fr/*
[2] *RAINBOW-I3S, UMR UNSA-CNRS 6070, Nice, France, http://www.i3s.unice.fr/~johan*

Abstract

This paper presents *µgrid*, a light weight middleware for grid applications, and focuses mainly on security issues -more specifically on the access control to resources - that are critical for the gridification of many medical applications. For this purpose, we use *Sygn* as a distributed, certificate based, and flexible access control mechanism, which has been fully integrated in µgrid. We discuss the advantages of the solution compared to classical grid approaches and the limitations of the final architecture.

Keywords

Grid middleware, authentication, access control, medical data

1. Grids for managing medical images and data

Grids are recognised as a promising technology to face the challenges of medical image processing and archiving [1][2][3]. However, data security and patient privacy remain a primary concern for porting medical applications on grids [4]. In a distributed and loosely controlled environment, data protection requires specific attention to avoid data leaks and to ensure the patient privacy. In an hospital, medical data should only be accessible by accredited persons. On a grid, data are distributed over different sites – for storage, archiving, or processing needs – that are administrated by different entities. Data may be sent outside of the health care centres (*e.g.* to a computing centre) and are thus exposed to a potentially much wider community of users or administrators. Therefore, strict protection of medical data based on users authentication and fine grain access control is mandatory. As grids serve an international and cross-organisational community of users, authentication should be organised appropriately to span across those boundaries.

In this paper, we address the problem of access control of medical data. Given the number of actors involved in health care (nurses, physicians, secretaries, etc) and the complexity of the relationship between the administrative entities (hospitals, medical centres, research units, etc) hosting these actors (several physicians of different units or even hospitals may be involved) it is important that the access control system is flexible enough for common medical and clinical usage of the data. This implies the possibility for permission management at single user level as well as for groups of users.

Many access control systems have been studied in the literature so far. CAS [5] and VOMS [6] propose grid specific access control services. In CAS, all access rights are centralised at the CAS servers. In VOMS the servers store only group memberships while the group access rights are stored on the resource sites. Both approaches have the drawback of introducing a highly valuable target for attacks, additionally to the resource server and the user's machine. If a CAS server is attacked successfully the attacker can grant himself any rights he wants on all resources administrated by the CAS. In VOMS the situation is

* This work is partly supported by the Région Rhône-Alpes project RAGTIME and the French ministry of research ACI-GRID program (MEDIGRID [16]).

not really better, since a successful attacker can make himself member of any group and then use the group access rights.

Akenti [7] and PERMIS [8] are both access control services that use certificates to encode and protect their access permissions. Therefore both systems considerably reduce the amount of vulnerable data since only the information about the *sources of authority* (SOA), which are allowed to issue valid access permissions, has to be protected. However in both systems the access control server handles the storage and retrieval of permissions. This creates a point of failure in the access control process, since the permission storage may break down and therefore permissions may become inaccessible. Replication of permissions is not a good solution to this problem, since this requires additional mechanisms to keep the permissions coherent on all replication sites.

In an application dealing with medical data, fast access to medical files is sometimes required to respond to an emergency: failure of access due to a third party is unacceptable. An authorised data access should only fail if either the user or the resource is disconnected from the grid. Our access control approach, *Sygn*, stores permissions with the users whom they concern. This makes it possible to eliminate any third party from the access control decision process. Furthermore this approach allows ad-hoc creation and direct transfer of permissions to the users, with no need of interaction with a third party permission storage.

Setting up real scale grids is resource and time consuming. This makes the prototyping of a service such as the Sygn access control system hardly feasible. The human cost for setting up a full scale grid has often been largely under-estimated, failing to reach the high expectations of many grid end users. To enable prototyping and testing at a low cost, we have developed a simple yet rather complete grid middleware called µgrid (*microgrid*). It relies on robust standards such as TLS and X509 to enable transport security and authentication. µgrid provides a lightweight interface for prototyping and testing grid enabled features, without pretending to be a solution for production grid deployment. The µgrid middleware has been used for executing and optimising medical image processing applications such as content based search in medical image databases [9].

The remaining of this paper first describes the µgrid middleware architecture and capabilities. The Sygn access control mechanism is then introduced. We present how Sygn and µgrid services can interoperate to build a secured data management system for medical data. Finally we conclude and discuss perspectives in medical grid deployment.

2. µgrid, a lightweight grid middleware

The µgrid middleware is designed to be very light weighted in order to remain easy to install, use and maintain. It was designed to use clusters of PCs available in laboratories or hospitals. Therefore, it does not make any assumption on the network and the operating system except that independent hosts with private CPU, memory, and disk resources are connected through an IP network and can exchange messages via communication ports. This middleware provides the basic functionalities needed for batch-oriented applications: it enables transparent access to data for jobs executed from a user interface. It is written in C++ and is built on a few standard components such as the OpenSSL library [10] for authenticated and secured communications, and the MySQL C interface for access to a database server [11].

The code of µgrid is licensed under the GPL and is freely available from the authors web page (see affiliation above).

2.1. Architecture design

In the µgrid middleware a pool of *hosts*, providing storage space and computing power, is transparently managed by a *farm manager*. This manager collects the information about the controlled hosts and also serves as entry point to the grid. As illustrated in Figure 1, the µgrid middleware is packaged as three elements encompassing all services offered:

- A **host daemon** running on each grid computing host that manages the local CPU, disk, and memory resources. It is implemented as a multiprocesses daemon forking a new process for handling each task assigned. It offers the basic services for job execution, data storage and data retrieval on a farm.

- A **farm daemon** running on each farm that manages a pool of hosts. It is implemented as a multithreaded process performing lightweight tasks such as user commands assignment to computing hosts.
- A **user interface** that provides communication facilities with farm daemons and access to the grid resources.

As discussed below, a grid daemon component should be added to enhance the scalability in the µgrid architecture. However it has not yet been implemented.

The user interface consists of a C++ API. A single class enables the communication with the grid and the access to all implemented functionalities. A command line interface has also been implemented above this API that offers access to all the functionalities through four UNIX-like commands (`ucp`, `urm`, and `uls` for file management similarly to UNIX `cp`, `rm`, and `ls` commands respectively plus `usubmit` for starting programs execution on the grid).

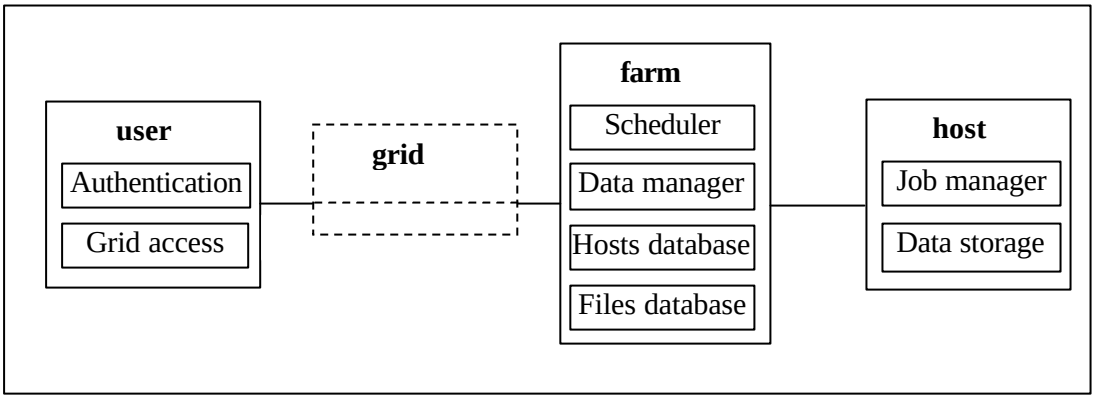

Figure 1. µgrid components and services.

Although logically separated, the three µgrid components may be executed as different processes on a single host. The communications between these processes are performed using secured sockets. Therefore, a set of hosts interconnected via an IP network can also be used to run these elements separately.

The µgrid middleware offers the following services:

- **User authentication** through X509 certificates. The authorization policy is described in the section 2.3.
- **Data registration and replication**. The middleware offers the virtual view of a single file system though data are actually distributed over multiple hosts. Files on the hosts need to be registered at the farm to be accessible from the grid middleware. Furthermore data can be transparently replicated by the middleware for efficiency reasons.
- **Job execution**. Computing tasks are executed on the grid hosts as independent processes. Each job is a call to a binary command possibly including command line arguments such as registered grid files.

2.2. System workflow

A minimal µgrid is started by running a farm daemon first, followed by a single host daemon. Both the farm manager and the host manager make use of a service certificate to authenticate to each other. Certificates are delivered by a certification authority that can be set up using the "`openssl req`" and "`openssl ca`" sample commands offered in the openSSL distribution.

The farm daemon role is to control a computing farm composed of one or more hosts. It holds a database of host capacities, grid files, and a queue of scheduled jobs. MySQL is used as database backend. When started, the farm daemon connects to the database backend. If it cannot find the µgrid database, it considers that it is executed for the first time and sets up the database and creates empty tables. In the other case, it finds in the database, the list of grid files that have been registered during previous executions and the hosts where files are physically instantiated.

The host manager role is to manage the resources available on a host. When started, it collects data about the host CPU power, its available memory and the available disk space.

It connects to the farm manager indicated on command line or in a configuration file to which it sends the host information. The host manager encompasses both a data storage/retrieval service and a job execution service.

To make use of the system, a user has to know a farm manager to which its requests can be directed. For convenience, the latest farm managers addressed are cached in the user home directory. Through the user interface, a user may require file creation, replication, or destruction, and jobs execution. These requests are sent to a farm manager which is responsible for locating the proper host able to handle the user request. To avoid unnecessary network load, the farm manager does not interfere any longer between the user interface and the target host, it only provides the user interface with the knowledge of the target host and then let the user interface establish a direct connection with the host for completion of the task. The system is fault tolerant in the sense that if the farm manager becomes unreachable (*e.g.* due to a network failure or the process being killed), the user interface parses the list of cached farm managers for completing the request. Similarly, if a worker node does not respond, it is declared "down" and removed from the farm manager list of known living hosts until it restarts and registers again.

The following sections detail the services offered by the µgrid middleware.

2.3. Authentication

User authentication in a wide scale system such as a grid cannot be achieved by user accounts creation on all nodes where the user may need to access for scalability reasons and since the target hosts are not known in advance. Instead, in µgrid as in many grid middlewares, the user is authenticated through a personal certificate. We use X509 certificates as (i) they are the *de facto* standard and (ii) tools to produce and manage them (such as the OpenSSL package) exist.

User certificates are delivered and signed by a certificate authority for a limited period of time. Classic cryptographic techniques are involved to ensure the validity of the certificates produced. An X509 certificate holds the user name, group and affiliation, thus enabling authorization policies on a per user and a per institute basis. In the µgrid system, the user certificate is used each time the user interface sends a request to the farm manager that will only accept the connection coming with a valid and signed certificate. The user passphrase is made mandatory (it is normally optional in OpenSSL) when using the certificate's private key, thus offering an extra level of protection.

All communication between the user interface and the farm or host daemons are encrypted using OpenSSL thus preventing any third party spying the network to capture information such as the user identity, the files he or she owns, or the kind of processes he or she is requesting.

2.4. Data management

For maximal transparency and ease of use, µgrid allows a user to create, modify, and delete files “on the grid” without worrying of the files location and the disk space limitation of each host. The user can register and browse grid files into a virtual file system that appears as a classical file system tree but spawns over different disks. The virtual file system tree is hold in the farm manager database. At the beginning none of the files available on the grid hosts are known to the µgrid middleware nor accessible to the user. Files have to be registered first by explicit request of the user (or implicit request such as may happen during job execution). A file is registered in the virtual file system by copying the file to a free space (a physical instance of the file is created on a controlled host) and by binding it to its full path name in the virtual tree and its target host (we will refer to the *grid file name* to designate the name of the file in the virtual tree). The user may browse the virtual tree by changing its current directory and listing files available in each directory similarly to what is done in any file system. To enable per-user file classification, the µgrid middleware defines a virtual home directory for each user whose name is based on the information read from the user certificate (thus, uniqueness is ensured provided that the CA does not deliver different certificats with the same names and affiliations).

The current µgrid middleware does not make any access control on the files, although basic control based on the user name, group and affiliation read from its certificate could be easily implemented. (The subject string would be stored in the farm manager table on file registration allowing the system to control file access at each user level). Even though, this

access control mechanism would be limited as only the owner of a file could access it. The integration with Sygn (see section 3) enables finer access control and rights delegation.

In addition, the grid data manager supports replication: to a grid file name may be associated several identical physical instances of a file recorded on different hosts. Replication may be ordered explicitly by the user but usually happens transparently for the system needs as described in the section on jobs management. These multiple instances of a file are kept on the nodes for caching in case of subsequent use, unless disk space is lacking. This replication of files causes an obvious problem of coherence that is not handled in the current implementation: the user is responsible for creating a new file in case of modification.

Since we do not make any assumption on each host file system, each host is supposed to handle its own storage resources, not necessarily visible from the other hosts. At creation time, a grid node declares its available space to the farm manager and will send frequent updates of this value. The user always refer to a file through its grid name without needing to know its physical location.

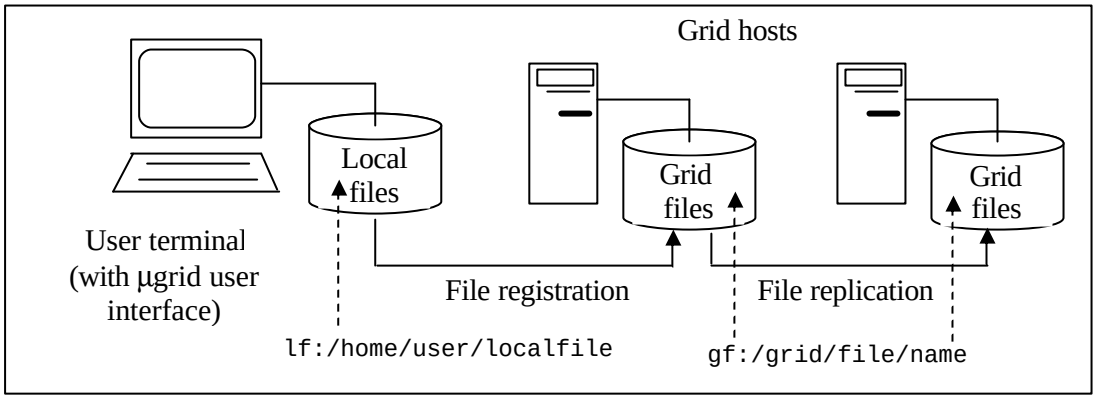

Figure 2. local and grid files

Figure 2 illustrates the difference between local files and grid files created during file registration or file replication. A local file is available on the user disk. On request, the file is registered onto the grid and is assigned a grid file name that uniquely identifies it. The file may later on be replicated in several instances, all sharing the same grid file name.

2.5. Job management

The µgrid middleware is a batch oriented system: the user interface submits tasks in the form of command lines and the farm manager is responsible for finding the adequate resources to fulfil the request. Job management is handled on a pull model: each host declares when it is free to the farm manager and the farm manager assigns computing tasks among the free hosts. Each time the farm manager is notified of a free host, it checks in its jobs queue for the oldest job that can be executed on this resource. Once assigned to an host, a job command line is executed on the target as an independent process forked by the host daemon.

The user may specify minimal memory and disk requirements for its job to ensure that the execution will happen on an host able to process it. In addition, a job may access to grid files (input data) or register the result of its processing into a grid file (output data). The farm manager is responsible for finding a suitable host to handle the computation. It will therefore send the jobs on the host holding the required input data or it will order replication of the required input data on a selected target before ordering the job execution there. Replication is performed directly between the grid nodes owning and receiving a replica to guarantee maximal performance.

Each time a computing request is queued by the farm manager, the scheduling algorithm establishes a list of suitable target hosts prioritised according to:

1. Their readiness.
2. The amount of data that has to be transferred to execute the job on this target.
3. Their processing power.

Later, it will be possible to introduce other parameters such as network bandwidth that is affecting the data transfer time.

To ensure transparency from the user point of view, the job submitter of the farm manager substitutes grid file names found in the job command line by the physical file names that will be found on the target host. Thus, the user can write a command line containing grid file name and open files in his or her code considering that the file names transmitted will in fact be local files. Similarly, the user can specify output files to be registered onto the grid at the end of the processing (by prefixing the file names by 'of:'). The output files will be preserved and registered into the farm manager files database once the job is completed. Note that using this mechanism, the binary file to be executed might be designated through a grid file name that will be substituted. Thus, the user may register binaries in the grid file system and order execution of commands making use of these binaries. As any input file, the binaries will be transported on the target host prior to execution.

The µgrid middleware does not deal with heterogeneous platforms today: all hosts are considered to be binary compatible which solves the problem of registering several binaries corresponding to a single application for several targets, but it is clearly restrictive for a full scale grid.

2.6. Perspectives

In the current implementation, µgrid is not a scalable system yet although its architecture has been designed to ease scale extension. The farm managers clearly represent a bottleneck in the system that cannot manage an unlimited number of hosts. The idea is that a farm manager should only handle local resources (for which connections are fast and reliable) and that farms will become interconnected by a higher level *grid manager* process. Thus, farms would publish their statistics (number of workers, load, available data...) to their grid manager. When a request cannot be handled locally because of resources shortage (no host available or data not accessible in the local cluster), the farm manager could delegate the request to their grid manager which can take decision on optimal resources to use. Grid managers should be organised in a communicating graph of processes similarly to what is done in DIET [12] to ensure scalability. P2P technologies offer a promising background to ensure communications between all grid agents without requiring a N-to-N connectivity graph.

3. The Sygn access control server

As we have seen in the previous section, µgrid provides secured authentication of users, but access control to files is very limited. We have stated in the introduction the need of finer access control mechanisms in the medical area so that each user involved in healthcare can control the accessibility of the data he or she administrates.

Sygn is an access control server developed at LIRIS. It implements user side permission storage. Sygn uses XML encoded certificates to specify resource access permissions. These certificates are digitally signed to detect unauthorised modifications.

Two kinds of users exist in the Sygn architecture:

- Users acting as resource administrators: these users want to control the use of their resources, and to grant permission to anybody. They use the *Sygn admin client* (SAC) for this purpose.
- Users acting as resource consumers: these users want to access their permitted resources. They use the *Sygn user client* (SUC) to store owned permissions and retrieve resources according to these permissions.

As a particular user may act both as a resource administrator and a resource consumer, only one client has been developed, acting as one or the other depending of the context.

In Figure 3, a resource administrator stores some resource on a (possibly remote) *resource server*. He uses his SAC client to contact the resource server and register himself as *source of authority* (SOA) for this resource in the *Sygn server* metadata-base of the resource server (step 1). For each resource, the database stores only its SOA's identifier.

When he wants to grant access to this resource to another user (step 2) the resource administrator issues permission certificate that allows access to this resource. Note that this process involves only himself and the concerned user(s), and can be done offline.

When receiving the permissions the SUC client allows a user to store and retrieve these permissions as needed. To access the resource, the user contacts the corresponding resource server (we assume that the localisation of the correct server is realised by the grid

middleware) and submits the request along with the needed permission certificates as shown in step 3.

The resource server needs two different Sygn modules. The *Sygn server* that takes an access control decision based on the permission certificates provided by the user and a *Sygn integration module*, responsible for the interactions with the grid middleware, which is described in detail in section **4**.

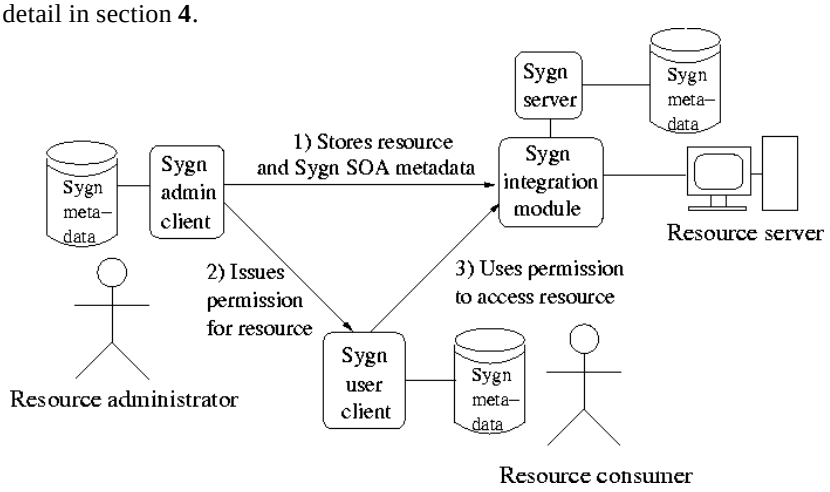

Figure 3. Components of Sygn and their interactions

4. Interoperating Sygn services and µgrid services for data management

4.1. Interoperation of Sygn and a grid middleware

The Sygn access control server is designed to interoperate with a grid middleware through an integration module. A request issued by a grid user contains the identifier of this user, the command to be executed and the accompanying Sygn permissions. When it arrives at the resource server, this module has the following functions:

- Verify if the request issuer corresponds to the owner of the submitted Sygn permissions.
- Verify if the submitted Sygn permissions correspond to the submitted request.
- Query the Sygn server to verify that the permissions are valid and allow or deny the submitted request.

The rationale behind the first point is that authentication is not part of Sygn's services. If we did not check for correspondence between permission holder and grid user submitting a request, any user could copy the permissions of an authorised user to gain access to resources.

Secondly, checking if a Sygn permission applies to a grid request keeps the format of grid requests independent from the format of Sygn permissions. The correspondence checks are necessary, otherwise a user could pass a grid request for some resources using valid Sygn permissions for other resources.

Finally the Sygn permissions are submitted to the Sygn server, which checks their validity and responds with either *granted*, *denied* or *failed* messages. *Granted* means the permissions are valid, *denied* that they are invalid and *failed* means that the validity could not be verified due to a failure in the Sygn servers components (*e.g.* if the database is unavailable). The integration module interprets the Sygn server's response, launches the grid request if it was positive or returns an error message explaining why the request was rejected.

4.2. Interoperate Sygn and µgrid

We have created a Sygn integration module that empowers µgrid with the Sygn service. The actual integration is limited to file access control on µgrid using Sygn

permissions. It enables access control for the command line functions as well as for the µgrid C++ API allowing to copy, upload, download and delete files in µgrid. If the Sygn use is made mandatory (an option at µgrid start-up), it becomes impossible for µgrid users to access files without the required permissions.

One of the main issue for interoperating Sygn and µgrid was the place of Sygn in the µgrid middleware. Two possibilities arise from the µgrid architecture :

- at the farm level, since it represents the entry point of µgrid, thus leading to a unique Sygn server for the grid –the whole set of physical servers is virtualised as only one;
- alternatively, on a set of distributed Sygn servers, maybe on each resource server (but two µgrid servers can use the same Sygn server, a 1-to-1 mapping is not necessary).

The first solution obviously leads to a bottleneck for the single Sygn server (all the requests occur there and the metadata base might be very large), as well as a single point of failure. We have thus chosen the second solution.

In this solution, the workload is balanced between the Sygn servers. Two problems appear within this solution: the consistency of the view of the farm with the actual servers, and the consistency of the permissions due to file replication.

The first problem concerns the consistency of the view on the files present at a given site between the farm manager and that actual site: indeed, a user command might be seen as acceptable by the farm manager but denied locally by the Sygn server. For instance, if a user wants to delete a grid file, the farm manager might update its database on the localisation of this file, even if the local access control made by Sygn denies this action, leading to a consistency failure. Thus, once approved but the Sygn server, the decision (request done, request failure) is returned to the farm manager so that it can keep a consistent view of the grid files.

A second problem arises due to the replication of files. µgrid allows for a transparent replication of files among the resource servers. If a file is replicated without its permissions, it will be either access free to everyone (if that server does not hold a Sygn server, which is possible if Sygn is made optional in µgrid) or non accessible to everyone (the local Sygn server does not know how to resolve access permissions for this file, because it is not referred in its local metadata base). The free access is eliminated by denying replication of controlled files on servers where Sygn is not running. The non-accessibility is tackled by replicating the file SOA's identifier at the destination, as this identifier is the only mandatory metadata the Sygn server needs for its access decisions.

4.3. Examples of requests in µgrid, controlled by Sygn

We start by presenting command line examples of the possible operations involving grid files and explaining the required permissions for these commands to successfully execute.

Registering a local file to the grid requires to have write permission on the grid-file if it already exists (actually the user is overwriting the grid file with his local file). If the grid-file does not exist the user copying the file is registered as the SOA for that grid-file. Copying a grid-file to another grid-file requires read permissions on the source file and write permissions on the target file (if it already exists). Retrieving a grid-file to a local file requires only read rights on the grid-file. The copy command (used for registration, copy, or retrieval depending on the source and destination file kind) has the following structure in the command line interface:

```
ucp lf:local-file gf:grid-file --sygn
```

where the '`lf:`' prefix designates a local file while the '`gf:`' prefix designates a grid file. The `--sygn` option activates the SUC, which automatically retrieves and appends the correct permissions to this request (if the user holds these permissions).

Deleting a grid file requires write permissions and is done through the following command:

```
urm gf:target-file –-sygn
```

To use permissions in µgrid with the `--sygn` option, the user must import these in the SUC once and they will be available subsequently.

The µgrid C++ API permits registration (local file to grid-file), retrieval (grid-file to local file), copying (grid-file to grid-file) and deletion of grid files. As the command line interface uses the API to execute its functions, the same access control requirements apply as above.

The operations are provided as methods of an interface class that can be included and instantiated by user programs.

5. Discussion on security issues

The architecture presented in this paper can be compared globally with what can be found in Globus [13] using GSI [14] for authentication and CAS [5] for fine grain access control. It is far less developed in terms of functionalities for the grid usage and tools, but it is relevant to discuss the comparison from security point of view. The advantages of our model are that it does not rely on the grid architecture to delegate, obtain and manage permissions, nor does it rely on a remote server to store and hold these permissions. This minimizes the impact of a successful attack on a component of Sygn. The SUC stores no security-critical data, since all permissions it holds require prior authentication of their owner to be used. The SAC is not a good target either, since access to a SOA's private key is required to forge valid permissions. Therefore both components only require an adequate protection of the users private keys (owner private key for the SUC, SOA's private key for the SAC) which is a common requirement in distributed systems.

Sygn servers are deployed at resource server level, thus they represent less valuable targets for attackers than a permission repository (for example a CAS server). A successful attack on a Sygn server only disables the access control on the attached resource server.

Unlike Akenti [7] or Permis [8], Sygn does not involve a third party permission storage. The SUC permits to store all the permissions obtained by one user for his/her access on remote resource servers. Users are the only providers of their permissions to the Sygn servers.

Moreover, to completely fulfil the requirements of privacy protection, the data managed should also be encrypted. Indeed, the physical files on remote hosts must be protected against possible attacker on the operating system underlying the grid middleware. The Sygn architecture is fully compatible with another work of our team on distributed, encrypted storage making the disclosure of private data almost impossible in a real system [15]. Even if someone gets direct access to the data, he would not be able to decrypt its content.

As value added features of Sygn (not discussed in this paper), we can manage groups of users and files, non-repudiation of access logging, black lists and permission revocation lists.

The presented architecture of the combination of µgrid and Sygn is developed under GPL license. It is freely available and can be downloaded from `http://liris.cnrs.fr/~lseitz/software`.

6. Conclusions and perspectives

We have presented in this paper two interesting tools for grid users and developers. The first one, µgrid, is a light weight middleware for grids, enabling basic commands on a set of hosts, for computing and data management purpose. The second one, the Sygn architecture, proposes an alternative to standard access control mechanism that can be found in today grids, allowing for a fine grain access control in a fully decentralised way. Sygn responds to the complex needs for authorisation arising in grid enabled medical applications. Finally, we have exhibited the interoperability of the two softwares and the feasibility to integrate the latter in a basic grid middleware.

The future of this project is twofold. Concerning the middleware, we plan to extend the µgrid prototype in order to handle multiple farm managers, as explained in section 2.6. On the access control side, we want to finally integrate the Sygn architecture in a standard grid service middleware such an OGSA or WS-RF service.

References

[1] Montagnat J., Duque H., Pierson J.M., Breton V., Brunie L., and Magnin I.E., “Medical Image Content-Based Queries using the Grid”. Proceedings of the first European HealthGrid conference, pp 142-151, Lyon, France, January 2003.

[2] Montagnat J., Breton V., Magnin I.E., “Using grid technologies to face medical image analysis challenges”, Biogrid'03, proceedings of the IEEE CCGrid03, pp 588-593, May 2003, Tokyo, Japan.

[3] Montagnat J., Bellet F., Benoit-Cattin H., Breton V., Brunie L., Duque H., Legré Y., Magnin I.E., Maigne L., Miguet S., Pierson J.-M., Seitz L., Tweed T.. "Medical images simulation, storage, and processing on the European DataGrid testbed". To appear in Journal of Grid Computin g, 2005.
[4] Claerhout B. and De Moor G., "From GRID to HealthGRID: introducing Privacy Protection", Proceedings of the first European HealthGrid conference, pp 152-162, Lyon, France, January 2003.
[5] Pearlman L., Welch V., Foster I., Kesselman C. and Tuecke S., "A Community Authorization Service for Group Collaboration". Proceedings of the 2002 IEEE Workshop on Policies for Distributed Systems and Networks, 2002.
[6] Alfieri R., Cecchini R., Ciaschini V., dell'Agnello L., Frohner Á., Gianoli A., Lörentey K. and Spataro F., "VOMS, an Authorization System for Virtual Organizations". Proceedings of the 1st European Across Grids Conference, 2003.
[7] Thompson M., Mudumbai S., Essiari A. and Chin W., "Authorization Policy in a PKI Environment", Proceedings of the 1st Annual NIST workshop on PKI, 2002 .
[8] Chadwick D. and Otenko A., "The PERMIS X.509 role based privilege management infrastructure", Proceedings of the 7th ACM Symposium on Access Control Models and Technologies, pp 135-140, 2002.
[9] J. Montagnat, V. Breton, I. E., "Medical image databases content-based queries partitioning on a grid". HealthGrid'04, January, 2004, Clermont-Ferrand, France.
[10] OpenSSL. Secured Socket Layer. http://www.openssl.org/.
[11] MySQL. SQL database. http://www.mysql.org/.
[12] Caron E., Desprez F., Lombard F., Nicod J.-M., Quinson M., and Suter F., "A Scalable Approach to Network Enabled Servers". Proceedings of the 8th International EuroPar Conference, volume 2400 of LNCS, Paderborn, Germany, pp 907-910, August 2002. Springer-Verlag.
[13] Globus Project, http://www.globus.org
[14] Globus Project, "Globus Security Infrastructure-GSI", http://www.globus.org/security/
[15] Seitz L., Pierson J.M., and Brunie L., "Key management for encrypted data storage in distributed systems". Second International IEEE Security in Storage Workshop (SISW). Washington DC, USA, October 2003.
[16] MEDIGRID, French ministry for Research ACI-GRID project. http://www.creatis.insa-lyon.fr/MEDIGRID/.