



IST-2001-32133

GridLab - A Grid Application Toolkit and Testbed

Documented first release of the security infrastructure and APIs

Author(s):	Marcin Adamski, Michal Chmielewski, Sergiusz Fonrobert, Jarek Nabrzyski, Tomasz Nowocień, Tomasz Ostwald
Document Filename:	
Work package:	WP6 Security
Partner(s):	PSNC, MPG, MU, ISUFI
Lead Partner:	Poznan Supercomputing and Networking Center
Config ID:	GridLab-6-D6.4-0004-0.9
Document classification:	INTERNAL

Abstract: In this document, a description of implementation and tests of the first release of the Grid Authorization Service developed as a part of WP6 of the GridLab project is presented. The document contains general information about the first release, a further development plan, a description of upcoming tests and a general plan of integration with other Workpackages



Last amendment date: 2004/01/16 & time: 10:30:00



Contents

1	Introduction	2
1.1	Purpose of Document	2
1.2	Structure of Document	2
1.3	Status of Document	2
2	Goals and requirements for the first release of the GAS	3
3	The general design of the Grid Authorization Service	4
3.1	The GAS server	4
3.1.1	Program and config file	4
3.1.2	The core of GAS server	5
3.1.3	Data structures	5
3.1.4	Database	7
3.1.5	API	10
3.1.6	GAS management	11
3.2	GAS clients applications	12
3.3	The Security Portlet for GridSphere	14
3.4	Communication between components	14
4	Integrations	16
4.1	Integrations with GRMS	16
4.2	Integrations with Monitoring	16
4.3	Integrations with Mobile	16
4.4	Integrations with Portal	16
5	Enable Scenarios for first release of the GAS	18
6	The near future of the GAS	19
6.1	Extending scenarios functionality of the GAS	19
6.2	Extending method of generating security policy	19
6.3	GUI gas_admin_client	19
6.4	Possible integrations with other security solutions	19
7	Summary	20

1 Introduction

1.1 Purpose of Document

This document contains information about the first release of the Grid Authorization Service. Specifically it is focused on issues like building the GAS service, state of integration with other workpackages and future plans for the GAS service.

1.2 Structure of Document

The document is divided into 7 general sections.

In section 1, a general document structure and its goals are presented.

In section 2, the main goals and requirements for the GAS are shown.

Section 3 describes the general design of the Grid Authorization Service. Internal, external components and communication methods between them are described here.

In Section 4, the current status of integration with external services is presented. Some plans for possible future integrations are also described.

Section 5 is dedicated to a description of possible scenarios, which can be used for the first release.

Section 6 discusses possible extensions to the GAS after the first release.

The summary and final notes are given in section 7.

1.3 Status of Document

This document is a working draft and refers to the current state of the project. As a working draft, this specification may be updated, replaced, or made obsolete at any time. It is distributed for discussion purposes only, and should not be used as a reference.

2 Goals and requirements for the first release of the GAS

The main requirements for Security Workpackage 6 in the GridLab project have been presented in the first deliverable, *WP6 Security: Initial Requirements* [1]. In the following deliverable entitled *Technical Specification for Authorization Service* [2] the architecture and building of the GAS is described. The general overview of the first prototype was presented in *Implementation and Test Plan* [3]. On the basis of these three deliverables, the first release of the Grid Authorization Service is developed.

Each authorization service has to provide functionality, which is shown below:

- getting a single authorization decision for the user,
- getting part of security policy for the user,
- managing the authorization service.

In the first release of the GAS all these tasks are supported in a few possible ways. One of the main features of the first release is to support two authorization security models: the RAD (Resource Access Decision) [6] and the RBAC (Role Based Access Control) [5]. The first release of the GAS will try to answer which model is better for the specific application of the GridLab project. The first release of the GAS is developed in accordance with *Technical Specification for Authorization Service* [2] where the main goals of the Grid Authorization Service are presented:

- The main requirement is flexibility of the Grid Authorization Service. The GAS is about to provide a universal way of defining security policy for the whole Grid, independent of technologies used at lower levels;
- It should be able to implement most security models for Grids and use many different scenarios at the same time;
- It should support many different security technologies;
- There has to be secure and stable implementation, as the GAS is considered as a trusted component of the security model.

Based on such requirements and goals, with the first release of the GAS it is possible to:

- create a structure for security policy based on services requirements with time or capability limitation;
- add security policy to the GAS server by a special admin client or web admin client and store this policy inside a database;
- generate the authorization decision for a user or a service request;
- generate part of security policy for a user or a service (based on rules defined by the service);
- generate full security policy for a user or a service (in this mode the GAS is in the CAS scenario [2]) and is compatible with the CAS (Community Authorization Service); the generated policy can be used with the CAS enabled components (like gsift) [4];
- presents security policy in a text form (in `gas_admin_client` and in protlet in Gridsphere).

3 The general design of the Grid Authorization Service

3.1 The GAS server

The GAS server is the main and the most important component of the GAS service. This program is placed at the central point of the system and all other components are connected to this item. Information on how to run this server and what startup parameters look like is given in subsection 3.1.1. The next subsection contains information about the core component of the GAS server. In next two subsections 3.1.3 and 3.1.4 the data structure and database for the GAS server are presented. Subsection 3.1.5 describes the GAS server API which can be used by external services. Issues related with administration of the GAS server is presented in subsection 3.1.6.

3.1.1 Program and config file

The GAS server was designed as a separate program, named `gas_server`. The path to the config file is the only parameter for this program. In the config file there are a few parameters available:

- `StructureFile` - a path to the file which has the structure of data for the GAS server. This path was used for the first prototype of the GAS, when the database was taken from the text file but it is still possible to work with this structure for use in case of simple applications;
- `DatabaseFile` - a path to the file which covers the security policy database. This path was used for the first prototype of the GAS similarly to the path from the "StructureFile" key but it is still possible to read data from text files;
- `Server_Certificate` - a path to the file which covers the server certificate in X.509 format;
- `Server_Private_Key` - a path to the file which covers the server private key;
- `Server_Certificate_Dir` - a path to the directory which contains all CA certificates acceptable by the GAS server;
- `LogFile` - a path to the log file where all information about events into the GAS server will be stored
- `PortGSI` - a port number for GSI connections;
- `PortSOAP` - a port number for gSOAP connections.

A sample configuration file is presented below:

```
StructureFile=/home/gas_user/test.tt
DatabaseFile=/home/gas_user/testDB.tt
Server_Certificate=/home/gas_user/.cas/usercert.pem
Server_Private_Key=/home/gas_user/.cas/userkey.pem
Server_Certificate_Dir=/home/gas_user/.cas/certificates
LogFile=/home/gas_user/.cas/gas.log
PortGSI=12351
PortSOAP=12350
```

3.1.2 The core of GAS server

The Grid Authorization Service has a modular structure. This structure was described in the deliverable entitled *Technical Specification for Authorization Service* [2]. The current status of specific modules of the first release is presented on figure 1. The color points modules which are developed or finished. All others modules will be developed in the near future.

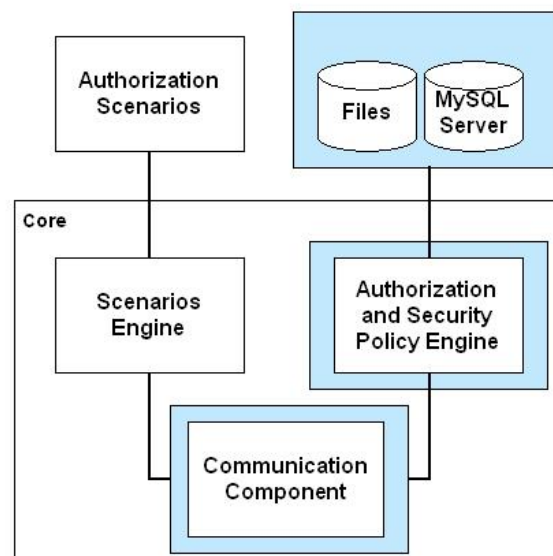


Figure 1: The first release of core

Generally, some parts of the core module like the database module and the communication module are partially finished (in the figure they are presented in color). These modules are ready to work. The process of developing the authorization module was started and this module is ready for simple usage. Functionality of this module will be significantly extended in the future and together with the scenario module which is not developed yet makes up the most complicated part of the GAS service.

3.1.3 Data structures

The building of the data structure for the GAS server results form security authorizations models which are supported by the GAS server. Generally such models are described using the following entities:

- objects - entities like servers, files, web pages, represent items security policy can be defined for;
- operations - represents an action, which can be performed on objects
- subjects - represents users or others entities which want to perform operations on objects

- roles - represents a special context in which subjects want to perform an operation on an object,

In the GAS server the structure of objects definitions is designed in the way to enable creating accurate representations of world. Generally this structure looks very complex but it was created in such way for the following reasons:

- world is complex so structures that describe world can hardly be simple;
- all items of the world have to be connected into one structure for return unambiguous authorization decision;
- managing security policy has to be unambiguous and easy.

This structure, presented in figure 2, is a multi-layered graph, which can be used in different implementations. In the GridLab environment two layers can be distinguished: physical and logical. On the physical layer the network items can be placed as: servers, files, webpages and etc. On the logical layer the items can be placed as: services, resource, queue and jobs. On each layer all objects are connected into the graph structure.

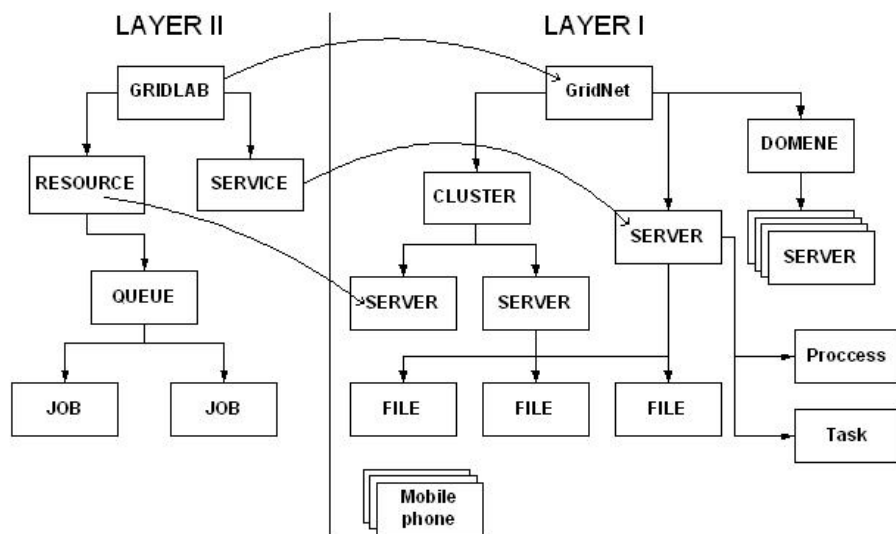


Figure 2: Data structure - a definition structure of sample objects which can be used in GridLab

Each object can be connected to its own set of operations (describing what kind of actions can be performed on an object). The structure of the operation is not complex. On figure 3 a sample structure for operations which can be connected to the object from figure 2 is presented.

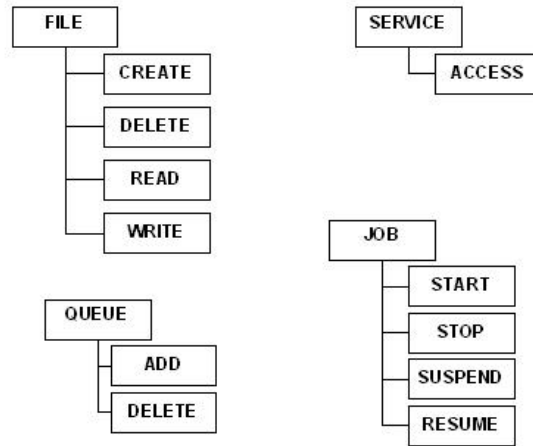


Figure 3: Data structure - sample operations for object definitions

The structure of subjects is hierarchical starting from groups at the top level and with detailed subjects at the bottom level. Groups can contain other groups and subjects. In the sample solution for the GridLab, a Virtual Organization object is on the top and below groups of users are placed (figure 4).

Relation (security policy rule for the RAD model) and permission (security policy rule for the RBAC model) structures are shown in figure 5. Relation connect object, operation (on this object) subject and array of limitations however permission connect the same items as relation plus the role the subject is in. Each relation or permission defines one security policy rule.

In the first prototype, the GAS used a simple data structure that contains only a simple array of objects. Using this data structure it was possible to test only one operation on an object - the access operation. This structure is still supported in the first release but can be used only while working with text files.

3.1.4 Database

In the first release of the GAS it is possible to store the data structure into an external SQL database server. The GAS server uses the unixODBC library to connect to the SQL server. Using the unixODBC driver is the only requirement for selecting specific database system. It is important to mention that during internal tests the GAS server was used only with the MySQL database server, thus this database server will be probably used by the GAS service for the GridLab.

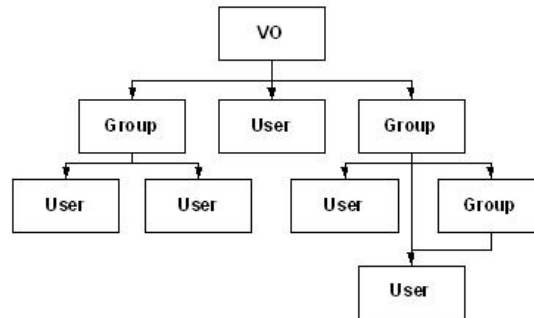


Figure 4: Data structure - a sample structure of subjects and groups

There are 26 arrays defined:

- **ObjectDef** - the table contains objects definitions which cover information like object name, layer and id;
- **SubjectDef** - the table contains subjects definitions which cover information like subject name and id;
- **RoleDef** - the table contains roles definitions which cover information like role name and id;
- **GroupDef** - the table contains groups definitions which cover information like group name and id;
- **ObjectAttrDef** - the table contains information about attributes definitions for objects;
- **SubjectAttrDef** - the table contains information about attributes definitions for subjects;
- **RoleAttrDef** - the table contains information about attributes definitions for roles;
- **GroupAttrDef** - the table contains information about attributes definitions for groups;
- **OperationDef** - the table contains information about operations definitions for objects;
- **ConnectionObjToObjDef** - the table of connections between the **ObjectDef**;

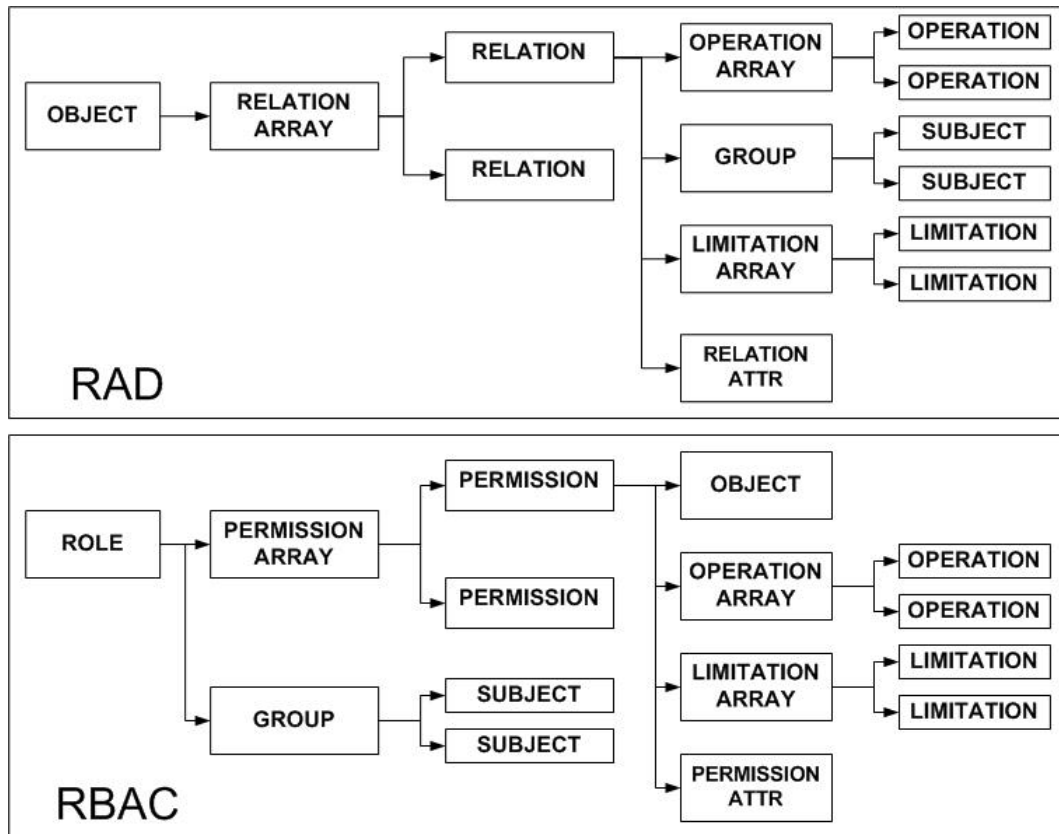


Figure 5: Data structure - structure of policy rule for the RAD and the RBAC models

- **Object** - the table contains information about objects based on the `ObjectDef` definition;
- **Subject** - the table contains information about subjects based on the `SubjectDef` definition;
- **Group** - the table contains information about groups based on the `GroupDef` definition;
- **Relation** - the table contains information about relations;
- **ConnectionObjToChildObj** - the table of connections between objects (from the same layer);
- **ConnectionObjToLayerDownObj** - the table of connections between objects but from others layers;
- **ObjectAttributes** - the table contains information about objects attributes based on the `ObjectAttrDef` definitions;
- **ObjectRelations** - the table of connections from relations to objects;
- **ConnectionToRelationToOperation** - the table of connections from relations to operations;
- **GroupSubjects** - the table of memberships between groups and subjects;

- `ConnectionToGroupToChildGroup` - the table of memberships between groups and groups;
- `RoleSubjects` - the table of connections from subjects to roles;
- `RolePermissions` - the table of connections from permissions to roles;
- `Role` - the table contains information about roles based on `RoleDef` definition;
- `Permission` the table contains information about permissions;
- `PermissionOperations` the table of connections from operations to permissions;

The GAS server can use multiply databases. It is possible to load content of database to the data structures using a command from the `gas_client_admin` program. The `database` module will be extended in the future to support all edit operations on databases and optimized, especially for memory usage. Currently, the database is the only place where data integrity can be tested.

3.1.5 API

The gSAOP interface for the GAS server is presented below. This interface can be used by external services to access the GAS server functionality.

- `ns__getServiceDescription` - the function for getting standard service description, the function returns string;
- `ns__sendCommandAdminString` - the function for sending command string to server, this function returns a string;
- `ns__getMonitoringStaticRules` - the function for generating rules for monitoring service returns a string;
- `ns__getResourcesList` - the function for getting resources for user (input parameter) for the GRMS service returns a string;
- `ns__getAuthorizationDecision` - the function return authorization decision for the input user and object returns 0/1;
- `ns__getCASCompatiblePolicy` - the function return the policy string into the CAS compatible form;
- `ns__getRADAuthorizationDecision` - the function returns the authorization decision into the RAD model for the given subject (user name) object (object name) and operation (operation name), returns 0/1;
- `ns__getRBACAuthorizationDecision` - the function returns the authorization decision into the RBAC model for the given subject (user name) object (object name) and operation (operation name) and role (role name), returns 0/1;
- `ns__getSignedMonitoringStaticRules` - the same as for function `ns__getMonitoringStaticRules` but returns string signed by the GAS server;
- `ns__getSignedResourcesList` - the same as for function `ns__getResourcesList` but returns string signed by the GAS server;

- `ns__getSignedAuthorizationDecision` - the same as for function `ns__getAuthorizationDecision` but returns string signed by the GAS server;
- `ns__getSignedCASCompatiblePolicy` - the same as for function `ns__getCASCompatiblePolicy` but returns string signed by the GAS server;
- `ns__getSignedRADAuthorizationDecision` - the same as for function `ns__getRADAuthorizationDecision` but returns string signed by the GAS server;
- `ns__getSignedRBACAuthorizationDecision` - the same as for function `ns__getRBACAuthorizationDecision` but returns string signed by the GAS server;

Please note that this API refers to the first release of the GAS service. In future, this interface will be changed and extended.

3.1.6 GAS management

Currently, there are four ways of the GAS server management:

- `gas_admin_client`;
- `gas_client_admin_soap`;
- portlet inside gridsphere;
- API function `ns__sendCommandAdminString`.

With all these components it is possible to use commands listed below:

- `add_obj_def` - this command is used to add a new item definition to the security policy database. There is syntax for this command: `add_obj_def type=X name=Y [layer=L] [value_type=V] [obj_type=T]`. Where "X" has to be one of the items from set: `ObjectDef`, `SubjectDef`, `RoleDef`, `GroupDef`, `AttrDef` and `OperationDef`. "Y" is the name of the item. "L" can be used only if `type=ObjectDef` and this is the number of the layer on which object will be placed on. "V" and "T" can be used only when `type=AttrDef` and means a value type (`STRING`, `INT`, `TIME`) and an object type this attribute belongs to (from set of: `ObjectDef`, `SubjectDef`, `RoleDef`, `GroupDef`).
- `add_obj` - this command is used to add new item to the security policy database. There is syntax for this command: `add_obj type=X name=Y def=Z [obj_name=V] [obj_type=T]`. Where "X" has to be one of the items from set: `Object`, `Subject`, `Role`, `Group`, `Relation`, `Permission` and `Attribute`. "Y" is the name of the item. "Z" is the name of the object definition which was added using command `add_obj_def`. "V" and "T" can be used only with `type=Attribute` and mean the object name and the object type this attribute will be added to. The parameter `obj_type` can be one of the items from set: `Object`, `Subject`, `Role`, `Group`, `Relation` and `Permission`.
- `list_obj` - this command lists all objects which meet the input condition. There is syntax for this command: `list_obj X`. Where "X" has to be one of the items from set: `ObjectDef`, `SubjectDef`, `RoleDef`, `GroupDef`, `AttrDef`, `OperationDef`, `Object`, `Subject`, `Role`, `Group`, `Relation`, `Permission` and `Attribute`.

- `list_obj_attrs` - this command lists all objects which meet the input condition. There is syntax for this command: `list_obj_attrs type=X name=Y`. Where "X" has to be one of the items from set: `ObjectDef`, `SubjectDef`, `RoleDef`, `GroupDef`, `AttrDef`, `OperationDef`, `Object`, `Subject`, `Role`, `Group`, `Relation`, `Permission` and `Attribute`. "Y" is the name of the item.
- `list_obj_operations` - this command lists all objects which meet the input condition. There is syntax for this command: `list_obj_operations Y`. Where "Y" is the name of the object (operation can be only added to the `Object` item).
- `info_obj` - the command for getting full information about a specific item. There is syntax for this command: `info_obj type=X name=Y [parent_type=T] [parent_name=V]`. Where "X" have to be one of the items from set: `ObjectDef`, `SubjectDef`, `RoleDef`, `GroupDef`, `AttrDef`, `OperationDef`, `Object`, `Subject`, `Role`, `Group`, `Relation`, `Permission` and `Attribute`. "V" and "T" have to be specified when `type=AttrDef` and means the object name and object type this attribute belongs to. The object type can be one of the items from set: `ObjectDef`, `SubjectDef`, `RoleDef`, `GroupDef`, `OperationDef`, `Object`, `Subject`, `Role`, `Group`, `Relation` and `Permission`.
- `close` - the command which closes the connection to the GAS server. There is syntax for this command: `close`
- `stop_server` - stops the GAS server. There is syntax for this command: `stop_server`
- `save_database` - saves data structure to the database. There is syntax for this command: `save_database X`. Where "X" means the name of the data link to the database from the `odbc.ini` file.
- `load_database` - loads the database to data structure. There is syntax for this command: `load_database X`. Where "X" means the name of the data link to the database from the `odbc.ini` file.
- `query` - the command returns the authorization decision. This command can only be used to test security policy stored on the GAS sever. There is syntax for this command: `query type=X ObjectName=X1 SubjectName=X2 OperationName=X3 [RoleName=X4]`. Where "X" has to be one of the items from set: `RAD_AuthorizationDecision`, `RBAC_AuthorizationDecision`. "X1" is the name of object, "X2" is the name of the subject (user), "X3" is the name of the operation which the user wants to perform on the object and "X4" has to be given in the RBAC model and means the role of the user.

3.2 GAS clients applications

From the technical point of view the first release of the GAS will be composed of seven components:

- `gas_client_admin` - this is a command line tool for adding new objects, subjects, attributes or other items to security policy stored on the `gas_server` (see 3.1.6). This program communicates with the `gas_server` by a protocol compatible with GSI. The command line string for this program is the following: `gas_admin_client [-s server] [-p port] [file_with_commands]`. The "server" parameter is the host domain name on which the `gas_server` runs at a port given by the "port" parameter. The last parameter

is the path to the script file which contains a set of commands for the `gas_server`. All parameters are optional but when "server" parameter is not given, the "localhost" name is used instead. The default port for the connection by the GSI protocol with the `gas_server` is 12350.

- `gas_client_admin_soap` - this is a command line tool. This program is similar to the `gas_client_admin` with exception of the protocol used for communication. The `gas_client_admin_soap` uses the gSOAP protocol for communication with the `gas_server`. The command line string for this program is as follows: `gas_client_admin_soap [-s server] [-p port] [file_with_commands]`. The "server" parameter is the host domain name on which the `gas_server` runs at a port given by the "port" parameter. The last parameter is the path to the script file which contains a set of commands for the `gas_server`. All parameters are optional but when the "server" parameter is not given, the "localhost" name is used instead. The default port for the connection by the gSOAP protocol with the `gas_server` is 12351.
- `gas_client` - this is a short program which obtains full security policy for the user from the `gas_server` and stores this policy into the CAS proxy file (this program is similar to `cas-proxy-init` [4]). This program communicates with the `gas_server` by the GSI protocol. The command line string for this program is the following: `gas_client -s server -p port`. The "server" parameter is the host domain name on which the `gas_server` runs at the port given by "port" parameter.
- `gas_client_soap` - this is a short test program which shows how the GAS server interface API functions works. This program communicates with the `gas_server` by the gSOAP protocol. The command line string for this program looks like that: `gas_client -s server -p port`. The "server" parameter is the host domain name on which the `gas_server` runs at port given by the "port" parameter.
- `gas_enabled_tcp_server` - this is a sample server application which can use security policy that is provided by a client application (see `gas_enabled_tcp_client`). The command line string for this program is as follows: `gas_enabled_tcp_server [port]`, where "port" is the port number the server will be run on.
- `gas_enabled_tcp_client` - this is a sample client application which can use a cas proxy file to access the `gas_enabled_tcp_server`. The command line string for this program is the following: `gas_enabled_tcp_client [port]`, where "port" is the port number the server will be run on.
- `cas_proxy_viewer` - this is a tool for printing security policy which is included into a standard CAS proxy file or the GAS proxy file on the screen. The command line string for this program is the following: `cas_proxy_viewer [path_to_proxy_file]`, where the "path_to_proxy_file" is full path to the valid proxy file.
- `gas_enabled_module` - this is a library which can be included into the service for communication with the GAS and getting full or part of security policy or an authorization decision based on the service request.

All these components are written in the C language.

3.3 The Security Portlet for GridSphere

The Security Portlet is to be a part of the Gridportlets set. Its main service is to give an administration interface for the GAS. The Security Portlet has two modes of action: the configuration mode and the working mode. In the configuration mode it is possible to configure interaction parameters such as the GAS server and port to connect to. Each portlet instance has to be well configured before working. In the working mode it is possible first of all to get an authorization decision from the particular service (set in the configuration mode). It is also possible to send any command accepted by the server, get a full answer from the server (for managing the GAS server) and presents information about the GAS server items like: objects, subjects, operations etc. Currently a command has to be typed in the command line (edit-box) and the full answer is displayed in a separate frame (text-area). Efforts have been undertaken to simplify the security portlet interface. The first step is to change the command line interface to a more user-friendly interface, that would contain elements such as combo-boxes, edit-boxes etc. and thanks to this make the portlet easier to use. The next step will be a fully Graphic User Interface. But both the functionality and implementation method of the GUI are still in the analysis phase. There are portal functions which can be used for this aim.

3.4 Communication between components

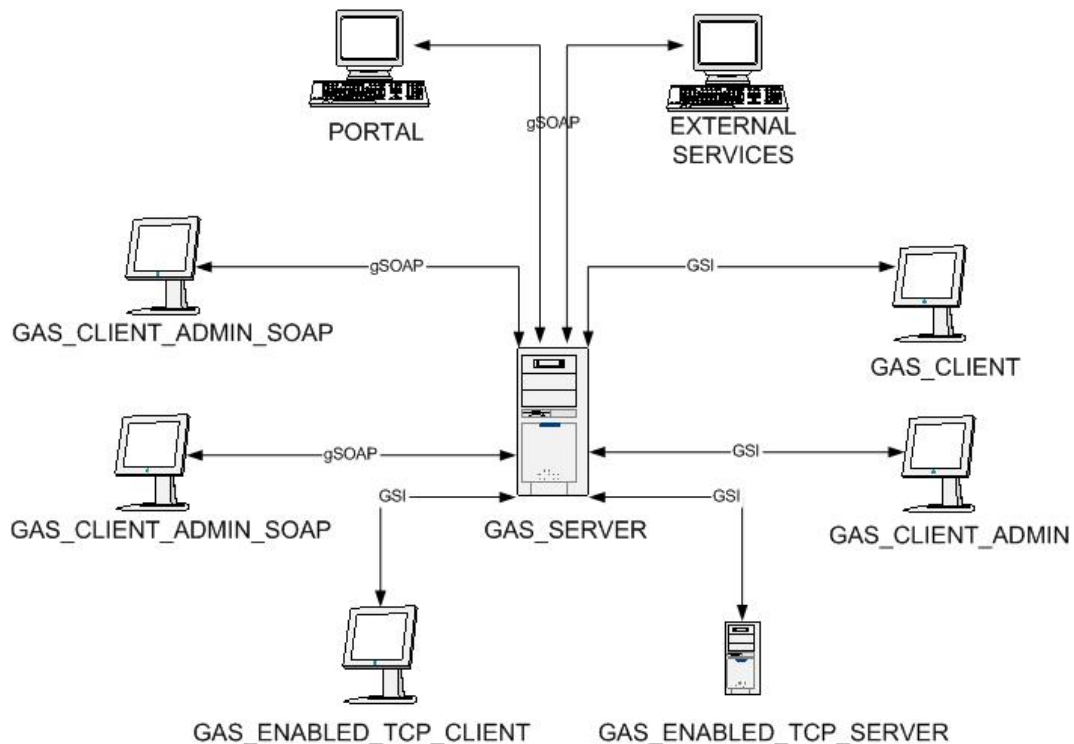


Figure 6: Components and communication

Generally, there are two ways of communication between different components of the GAS service:

- using its own protocol based on the GSI standard protocol;

- using the gSOAP protocol (gSOAP plugin for GSI);

The first protocol is used for communication between the GAS server and GAS internal components like `gas_admin_client`, `gas_client`. This protocol is faster than the second one but is not fully universal. For this reason, it will be used only internally. With the second protocol it is possible to establish communication between the GAS server and internal components, external services or portals. The second protocol is more universal and easy to use by anyone who wants to communicate with the GAS server. Components and communication are presented on figure 6.

4 Integrations

With availability of the first GAS prototype experiments aimed at integrations with different external service (provided by other workpackages) have been started. Below, a brief overview of status of integration between the GAS service and selected other GridLab services is given.

4.1 Integrations with GRMS

The integration with the GRMS is currently the most advanced. Firstly, a special API function for the GRMS was prepared for getting accessible resources for a specific user (this function was included into the first prototype of the GAS). In the next step structure of objects was created for the GRMS, which is an attempt of appropriate modeling the GRMS world inside the GAS.

Two functions can be used by the GRMS: a function for getting a list of resources, and a function for getting the authorization decision if someone can make an operation for a specific object. In next steps integration will be extended by the possibility of managing jobs (it will be possible to test what the user can do with jobs) as well as to make a warranty that the authorization decision given by the GAS is valid because of integrating the GAS with globus components on gatekeeper level.

4.2 Integrations with Monitoring

Initial steps aimed at integration with the Monitoring have been completed. A special data structure was created for the Monitoring service based on a discussion with the monitoring group. The integration with this service needs a possibility of generating part of security policy upon the user request. This functionality is under heavy tests now and will be accessible in the very near future. In figure 7 sample data structure for monitoring service is presented.

4.3 Integrations with Mobile

At this time, integration with mobile components is under heavy development. The main effort is focused on creating data structures for mobile services. There are the following mobile services which will probably be integrated with the GAS:

- Visualization Service
- Mobile Command Center
- Message Box Service

The integrations with first two services will be relatively easy, because it will require only a decision if the user has access to these services. The last integration will be more complex and a special data structure will be required for this service.

4.4 Integrations with Portal

From the point of view of the GAS service, integration with a portal group was started with the first version of the administration client for the GAS service prepared as a portlet for the Gridsphere. Another point for integration is to use the GAS for getting an authorization decision

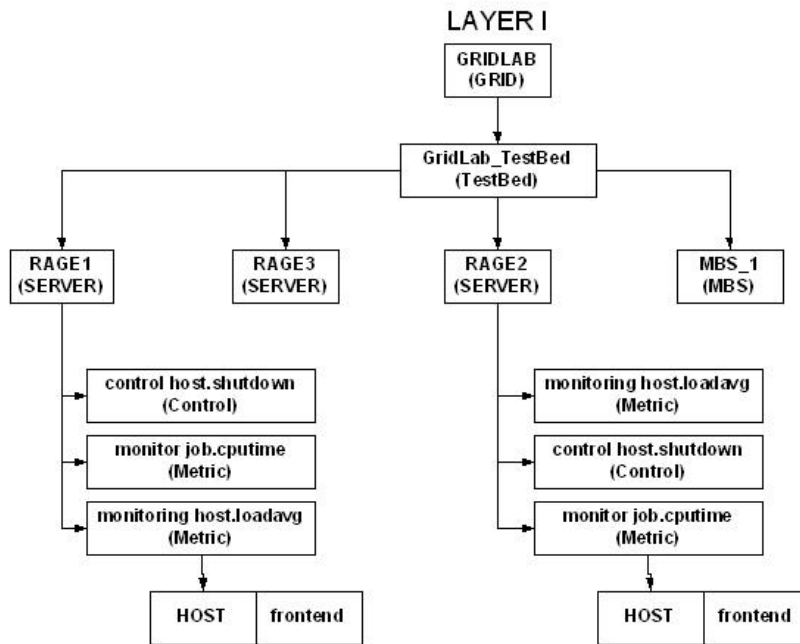


Figure 7: Monitoring objects inside the GAS

for the question: which API functions for external service are accessible by the Gridsphere user. Requirements for this integration will be presented soon.

5 Enable Scenarios for first release of the GAS

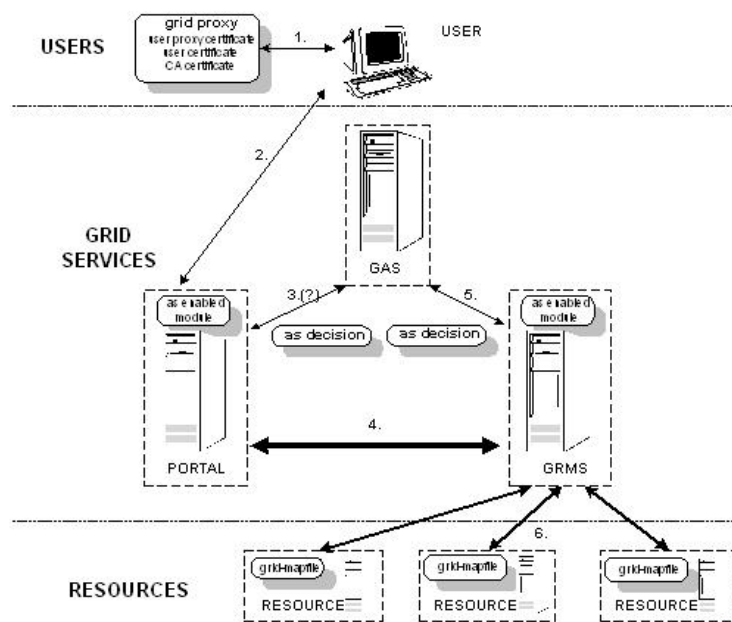


Figure 8: The Eger scenario

Each scenario, which can be used for the GRMS integration and the Portal integration needs two simple features:

- it has to be possible to manage the GAS server,
- it has to be possible to return the authorization decision upon the user request.

Such a sample scenario (named the Eger scenario) is shown on figure 8. This scenario is topical for the first release of the GAS. However, for integration with monitoring service the Eger scenario is not sufficient. In this situation another scenario is needed that would make possible to get the logical part of security policy upon user's request. This scenario is partially developed and will be finished soon.

In the near future a `scenario` module will be developed. The `scenario` module will extend the possibility of using different scenarios. It will be possible to use more complex security scenarios containing a session object for the users of the GAS. There is a plan to extend the possibility of managing scenarios, and it should be possible to:

- use more than one security scenario;
- load new scenario online from the text file which describes the scenario;

6 The near future of the GAS

6.1 Extending scenarios functionality of the GAS

One of the main reasons for creating the GAS was the need to have the system of authorization, which can work with the varied security scenarios. In the first prototype it is possible to work with the two scenarios: the CAS compatible scenario (in which full policy is generated for a user) and the simple super-scheduler scenario (in which a service gets the authorization decision from the GAS for a user). After the first release, the extension of the scenarios engine will be provided to work with other security scenarios (More Complex Super-Scheduler Scenario, User to User Asynchronous Collaboration Scenario, ...). The more accurate scenarios are described in the *Technical Specification for the Grid Authorization Service* [2].

6.2 Extending method of generating security policy

The methods of generating the security policy are significantly simplified and not fully optimized in the first version. After getting sufficient experience about performance and other aspects of generating the security policy, it will be advisable to make a review of the codes and make same corrections and improvements.

6.3 GUI `gas_admin_client`

For the first release it is possible to use the command line tool `gas_admin_client` for managing the GAS server and simple security portlet in Gridsphere (which works similarly to the `gas_admin_client`). Next, it is planned to develop the GUI admin client which can help the user to manage the `gas_server`. This client is likely to be written in Java and will extend the functionality of portlet which was written.

6.4 Possible integrations with other security solutions

The main security solution which is supported in the GAS is GSI (the Grid Security Infrastructure). From the beginning the GAS will have the modular structure and this solution makes it possible to divide specific functionalities into modules. All connections (the functions calling) to GSI will be put into one module and there will be a specific set of API functions which will communicate with this module. This situation enables to use other security solutions without the modification of the internal structure of the AS, only by exchanging one module. The new module will have to cover all functions calls to this new security solution. It is planned to make support for Microsoft or SUN security technologies, if there is enough time to develop, but obviously the GAS will be ready to support the future security technologies.

7 Summary

The first release of the GAS is ready to work and integrate with other workpackages. The GAS is ready for many integrations and applications by using a complex data structure, and the SQL database server for storing this structure. The fact is that the GAS server has worked only in the sample testing environment until now. After just a few full integrations it will be possible to answer the question how many requirements of external services the GAS server fulfills and what is missing inside the GAS. It is important to emphasize that the GAS server still has not been sufficiently tested and requires a lot of tests in order to eliminate possible errors. On the other hand, there are many extensions planned for this service. Some of these extensions like scenarios support, or the functionality of the authorization module are very critical, others not so critical but important from the GAS users' point of view (like the GUI client for managing the GAS). The first release of the GAS is currently available but there is still a lot of work needed to make the GAS more useful and attractive for potential users.

References

- [1] M. Adamski, M. Chmielewski, S. Fonrobert, A. Gowdiak, B. Lewandowski, J. Nabrzyski, T. Ostwald, and J. Pukacki. *WP6 Security: Initial Requirements*, April 2002.
http://www.gridlab.org/Internal/Drafts/wp6_requirements_draft.pdf.
- [2] M. Adamski, M. Chmielewski, S. Fonrobert, A. Gowdiak, B. Lewandowski, J. Nabrzyski, T. Ostwald, and J. Pukacki. *Technical specification for Authorization Service*, August 2002.
<http://www.gridlab.org/Resources/Deliverables/D6.2b.pdf>.
- [3] M. Adamski, M. Chmielewski, S. Fonrobert, A. Gowdiak, B. Lewandowski, J. Nabrzyski, T. Ostwald, and J. Pukacki. *Implementation and Test Plan*, January 2003.
<http://www.gridlab.org/Resources/Deliverables/D6.3.pdf>.
- [4] The Globus Project. *CAS User and Administrative Guide*, August 2002.
http://www.globus.org/security/cas/alpha-r2/cas-guide-v0_2.pdf.
- [5] Information Technology Industry Council (ITI). *Role Based Access Control*, April 2003.
<http://csrc.nist.gov/rbac/rbac-std-ncits.pdf>.
- [6] Object Management Group, Inc. *Resource Access Decision Facility Specification*, April 2001.
<http://www.omg.org/docs/formal/01-04-01.pdf>.