



## WestminsterResearch

<http://www.wmin.ac.uk/westminsterresearch>

### Security models for lightweight grid architectures.

**Lazar Kirchev<sup>1</sup>**  
**Minko Blyantov<sup>1</sup>**  
**Vasil Georgiev<sup>1</sup>**  
**Kiril Boyanov<sup>1</sup>**  
**Maciej Malawski<sup>2</sup>**  
**Marian Bubak<sup>2</sup>**  
**Stavros Isaiadis<sup>3</sup>**  
**Vladimir Getov<sup>3</sup>**

<sup>1</sup> Institute on Parallel Processing – Bulgarian Academy of Sciences

<sup>2</sup> ACC CYFRONET & ICS AGH, Poland

<sup>3</sup> Harrow School of Computer Sciences, University of Westminster

This is a reproduction of CoreGRID Technical Report Number TR-0020, January 25, 2006 and is reprinted here with permission.

The report is available on the CoreGRID website, at:

<http://www.coregrid.net/mambo/images/stories/TechnicalReports/tr-0020.pdf>

---

The WestminsterResearch online digital archive at the University of Westminster aims to make the research output of the University available to a wider audience. Copyright and Moral Rights remain with the authors and/or copyright owners. Users are permitted to download and/or print one copy for non-commercial private study or research. Further distribution and any use of material from within this archive for profit-making enterprises or for commercial gain is strictly forbidden.

---

Whilst further distribution of specific materials from within this archive is forbidden, you may freely distribute the URL of WestminsterResearch.  
(<http://www.wmin.ac.uk/westminsterresearch>).

In case of abuse or copyright appearing without permission e-mail [wattsn@wmin.ac.uk](mailto:wattsn@wmin.ac.uk).

# Security Models for Lightweight Grid Architectures

*Lazar Kirchev<sup>#</sup>, Minko Blyantov<sup>#</sup>, Vasil Georgiev<sup>#</sup>, Kiril Boyanov<sup>#</sup>, Maciej  
Malawski<sup>\*</sup>, Marian Bubak<sup>\*</sup>, Stavros Isaiadis<sup>†</sup>, Vladimir Getov<sup>†</sup>*

*<sup>#</sup>Institute on Parallel Processing – Bulgarian Academy of Sciences  
e-mail: [l kirchev, mblyantov, vasko, boyanov]@acad.bg;*

*<sup>\*</sup>ACC CYFRONET & ICS AGH, Poland  
e-mail: [malawski, bubak]@agh.edu.pl;*

*<sup>†</sup>University of Westminster, UK  
e-mail: [s.isaiadis, v.s.getov]@wmin.ac.uk;*



---

CoreGRID Technical Report  
Number TR-0023

January 25, 2006

Institute on Grid Systems, Tools and Environments

CoreGRID - Network of Excellence  
URL: <http://www.coregrid.net>

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme  
Project no. FP6-00426

# Security Models for Lightweight Grid Architectures

Lazar Kirchev<sup>#</sup>, Minko Blyantov<sup>#</sup>, Vasil Georgiev<sup>#</sup>, Kiril Boyanov<sup>#</sup>, Maciej Malawski<sup>\*</sup>, Marian Bubak<sup>\*</sup>, Stavros Isaiadis<sup>†</sup>, Vladimir Getov<sup>†</sup>

<sup>#</sup> *Institute on Parallel Processing – Bulgarian Academy of Sciences*  
e-mail: [lkirchev, mblyantov, vasko, boyanov]@acad.bg;

<sup>\*</sup> *Academic Computer Centre CYFRONET - Krakow, Poland*  
e-mail: [malawski, bubak]@agh.edu.pl;

<sup>†</sup> *University of Westminster, UK*  
e-mail: [s.isaiadis, v.s.getov]@wmin.ac.uk;

CoreGRID TR-0023

January 25, 2006

## Abstract

*Security management is important for the effective functioning of a grid system. Here we present a security management model developed for the lightweight multilevel grid system. We decided to implement a role based security model, where a number of roles are defined for the grid. Different roles have different access restrictions. They are assigned to users and this is how users receive rights in the grid. Every cluster in the grid has security service, which handle user's identification and access control. The role based access control allows us to incorporate some simple access decision logic in the information service, which makes controlling user rights easier. Further in this paper we compare our security architecture with that of the lightweight middleware H2O and MOCCA in order to identify the important requirements, common concepts and technical solutions which may be reused.*

## Key words

Grid Computing, Lightweight Architecture, Web Services, Security Model, User Scenario, Role-Based Security

## 1. Introduction

Security is a central issue in computational grids. These are usually composed of resources in multiple administrative domains, and are accessed by large numbers of users. In order to maintain a certain degree of security it is important that not every user has the same level of access to the available resources.

The purpose of this report is the construction of a lightweight grid in the light of security and user management requirements. We set as a main priority the simplicity of the architecture and implementation. In our architecture, the grid system consists of inter-connected clusters of computers/nodes. The cluster is the unit for node organization in the system. It is logical rather than physical unit – it is not necessary that nodes in the same cluster reside on the same physical location.

---

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

We define the following as the most important security requirements for grid: user management (including user registration and user authentication), authorization for access to resources, data encryption and security of executing code, delegation of rights and, finally, auditing of user access and resources usage. Security management model however is a relevant issue for many other system services, e.g. services that are responsible for discovery and allocation of resources. We have considered different solutions for the above-mentioned issues. There are presently different approaches to grid security and we will present the most important ones in the paragraphs that follow.

In the Globus Toolkit [6] most security issues are handled by the Grid Security Infrastructure (GSI) [3, 5]. In GSI three entities exist: users, resources and programs. Every entity has a certificate that represents its global identity. The certificate is in standard X.509 format and contains the global name of the entity as well as additional information (e.g. the public key). Verification of identity is done using SSLv3 protocol which also verifies the identity of the Certification Authority that issued the certificate. In GSI delegation of rights is supported – one entity may delegate its rights (or part of them) to other entities using a short-term proxy. The proxy certificate is signed by a user or another proxy as opposed to the user certificate which can only be signed by the Certificate Authority. In this way a chain of delegation is created that reaches the Certification Authority. Every resource may specify its access policy (with Access Control List or in some other way). The authentication protocol verifies the user's global identity, but the global name should be mapped to a local user name, which resides on a local system, in order to be used by the local security subsystem. In order to achieve this, the GSI uses a map file in which the mapping between global and local names is recorded.

An extension to the GSI is the Community Authorization Service (CAS) [16]. CAS makes it possible for a security policy to be enforced on the basis of a global identity, so that mapping to local user accounts is not necessary. The CAS server stores information about who has permission, what permission is granted and which resource is the permission granted on. The resource owners give access to a community (e.g. a Virtual Organization) as a whole, and the community defines the finer rights. When a community member wants access to a resource, he / she sends a request to the CAS server. The server checks if the community policy permits such access. If it does, the server issues a capability that allows the user to perform actions. The user presents this capability to the resource server. The latter checks if both the local policy and the capability permit the requested access, and if so the resource server grants the requested access to the user. The CAS server is managed by an administrator who adds users, creates groups, and assigns rights and roles. The credentials that the CAS server delegates to the users have the form of restricted proxies. In this approach the CAS server is susceptible to possible security-caused problems – for example, a denial of service attack against it may cut off the users from the resources.

Another solution for authorization, to some extent similar to the CAS server, is the Virtual Organization Management Service (VOMS) [4], implemented in the DataGrid project. Every Virtual Organization has a VOMS server, which stores all user information – accounts, rights, groups, and roles. When a user authenticates, he/she contacts the VOMS, presenting his/her certificate and VOMS creates a proxy certificate, which includes a list of the user's groups, roles and rights, as well as the user's credentials. This proxy certificate is signed by the VOMS server and presented by the user when requesting access to a resource. The permissions, granted by a VO, can be locally overridden.

JGrid [10] is a Java and Jini based computing grid infrastructure, developed by the Veszprem University, Hungary. For user management it uses two services [11, 14]. The

Authentication Service is responsible for user authentication and single sign-on. It issues short-term credentials (private key and X.509 certificate) to those users, who have no long-term certificate. The Registration Service stores all user information and offers role based access control (in which each role represents a permitted actions list), user registration and user management (for the administrators). When the user wants to use a service, he contacts the authentication service, which issues the user a certificate containing the user's registration information. This certificate is used for user authentication hereafter. Then the user contacts the service via a SSL session (the short-term certificate is sent to the service). It receives the user information, containing his roles, from the registration service and decides whether to grant access to the user.

GridLab [7] is a project of the Poznan Supercomputing and Networking Center. GridLab uses a Grid Authorization Service (GAS) [8, 2] for controlling user access. It represents a single logical point for defining security policy. The GAS subsystem comprises an AS Server, database, management module, and has modules for communication with services/applications/users and integration with other security solutions. It has initial support for the Role Based Access Control (RBAC) security model. The GAS is responsible for a single Virtual Organization (VO). The rights are given to users and groups and are for specific objects. When the user wants access to a resource, he or she requests authorization from the AS Server of the VO. The access policy is kept in a database, and the information is in the form of triples subject (user), object (resource) and their attributes, which describe the specific rights. Using the database, the AS decides whether to give or deny access.

PROGRESS [18] is another project carried out by the Poznan Supercomputing and Networking Center in collaboration with other institutions. This Grid system [12] has a portal, where authentication and access control take place. The authentication is done with username and password. For user identification an Identity Server is used, which authenticates the user and manages sessions. For authorization the Resource Access Decision [19] model developed by the OMG is used. The authorization is performed by Resource Access Decision (RAD) Module. The resources and services are classified in a number of types and roles representing specific rights for every type are created. The access rights are associated with the roles. When a user is authenticated, he is issued a token (which usually is the user's session cookie). At resource access time the user's token is sent to the portal, which includes it in the call to the Grid Service Provider (the subsystem that is responsible for resource management). The Identity Server verifies the token, and after that the user name is extracted from it. A request to the RAD module is made, containing the user name, the requested access rights and the resource. The RAD module decides whether the user has rights, and if so – the operation is carried out and the result is returned to the portal.

H2O [9, 13] is a component-based and service-oriented framework, intended to provide lightweight and distributed resource sharing. It is developed at the Department of Math and Computer Science at Emory University. This architecture is based upon the idea of representing resources as software components, which offer services through remote interfaces. Resource providers supply a runtime environment in the form of component containers (kernels). These containers are executed by the owners of resources and service components (pluglets) may be deployed in them not only by the owners of the containers, but also by third parties, provided that they possess the proper authorization. The clients access the services through their interfaces. For interaction of components different protocols may be used. The security mechanisms in the framework are located at the containers. Both containers and services are accessed through security proxies. The execution of the pluglet's code is under the control of the container. Code, invoked by the container, is executed in the security context of the pluglet deployer, while the

remote calls are executed in the security context of the user, who obtained a particular reference. Authentication is performed through JAAS. X.509 certificates may be used but are not mandatory. The protection of resources owned by the container providers is performed by the Java security model and JAAS. Two types of policies are established – global and pluglet-related. The global one, specified by the container provider, defines what actions the code of a pluglet may perform. The provider grants permission on the bases of the source, where the code comes from, the party that has signed it, the party that invokes it, and the time when the code is allowed to be executed. This is achieved by substituting the default JRE policy provider with a custom one in the framework. The pluglet policy is specified by the service deployer and controls access to particular interfaces. The deployer may specify exactly which remote interfaces may be used by a particular client.

After considering different solutions, we decided to implement in our grid a role based security model, as in [4, 14, 8]. A Role Based Security Model (RBAC) is also used in the PERMIS System for user authorization [17]. In this model, a number of roles are defined in the grid and rights are associated with roles, not with particular users. This approach has the advantage that there is no need to assign rights to every user separately – the user is assigned one or more roles and he / she automatically receives the rights with them. We consider this to be a scalable solution. Moreover, the node manager of every cluster node may impose further restrictions upon the access to the resources, which it controls. In this way an additional flexibility of defining access policy is gained. For security at the level of code execution, sandboxing will be used, similar to the approach taken in the AliCE grid system [1, 20]. Thus, data and code security are guaranteed by the implementation technology that will be used, namely Java, as is in the case of H2O project [13]. A detailed comparison between our grid system and H2O is presented in the paper.

In the rest of the paper we will discuss the following: Section 2 delves upon the user profile management, in Section 3 the user authentication mechanism is presented; Section 4 makes overview of the authorization; Section 5 examines some issues connected with auditing, data and code security, Section 6 gives examples of grid access; Section 7 elaborates on the comparison of our security model and that of H2O and addresses possible collaboration with H2O/MOCCA [13, 15] in the future; and Section 8 makes a conclusion.

## **2. User accounts**

Before discussing the user profile management in our security model, we will make a short overview of the grid architecture. We assume the grid consists of interconnected clusters of computers. The architecture is hierarchical and has three levels. The first level is the local level where all the clusters that comprise the system reside. The second level is made by connecting the clusters and represents the grid. The last level is the intergrid level. At this level a connection with other grids is made possible. Both the grid level and the cluster level have portals – the grid portal acts as the grid entry point and the portal of every cluster is its entry point. These portals present the user with a front-end for logging into the grid or a particular cluster. The cluster is the unit used for organizing resources and users in the grid. It is logical rather than physical – it is not necessary that the nodes, which belong to one and the same cluster, be physically close to each other. The system services are local for every cluster, and also there are respective system services for the grid level.

Every user in the grid will have a personal user account. The account will be unique in the whole grid and every person will have one account – not different local accounts and one global, as is the solution used in other grid systems such as Globus for instance [2]. This user account belongs either to one of the clusters that comprise our grid, or to the grid level, if it is for an

external user. In order to use resources in the grid, the user should have an account. So when he/she contacts the grid portal for the first time, the user will be prompted to register so that an account will be created for him/her. In this case, the account will be created at the grid level and the user information will be stored at the Grid Information Service. When the user registers at a cluster portal, the account that will be created for him/her will belong to the respective cluster and the user information will be stored by the Cluster Information Service. At registration – regardless if the user registers at the grid or cluster portal - the user will provide various personal information and on the basis of this information the grid or cluster administrator will decide what rights to give to the user account. At first, when the account is created, it will be assigned the most restricted rights, and after that the administrator will change them according to the user's needs and personal information. Some users may have rights only to access resources that are local to the cluster, where their account belongs, while others may have rights to access resources in other grid clusters too. Thus we may distinguish five types of grid users:

1. System Grid Administrator - has administrative rights to the whole grid.
2. System Cluster Administrator – has administrative rights only to the cluster which he/she administers.
3. User with Cluster access – has access only to the local cluster to which his/her account belongs.
4. User with Cluster and Grid access – has access to his local cluster and the global grid.
5. External Grid users – users who are not members of any cluster.

The information for the cluster users will be kept in a database at the cluster where the user's account belongs. This database will be maintained by the Cluster Information Service. At the grid level the information about the external user accounts will be stored and maintained by the Grid Information Service. This service will also keep information about the rights of the users in clusters, different from their own. Groups will be used for better organization of user accounts. A role based security model will be implemented where roles describe the type of access. There is a number of roles for the different resource types, and for every type of role there are “subtypes”, which represent different levels of restriction on the access to the respective type of resource, represented by the role. The grid administrator and cluster administrators will assign roles to users and groups.

The roles defined in our grid system are listed in Table 1.

<b><i>Resource Type and Resource Usage</i></b>	<b><i>Weak</i></b>	<b><i>Normal</i></b>	<b><i>Strong</i></b>
Compute	WeakComputeUsage	NormalComputeUsage	StrongComputeUsage
Storage	WeakStorageUsage	NormalStorageUsage	StrongStorageUsage
Data	WeakDataUsage	NormalDataUsage	StrongDataUsage
Network	WeakNetworkUsage	NormalNetworkUsage	StrongNetworkUsage
Software	WeakSoftwareUsage	NormalSoftwareUsage	StrongSoftwareUsage
Services	WeakServicesUsage	NormalServicesUsage	StrongServicesUsage

Table 1. Map of User Roles.

The users with cluster access have rights for the local resources in their cluster, which are defined by a set of roles, kept by the Cluster Information Service. The users with cluster and grid access have rights for resources in remote clusters too. The roles for these users, which define their rights for access of remote resources, are kept by the Grid Information Service. These roles are essentially the same as the local roles, the only difference being that they give access to remote resources. Thus, the rights of each user in the system are defined by the roles, he/she possesses. Diagram 1. illustrates the organization of users and roles.

When a service is published, a part of its description in the Information Service database will be the list of roles, which have access to it, and also a list of roles, which the service needs for execution (in fact, these roles identify what resources the service will use). In this way the owner of a service may define the access level to his/her service. Moreover, the cluster administrator is able to change the list of roles, associated with a service. Thus the access policy is controlled locally and the access rights may vary for the users with same roles but of different clusters.

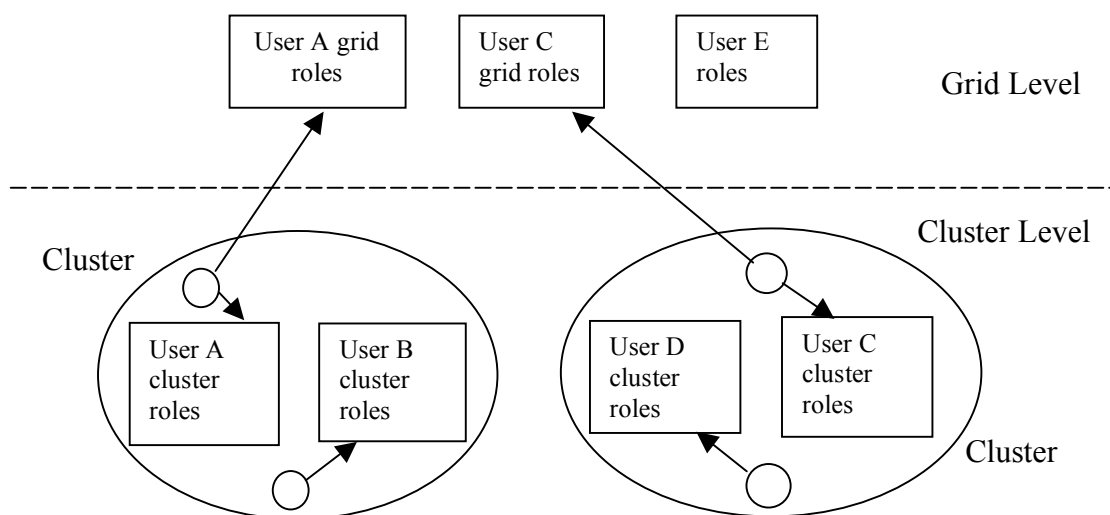


Diagram 1. Users and Roles.

Another feature will be the opportunity to restrict user access locally. For every node in the cluster there will be a node manager, which controls the functioning of the node. For example, it monitors the execution of jobs on the node, sends information about the current status of the node to the information service and so on. It will be possible to make further restrictions on the access to the node's resources by stating which roles may start processes on the node. The node manager will be responsible for enforcement of the restrictions.

In addition to the individual accounts associated with a particular cluster, there will be a guest account, which will offer anonymous access to the grid. This account will not reside at a particular cluster and will give very restricted rights.

The users will be able to view their personal information and eventually change it. This should be possible only with the knowledge of the administrator, because the user access rights may depend on this information. Thus a change in it may result in granting more rights, or



revoking rights. The administrator has the right to add, remove and change user accounts, as he is the one who defines the local security policy for the cluster.

### **3. Authentication**

Authentication is the process of proving one's identity. There are many different methods for authentication. Very often digital certificates are used to authenticate a user, but the method used depends mostly on the security level needed in the grid. We plan to support different mechanisms for authentication – at the beginning we will maybe use a simple username-password authentication and extend the functionality in the future so that it supports digital certificates and other methods. Here it is possible, according to the type of authentication used, different restrictions on the rights to be imposed – if a strong authentication is used more rights may be given to the user.

In order to use grid resources, the user has to authenticate first. For authentication the user contacts the grid portal or a cluster portal and requests authentication. In case the user contacts a cluster portal, the authentication is performed by the Cluster Security Service, which checks the username and password in the Cluster Information Service's database. If the user contacts the grid portal and the user is not an external user, the portal send the request to the Grid Security Service, which in turn forwards it to the Cluster Security Service of the cluster where the user account belongs. For this purpose, there will be a small database at the portal for mapping users to clusters. In order such mapping to be possible, every user should have a unique identifier. Digital certificates contain distinguished name, which is globally unique. But at the beginning we will not use certificates, and even when we begin to support certificates for authentication, we may not restrict users to use only this method for authentication. The solution is at registration time a unique identifier for the grid to be issued to every user. User e-mails may be used as identifiers, because they are globally unique, and the users remember their own e-mails. If the user is an external user, he/she is authenticated by the Grid Security Service, which uses the Grid Information Service.

After the user is authenticated, a token (which may be a short-lived proxy certificate or a session cookie) is issued to him/her, which includes the user's identity information and his/her roles. This token will be used when requests for a service are made.

The user will then see a personalized page, which lists the services, to which the user has access. Through this page the user may start a service or submit a job. The user will not see services to which he/she has no access. This filtering will be performed on the bases of the lists of roles in the description of every service in the Information Service database. Thus the user will see only those services with relevant roles. These lists with roles will be used with a similar purpose when a user submits a job. In this case, when the Resource Broker contacts the Information Service requesting a list of resources for executing the job, the Information Service will return only those resources which the user has right to use.

All user information – identity information, roles, etc. - will be stored in a database. Actually, there will be such a database in every cluster, and one for the grid level. The latter will store information only about the external user accounts and about the roles of users in other clusters. The Grid Information Service will control the grid level database whereas the Cluster Information Services will control the databases in the clusters. In fact, the Information Services will store all the information in the grid – the user information, resource information, roles descriptions, etc. Thus, the Information Services will act as wrappers for the databases with the different information, but will also offer additional functionality, such as filtering services and

resources for example. The Security Services will use the Information Services to retrieve user information when authenticating users. The Resource Management Services will use them for locating resources. Any service that needs to store information will use the Information Services.

#### **4. Authorization**

Authorization is the process of determining if the user has the proper rights to perform an operation – e.g., use a service. Every request for a service will be made at the grid portal or some cluster portal. Before making any request, the user should be authenticated. Actually, a direct request for a resource by the user will not be possible. The user will be able to perform two kinds of actions – submit a job for execution, and use a service. The resources will be presented as services, so that the use of a resource will be actually a use of a service. In the first case – job submission – the resources will be reserved by the Resource Management Service (RMS), and not directly by the user. When making the reservation, the RMS will have the user's token so that it will know the user's roles and identity. When the RMS contacts the Information Service (IS) while searching for resources, it will send as a part of the request the user's roles, so that the IS will be able to filter the resources according to the roles. After that, when the RMS decides which resource(s) will use from the list, made by the IS, it will contact the node manager of the resource. The node manager will check if there are no local restrictions for the user's roles and also it may make a request to the Cluster or Grid Security Service to find whether the user really has the roles stated in the token. Thus, an additional security check is added to the process of granting access to a resource. It is meant for cases where a user's token is forged.

Another important issue concerning authorization is the delegation of rights. When a process is started on behalf of the user – for example, a process to do some computation while performing a submitted job – the process should have the rights of the user in order to be able to use resources. That is why at creation time the process will be issued a token with the user's roles. It is possible the process's token to include only those user roles, which are needed for its work.

In our system there will be no special authorization service. With the chosen security model, authorization is implicitly realized by the Information Service, the resource and service providers when publishing resources and services, and eventually the node manager. Thus the authorization decision mechanism is incorporated in the functioning of the grid system.

#### **5. Logging and Code Security**

Logging is connected with grid security, but also with accounting and grid statistics. It is important that each user's authentication (login) be logged. The information will be stored in the database with user information in Cluster Information Service (or Grid Information Service if the user is external for the grid). Also, every request for service use should be logged at the cluster where the resource resides and every successful grant should also be logged. This information gives statistics about users and resources – for every resource the administrator will know how much it is used, and for every user – how much time does he/she uses the grid services. These statistics may be used for optimizing the functioning of the grid. For example, if some type of resources is used very much, it will be evident that more resources from this type are needed in the grid. The statistics may also be used for security purposes where they may help detect an intrusion or a breach in the grid security. They are also very important for calculating how much users should pay for the resources they used. Since we plan our grid as an economy grid, this application of statistics for accounting purposes is more important than in other grids, which do not have economy features.

Code and data security is another important aspect in grid security. For providing security for the stored data, we will store it in an encrypted format. Only those users, who have the permission to use the data, should be able to decrypt it. Moreover, the communication between the remote clusters, and even within the cluster (the nodes in the cluster may be physically remote) should use some secure protocol, such as SSL. For security of executed code and detection of malicious code, the Java technology security features will be used. The JVM offers language safety, type safety, byte code verification and protection domain mechanism. For enhancing security control, the JAAS framework may be used, which adds security features to the standard JVM features.

## **6. Grid Access Use Cases**

In this section we present two use-cases which demonstrate how our system functions. First we consider the case of job submission (we may call this using a non-persistent service), and after that the case of using a service (persistent service – it is offered by the grid system). As can be seen, the work done by the system in both cases is similar.

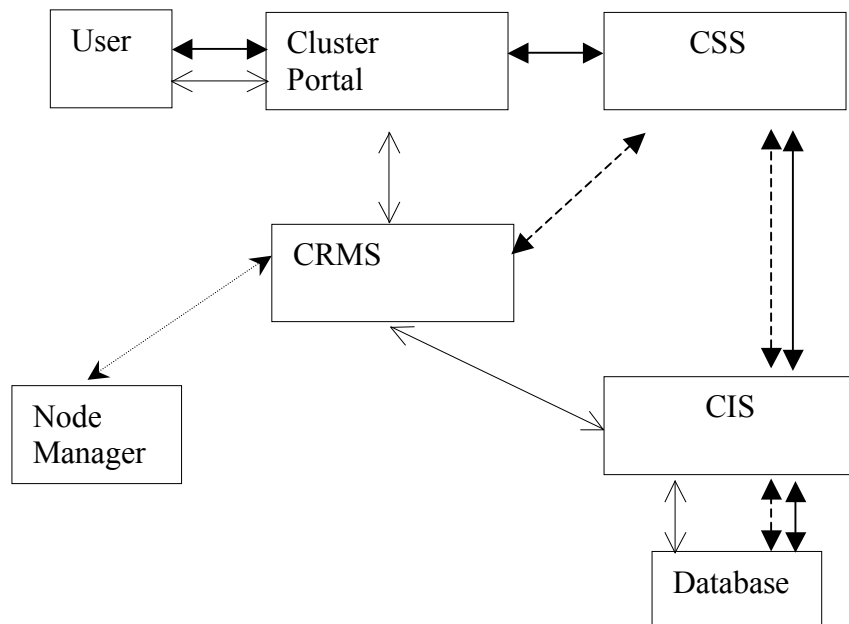
### **6.1. Job Submission**

Suppose a user wishes to submit a job to the cluster portal. Following are the steps that are usually performed in this case.

1. First of all, he or she must log on the grid. This is done at the grid or cluster portal. After successful logon, the user has a token with his or her identity and other information, such as the user's roles. Now the user may submit the job.
2. After the job is submitted, the portal contacts the Cluster RMS (CRMS), which is responsible for scheduling the job for execution on particular resources.
3. The CRMS may send request to the Cluster Security Service (CSS) for confirmation that the user really has the roles listed in the token. If the roles are confirmed, the CRMS proceeds with the scheduling. This additional step may enhance security in case of a forged token.
4. The CRMS uses the Cluster Information Service (CIS) in order to find the possible resources. The CIS will search its database for resources, which meet the requirements of the task. Since in the description of every resource the roles, which have access to it, will be listed, the CIS will be able to filter the resources and return to the Broker only those, to which the user has access.
5. After the results reach the CRMS, it decides which of them to use for the job (if there are more than needed) and partitions the job among them in case the job may be parallelized.
6. The CRMS contacts the chosen resources' Node Managers. They should confirm that the resources are free at the moment and may be used. The Node Managers also check whether there are any additional restrictions on the user's roles, which do not allow the execution of the task on the node. If there are no such local restrictions the job is started.

In the above scenario, a user cannot request directly a resource – every request is issued by the CRMS.

Diagram 2. illustrates the sequence of steps, performed by the system for a job submission.



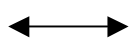
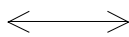
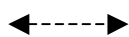

-  communications for the initial user authentication
-  communications for submitting a job by the user, finding resources and returning the result
-  communications for confirmation of user's identity
-  communications between the Node's Manager and the CRMS (giving the task and return of the result)

Diagram 2. Job Submission in Steps.

The situation is similar if the user contacts the Grid Portal and not a Cluster Portal. In this case after the job is submitted, the request is forwarded to the GRMS (here, too, the GRMS may request confirmation of user roles from the GSS), which contacts the Grid IS (GIS) for resources for execution of the job. The GIS communicates with CIS of the clusters in order to find available resources. After resources are found, the task is forwarded to the proper CRMS.

Both when a task is submitted to the Grid Portal and a Cluster Portal, it is possible if there are no available resources at the moment the task to be scheduled for a future moment.

## 6.2 Using a service

In addition to submitting jobs to the grid, the user will be able to use other resources and services. All resources will be presented as services, so using resources will be in fact using services. These services may be called persistent services – they are offered by the grid system and are not submitted by the users. Now suppose that the user wants to use a service. After the user is successfully authenticated, the user sees all services to which he or she has access. Now the user finds the service that he or she wishes to use. The cluster portal forwards the user request for the service to the CRMS and from now on the steps are the same as in the previous use-case. If the user contacts the grid portal, then the request is processed by the GRMS, in a similar way to the previous use-case.

## 7. Comparison to H2O Middleware and Future Work

When comparing the proposed architecture and security model of the our grid platform with those found in H2O, we can find several differences but also many commonalities. The main feature distinguishing H2O from other grid middleware is the separation of roles of resource owners (providers) and service deployers. This means that the provider may offer only a raw computational resource to share, and the role of service deployment is left to authorized parties (deployers) who are allowed to deploy pluglets into H2O kernels. This is distinct from the standard scenario proposed by OGSA, where services (even if transient) are offered and deployed by resource providers. Such standard scenario may cause a barrier discouraging providers from sharing, especially when the process of installation (deployment) of services is sophisticated and time consuming. The H2O sharing model takes much of the burden from resource providers to the deployers, therefore encouraging providers to share, e.g. in P2P metacomputing scenario. We would consider it valuable if our architecture could also support such a model of resource sharing with dynamic service provision (deployment), as it is in H2O.

The important part of security mechanisms in H2O is involved in the definition and enforcement of security policies. Both resource providers and pluglet deployers may specify their Java security policies, granting detailed set of permissions to the code executed by clients. The policies are based on the JAAS framework and extended with time-based constraints, protecting providers from malicious or erroneous code run by clients as well as restricting access to system resources (filesystem, network, etc.). We believe that these mechanisms, which are implemented in the H2O kernel may be useful for the building the prototype implementation of the GrOSD platform. We can observe, that the H2O does not implement the role-based security model in the form proposed here. The users and their roles in H2O are constrained to the H2O kernel boundary, because of the assumption of independence of kernel providers, who are not assumed to be aware of each other. However, as the H2O is based on the JAAS framework and Pluggable Authentication Modules (PAM), then it it should be possible to plug in the authentication method using Cluster Security Service. This possibility and also potential applicability of restricted X.509 proxies as those known from Globus GSI should be subject to more detailed investigation.

Another important observation is that since the focus of the CoreGrid project is on a component approach for programming grid systems, then adopting several features from the H2O to the our architecture will enable easier integration of the latter with the MOCCA component framework. This will lead to the possibility of running MOCCA component applications on our platform, taking advantage of the simplicity and scalability of the lightweight platform for resource sharing, as well as providing a simple and powerful component programming model.

Alternatively, we may consider the possibility of using MOCCA itself as a base component technology for building prototypes. Such features as dynamic deployment of components on shared resources using H2O mechanisms, inter-component communication using RMIX and simple programming model should provide a sufficient base for a lightweight, simple and scalable grid platform, adding the modularity and flexibility to the prototype. The detailed elaboration of such possible design will be the subject of our future research agenda.

## 8. Conclusion

We introduced the security architecture for a lightweight hierarchical grid. Our main purpose in making the design was the simplicity and scalability. We chose a role based access control security model and made some changes to the existing realizations of the model. Namely, the roles are not associated with particular rights and the access is specified by every resource and

service listing the roles that may use it. The lower level security will be realized by the Java technologies that will be used for the project implementation. We believe that this security model best suits our goals.

## References

1. ALiCE Grid Computing Project (<http://www.comp.nus.edu.sg/~teoym/atsuma.htm>)
2. Technical Specification for Authorisation Service, Marcin Adamski, Michal Chmielewski, Sergiusz Fonrobert, Jarek Nabrzyski, Tomasz Nowocien, Tomasz Ostwald, <http://www.gridlab.org/Resources/Deliverables/D6.2b.pdf>
3. "Design and Deployment of a National-Scale Authentication Infrastructure", R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, V. Welch., IEEE Computer, 2000
4. DataGrid Security Design, DataGrid Security Co-ordination Group, <http://edms.cern.ch/document/344562>
5. "A Security Architecture for Computational Grids," I. Foster, C. Kesselman, G. Tsudik, S. Tuecke, Proc. 5th ACM Conference on Computer and Communications Security Conference, pg. 83-92, 1998.
6. Globus Toolkit (<http://www.globus.org/toolkit>)
7. GridLab project ([www.gridlab.org](http://www.gridlab.org))
8. GridLab Security Architecture (<http://www.gridlab.org/WorkPackages/wp-6/index.html>)
9. H2O Project, [www.maths.emory.edu/dcl/h2o/](http://www.maths.emory.edu/dcl/h2o/)
10. JGrid project ([http://pds.irt.vein.hu/jgrid\\_index.html](http://pds.irt.vein.hu/jgrid_index.html))
11. JGrid Requirements Document, Zoltan Juhasz, Krisztian Kuntner, Mark Magyarodi, Gabor Major, Szabolcs Pota, Department of Informaiton Systems, University of Veszprem, accessible from [http://pds.irt.vein.hu/jgrid/documentation/JGrid\\_Requirements.pdf](http://pds.irt.vein.hu/jgrid/documentation/JGrid_Requirements.pdf)
12. Authentication and access control in portals: the PROGRESS grid access environment, Kosiedowski M., Slowikowski P., Polski Internet Optyczny: Technologie, Usługi i Aplikacje – PIONIER 2003 conference, April, 9<sup>th</sup>-11<sup>th</sup> 2003, Poznan, Poland
13. Dawid Kurzyniec, Tomasz Wrzosek, Dominik Drzewiecki, and Vaidy Sunderam. Towards self-organizing distributed computing frameworks: The H2O approach. *Parallel Processing Letters*, 13(2):273–290, 2003. ([http://www.mathcs.emory.edu/dcl/h2o/papers/h2o\\_ppl03.pdf](http://www.mathcs.emory.edu/dcl/h2o/papers/h2o_ppl03.pdf))
14. The Security Architecture of the Jgrid System, Mark Magyarodi, Department of Informaiton Systems, University of Veszprem, accessible from [http://pds.irt.vein.hu/jgrid/documentation/JGrid\\_security.pdf](http://pds.irt.vein.hu/jgrid/documentation/JGrid_security.pdf)
15. M. Malawski, D. Kurzyniec, V. Sunderam, MOCCA - Towards a Distributed CCA Framework for Metacomputing, Proceedings of the 10th International Workshop on High-Level Parallel Programming Models and Supportive Environments, 2005
16. "A Community Authorization Service for Group Collaboration.", Pearlman, L., Welch, V., Foster, I., Kesselman, C. and Tuecke, S., IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, 2002.
17. Permis project (<http://www.permis.org>)
18. PROGRESS Portal (<http://progress.psnc.pl>)
19. Resource Access Decision, Version 1.0. Accessed from [http://www.omg.org/technology/documents/formal/resource\\_access\\_decision.htm](http://www.omg.org/technology/documents/formal/resource_access_decision.htm)
20. ALiCE: A Scalable Runtime Infrastructure for High Performance Grid Computing, Y.M. Teo and X.B. Wang, Proceedings of IFIP International Conference on Network and Parallel

Computing, pp. xx, Springer-Verlag Lecture Notes in Computer Science, Wuhan, China, October 2004.