Journal of Information Assurance and Security
ISSN 1554-1010 Volume 6 (2011) pp. 106-114

A Cooperative Model for Resource Sharing on Grid

Alessio Merlo

Dipartimento di Informatica, Sistemistica e Telecomunicazioni (DIST), University of Genova
Via all'Opera Pia 13, 16145, Genova (Italy)
alessio.merlo@dist.unige.it

Abstract: **The access to Grid resources depends on policies defined by the administrators of the physical organizations and of the Grid middleware. This approach does not require support for access control in the middleware, but since changes in the access control policy of the Virtual Organization imply the involvement of one or more administrators, it lacks the flexibility needed in several application scenarios. In this paper we propose a group-based access control model for Grid environments that increases the flexibility of the access control model offered by state-of-the-art Grid platforms without requiring changes in the middleware. The approach is based on collaboration among Grid users and allows them to exchange access permissions to Virtual Resources without the intervention of administrators. We show that our solution can be defined on top of the access control mechanisms offered by state-of-the-art Grid middleware and illustrate how the proposed model can be implemented as a service in a service-oriented Grid environment.**

Keywords: cooperative access control, service oriented grid, globus toolkit

I. Introduction

The Grid Computing paradigm aims to realize a common distributed environment in which resources are shared and accessed by many users independently from the organizations the users and the resources belong to. Grid Computing has been motivated by the waste of resources that generally affects the management of single administrative domains.
In fact, administrative domains (e.g. a company) usually make partial use of the available computational, storage and network resources and an effective usage of the available resources is commonly perceived as a key challenge [1].
Grid Computing middleware (e.g. Globus Toolkit 4 [2] and gLite [3]) tackles the problem by providing a virtualization layer that allows for the creation and management of Virtual Organizations (VOs) on top of Physical Organizations (POs), i.e. the administrative domains. Physical Resources (PRs) (e.g. CPU, RAM, disk storage) in different POs are then mapped into VRs, thereby making them accessible outside the boundaries of the POs they belong to.
A user of a PO (e.g. a researcher in a University) needs a Grid User (GU) identity in order to access VRs. (A GU is uniquely identified by an identifier that constitutes an entry point to the Grid.) VRs are more complex structures than PRs and individual VRs usually comprise different PRs. For instance, an execution service (e.g. the GRAM service of the Globus Toolkit [2]) is made of different PRs like disk storage, RAM and CPU. Similarly, a GU is associated with a set of Physical Users (PUs - e.g. accounts on different machines possibly belonging to different POs).

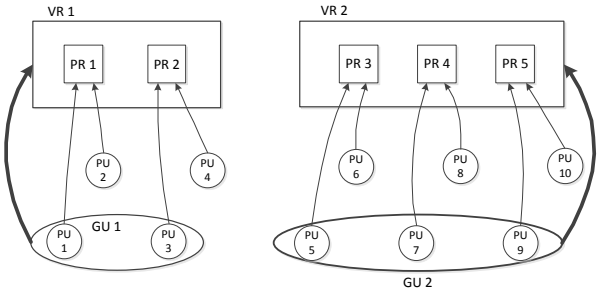


Figure. 1: Cross-layer Grid access control

The mappings between GUs and PUs and between VRs and PRs are kept in specific files and structures in current Grid middleware (e.g. the `grid-mapfile` in the middleware GT4) that can be manipulated only by the local Grid administrator (e.g. the *globus user*, a non-privileged account that has access to configuration files and structures in GT4).
By means of virtualization Grid middleware extends the visibility and accessibility of PRs, but this unavoidably complicates access control.
In a PO the access control policy is managed by an administrator who directly specifies the access privileges that registered users have on the available PRs.
In a VO access control is inherently distributed as there is no single administrator of the Grid. Moreover the PRs associated with the VRs are subject to the security policies defined by the administrator of the respective POs.
Thus, the access control policy at the Grid layer depends on the access control policy at the Physical layer as well as on the mappings that relate GUs and VRs with PUs and PRs respectively, as exemplified in Fig. 1.
This approach does not require support for access control in the middleware, but since changes in the access control policy of the VO imply the involvement of the administrators of the POs participating in the VO and/or of the local Grid administrator, it lacks the flexibility needed in several application scenarios. For instance, in collaborative environments

(e.g. research institutions) it is desirable to give GUs the freedom to share their access to VRs with other trusted GUs at their discretion. Also, in mission-critical applications (e.g. [4] and [5]) unexpected events may require a GU to access to VRs that the security policy as defined by the POs layer and at the Middleware layer would prevent.

In this paper we propose a group-based access control model for Grid environments that increases the flexibility of the access control model offered by state-of-the-art Grid platforms without requiring changes in the middleware. Our approach is based on collaboration among Grid users and allows users to share access permissions to VRs without the intervention of administrators.

We show that our solution can be defined on top of the access control mechanisms offered by state-of-the-art Grid middleware. This makes our approach non invasive and adaptable to different middlewares. In fact, non invasiveness frees the middleware from the burden of implementing proper connectors or extensions to support dynamic groups (that are implemented as common grid resources); as a consequence, the dynamic group model can be adapted to different general purpose middlewares.

Structure of the paper. In Sect. II we introduce the access control model in Grid environments and discuss its limitations. In Sect. III we present our extension to access control based on dynamic groups, while in Sect. IV we analyze how our model of dynamic groups can be effectively implemented on top of an existing Grid middleware in a non invasive way. Finally, in Sect. V we discuss the related works and in Sect. VI we draw some conclusions.

II. A model for Grid Access Control

Three layers are involved in the authorization decisions in Grid environments: the physical layer, the middleware layer and the Grid layer.

- **Physical layer.** The single administrative domains level. At this level, PUs are granted access to PRs according to the access control policy of the POs and enforcement of the access control policies is thus left to POs. For instance, a cluser of computers running the UNIX operating system will rely on the UNIX access control mechanisms to implement and enforce the access control policy of the PO they belong to. Let $PO_1, \ldots, PO_n$ be POs that participate in the VO. We model the access control policy of PO_i as the triple $PAC_i = \langle PR_i, PU_i, PA_i \rangle$, where PR_i is the set of PRs, PU_i is the set of PUs, and $PA_i \subseteq (PR_i \times PU_i)$ is the *permission assignment relation* of PO_i, for $i = 1, \ldots, n$. We assume that the sets of PUs and PRs in different domains are mutually disjoint, i.e. that $PR_i \cap PR_j = \emptyset$ e $PU_i \cap PU_j = \emptyset$ for all $i, j = 1, \ldots n$ with $i \neq j$. We define $PAC = \langle PR, PU, PA \rangle$, where $PR = \bigcup_{i=1}^{n} PR_i$, $PU = \bigcup_{i=1}^{n} PU_i$ and $PA = \bigcup_{i=1}^{n} PA_i$.

- **Middleware layer.** The Middleware layer is responsible for virtualizing the PRs of the single administrative domains into VRs of the VO. The middleware layer keeps track of the user correspondence between GUs and PUs as well as the resource correspondence between VRs and PRs:
  - *User Mapping:* $UR = \langle GU, PU, UM \rangle$, where GU is the set of GUs in the VO, $PU = \bigcup_{i=1}^{n} PU_i$ is the set of PUs in the different POs and $UM \subseteq (GU \times PU)$ is the *user mapping relation*. In GT4 this mapping is stored in the `grid-mapfile` and is only modifiable by the globus user.
  - *Resource Mapping:* $RR = \langle VR, PR, RM \rangle$, VR is the set of VRs in the VO, $PR = \bigcup_{i=1}^{n} PR_i$ is the set of PRs and $RM \subseteq (VR \times PR)$ is the *resource mapping relation*. In GT4 this mapping is defined in internal structures of the middleware and in the Monitoring and Discovering Service (MDS). The MDS is a basic service of GT4 that acts as an index of the services publicly available in the Grid.

- **Grid layer.** The Grid layer consists of the GUs and the VRs of the VO. At this layer, access control amounts to deciding whether any given GU gu is entitled to access a given VR vr. This depends on whether the PUs associated with gu have enough permissions to get access to the PRs associated with vr according to the access control policies of the respective POs. We model the access control policy at this layer by the triple $GAC = \langle GU, VR, GA \rangle$, where GU and VR are the sets of GUs and VRs of the VO respectively and $GA \subseteq (GU \times VR)$ is the *permission assignment relation* at the Grid level and is such that $(gu, vr) \in GA$ if and only if for all pr such that $(vr, pr) \in RM$there exists pu such that $(gu, pu) \in UM$ and $(pu, pr) \in PA$. If $(gu, vr) \in GA$, then we say that *gu is granted access to vr according to GAC*. The access control policy at the Grid layer therefore entirely depends on the access control policy and mappings defined at the lower layers.

To illustrate, consider the scenario in Fig. 1, where VRs VR_1 and VR_2 correspond to the POs PO_1 and PO_2 respectively. VR_1 is mapped to PR_1 and PR_2, and VR_2 to PR_3, PR_4, PR_5). Each of these PRs is then accessible by two PUs (i.e. PR_k accessible by $PU_{2k-1}$and PU_{2k}). We then suppose that GU_1 is associated to $PU_{2i-1}, i \in 1, 2$, and GU_2 to $PU_{2i-1}, i \in 3, 4, 5$. We obtain that each VR_i is accessible by the GU_i belonging to users of PO_i.

Let $PR = \{PR_1, \ldots, PR_5\}$, $PU = \{PU_1, \ldots, PU_{10}\}$ and $GU = \{GU_1, GU_2\}$. According to the model in Sect. II, the access control relations are $PAC = \langle PR, PU, \{(PU_1, PR_1), (PU_2, PR_1), \ldots, (PU_{10}, PR_5)\}\rangle$, $UR = \langle GU, PU, \{(GU_1, PU_1), (GU_1, PU_3), \ldots, (GU_2, PU_5), (GU_2, PU_7), (GU_2, PU_9)\}\rangle$, $RR = \langle \{VR_1, VR_2\}, PR, \{(VR_1, PR_1), (VR_1, PR_2), (VR_2, PR_3), (VR_2, PR_4), (VR_2, PR_5)\}\rangle$ and therefore $GAC = \langle GU, \{VR_1, VR_2\}, \{(GU_1, VR_1), (GU_2, VR_2)\}\rangle$.

In this configuration, each GU can access a single VR only and, for instance, GU_2 is not allowed to access VR_1.

The main limitation of the approach manifests itself in the previous scenario when a new *permission assignment* is required at the Grid layer, e.g. the need for GU_2 to access VR_1. Since such a permission relies on low level mappings and assignment relations, it requires the extension of the sets UM

and PA. In detail, the addition of the pair (VR_1, GU_2) to GA can be achieved by

1. adding two users, say PU_{11} and PU_{12}, for accessing PR_1 and PR_2 respectively at the Physical layer; this also means that PAC must be also extended so that both $(PU_{11}, PR_1) \in PAC$ and $(PU_{12}, PR_2) \in PAC$;

2. adding the correspondence between GU_2 and the new PUs PU_{11} and PU_{12} to UM by the Middleware admin, i.e. UM must be extended so that both $(GU_2, PU_{11}) \in UM$ and $(GU_2, PU_{12}) \in UM$.

The previous operations are carried out by different administrators. The addition of PU_{11} and PU_{12} in the set of PU is made by the PO administrator (i.e. the *root* user) while the new permission assignments can be made by the administrator and the unprivileged user that possesses the resources (e.g. PU_1 for PR_1). At the middleware layer, the resources and the user mappings (2) are defined and changed by the Grid administrator, e.g. the *globus* user in GT4.
With reference to our model, the *root* user defines the rules in PAC, by adding and removing accounts on the machine (PUs) while the *globus* user defines the mappings in UM (e.g. by editing the `grid-mapfile`) and in RM (e.g. by properly configuring the MDS and the middleware).

III. Group-Based Access Control

Informally a group is a set of users sharing the same set of permissions on a set of resources. In Unix-like operating systems, each user can be member of one or more groups. Each resource (i.e. a file) has a single owner and is associated with a single group. Moreover each resource is associated with an access control list (ACL) stating which permissions are granted to *(i)* the owner, *(ii)* the members of the group and *(iii)* all other users. Only the owner of a resource (besides the system administrator) can modify the ACLs of the resources she owns.
The idea of dynamic groups we propose for the Grid layer has some similarity to that of groups used in UNIX-like operating systems but differs also in some important aspects. Dynamic Groups are managed at the Grid layer and built on the top of the Grid model we have previously defined.
As illustrated before, each GU is allowed to access a set of virtual resources. Our extended models allows GUs to create and delete groups as well as join and leave groups.
The addition of dynamic groups to the Grid layer can be modelled as follows.
Let $GAC = \langle GU, VR, GA \rangle$ be the access control policy at the Grid layer as defined in Sect. II. The *group-based access control policy at the Grid layer* is a tuple $GAC^+ = \langle GAC, Grp, GrpU, GrpAC \rangle$, where Grp is the set of *groups*, $GrpU \subseteq (Grp \times GU)$ is the *group membership relation* and $GrpAC \subseteq (Grp \times VR)$ is the *group permission assignment relation*. We say that *grid user gu is granted access to virtual resource vr according to GAC^+* if and only if *(i)* gu is granted access to vr according to GAC, i.e. $(gu, vr) \in GA$, or *(ii)* there exists $grp \in Grp$ such that $(grp, vr) \in GrAC$ and $(grp, gu) \in GrpU$.
The policy GAC^+ is defined on top of the policy GAC and can be modified by changing GAC or by means of a set of actions that support the creation or deletion of groups, the addition or removal of user from groups as well as the addition or removal of elements from the pool of shared resources.
The creation and destruction of dynamic groups, the admission and removal of users from groups, and the management of shared resources are regulated by security policies that define under which conditions these activities can be carried out. For instance, a GU belonging to a group can leave or be removed from the group.
In the first case, a group can be left voluntarily, for instance, when the GU has no more interest in accessing the shared resources; in the second case, the removal can be decided as a response to an abuse.
In both cases, the action is possible only if they comply with the security policies associated with the groups. We distinguish between meta policies and group policies.
A *meta policy* governs the creation and destruction of groups. Meta policies are valid for a set of groups (e.g. all groups in a VO) and define conditions that must be met to modify the GAC^+. For instance, examples of rules in a meta policy are: limiting the number of active groups in a VO (i.e. the maximal cardinality of Grp), allowing the destruction of a group only if there are no active members or stating that a user can delete only groups she created. Meta policies are defined once by the administrator of the VOs.
A *group policy* governs the group membership and regards users and resource shared within a group. A group policy defines conditions for entering and leaving the group as well as the conditions under which a VR can be shared within the group. The group policy of a group is independend from the groups policies of other groups. A group policy is defined by the some user (e.g. the user that created the group) but it does not override the meta policy. For instance, a group policy can require a GU to share access to resources in order to join the group or can eliminate a GU from the group as soon as she stops sharing permissions.
The definition of meta-policies, group policies, and the type of requests for admission, removal and leaving a group are out of the scope of this paper. Here we focus on operations that modify the GAC^+ and assume that both $MPol$ and $GrPol$ are are the meta policy and the group policy respectively.
We now present the primitive operations that allows for changes in the group-based access control policy at the Grid layer, distinguishing between *group management operations* (subject to $MPol$) and *group membership operations* (subject to $GrPol$). Let $GAC^+ = \langle GAC, GrpId, GrpU, GrpAC \rangle$ be a group-based access control policy. Let $gu \in GU$, then the group management operations are defined as follows:

- *Group creation.* The execution of the command $CreateGroup(grp)$ by gu on GAC^+ yields a new group-based access control policy $GAC_1^+ = \langle GAC, GrpId \cup \{grp\}, GrpU[\{gu\}/grpid], GrpAC \rangle$,[1] if $MPol$ is satisfied, otherwise $GAC_1^+ = GAC^+$.

- *Group deletion.* The execution of the command

[1] If $f : X \to Y$, then $f[y_0/x_0]$ denotes the function $f' : X \to Y$ such that $f'(x_0) = y_0$ and $f'(x) = f(x)$ for all $x \in X \setminus \{x_0\}$.

$DeleteGroup(grp)$ by gu against GAC^+ yields a new group-based access control policy $GAC_1^+ = \langle GAC, Grp \setminus \{grp\}, GrpU, GrpAC\rangle$, if all conditions in $MPol$ are satisfied, otherwise $GAC_1^+ = GAC^+$.

Group membership operations on users and resources are defined in the following way:

- *User addition to group.* Let $grp \in Grp$ and $\{gu, gu'\} \subseteq GU$. The execution of the command $AddUserToGroup(grp, gu)$ by gu' against GAC^+ yields a new group-based access control policy $GAC_1^+ = \langle GAC, Grp, GrpU[GUs/grpid], GrpAC\rangle$ where $GUs = GrpU(grp) \cup \{gu\}$, if the adding of gu satisfies the conditions in the $GrPol$ of grp, otherwise $GAC_1^+ = GAC^+$.
- *Resource addition to group.* Let $grp \in GrpId$, $gu \in GU$ and $vr \in VR$. The execution of the command $AddResourceToGroup(grp, vr)$ by gu against GAC^+ yields a new group-based access control policy $GAC_1^+ = \langle GAC, Grp, GrpU, GrpAC \cup \{(grp, vr)\}\rangle$, if gu is granted access to vr according to GAC and the vr addition satisfies conditions in $GrPol(grp)$, otherwise $GAC_1^+ = GAC^+$.
- *Resource removal from group* Let $grp \in GrpId$, $gu \in GU$ and $vr \in VR$. The execution of the command $RemoveResourceFromGroup(grp, vr)$ by gu against GAC^+ yields a new group-based access control policy $GAC_1^+ = \langle GAC, Grp, GrpU, GrpAC \setminus \{(grp, vr)\}\rangle$, if gu is granted access to vr according to GAC, $(grp, vr) \in GrpAC$) and the removal satisfies conditions in the corresponding $GrPol$, otherwise $GAC_1^+ = GAC^+$.
- *User removal/leaving from group.* Let $grpid \in Grp$ and $\{gu, gu'\} \subseteq GU$. The execution of the command $RemoveUserFromGroup(grp, gu)$ by gu' against GAC^+ yields a new group-based access control policy $GAC_1^+ = \langle GAC, Grp, GrpU \setminus \{(grp, gu)\}, GrpAC \setminus \{(grp, V)\}\rangle$ with $\forall(grp, V) \in GrACs.t.(gu, V) \in GA$ if the removal of gu satisfies the conditions in $GrPol(grpid)$, otherwise $GAC_1^+ = GAC^+$. If $gu = gu'$ then the operation corresponds to a voluntary leaving of the user from the group.

In Sect. II we discussed the difficulties associated with changing permissions so to allow GU_2 to access VR_1. With the extension to dynamic groups, the access to the resource can be obtained as follows:

1. GU_1 invokes $CreateGroup(Grp_{GU_1})$ to build a group. Grp_{GU_1} is a new group and the associated group policy is defined.
2. GU_1 invokes $AddResourceToGroup(Grp_{GU_1}, VR_1)$ to share access to VR_1 in the group.
3. GU_2 requests admission to the group and GU_1 invokes $AddUserToGroup(Grp_{GU_1}, GU_2)$ to add GU_2 to the group.
4. Similarly, once GU_2 asks to share the access to VR_2 within the group, GU_1 invokes $AddResourceToGroup(Grp_{GU_1}, VR_2)$ for adding the VR_2 to the pool of shared resources.

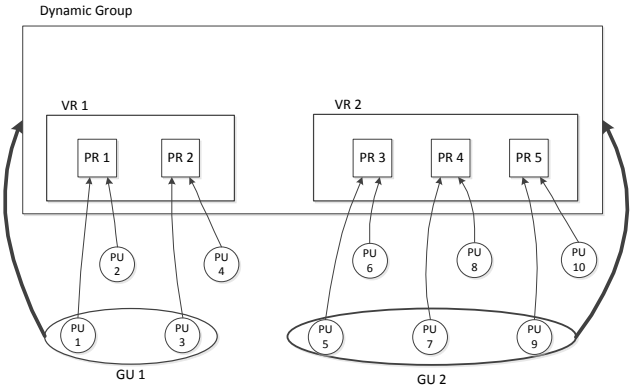


Figure. 2: A group-based access control policy

This leaves us with a group-based access control policy $GAC^+ = \langle GAC, \{Grp_{GU_1}\}, GrpU, GrpR\rangle$, where

- $GrpU(X) = \{GU_1, GU_2\}$ if $X = Grp_{GU_1}$, and $GrU(X) = \emptyset$ elsewhere.
- $GrpR(X) = \{VR_1, VR_2\}$ if $X = Grp_{GU_1}$, and $GrR(X) = \emptyset$ elsewhere.

which is depicted in Fig. 2.

IV. Implementing Group-based Access Control in GT4

The group-based access control model can be implemented as a service [6] in a Service Oriented Grid middleware as Globus Toolkit 4 (GT4). We do not consider pre-WS middleware since the SOA paradigm is by now supported by all state-of-the-art Grid platforms and it provides a higher level of generality and compatibility.
We show how group-based access control can be implemented as a Factory Grid Service using the standard technology of Grid Services in GT4 and show how GUs can exploit the functionalities of GT4 in order to join groups and sharing accesses.

A. Introducing GT4 Technologies

We briefly introduce here the characteristics of GT4 which are used in the proposed implementation.

1) Factory Grid Services and MDS

In GT4 a VR is defined as a Grid Service and it is uniquely identified with a URL. A Grid Service is an improved Web Service that is both stateful and transient. *Statefulness* means that actions made by GUs to the VR affect the VR, whereas *transiency* means that VRs can be created and destroyed on demand. The set of operations that GUs can invoke on a VR is the interface of the VR and is described in a Web Service Definition Language (WSDL) document [7]. VRs are published to the Monitoring and Discovery Service, which acts as an index service.

In a Grid, an index service is a proper grid service devoted to collect information on active Grid Service. In practice, each new built Grid Service provides information (i.e. the WSDL interface plus some additional information like the URL of the VR, its internal state, ...) to the index service. Thus, GUs can get information on the available Grid Services by querying an MDS Index service.
In GT4 there are no limitations on the way a Grid Service (and hence VRs) can be implemented and published on MDSs. Yet, it is often convenient to use some pre-defined Grid Service patterns. The Factory Grid Service suits our needs. A *Factory Grid Service* (FGS) is a meta-service that builds on-demand a service whose interface is specified in a WSDL document.
Interfaces of a FGS are very simple and support only operations for creating or deleting an instance of the given service. Each sub-service is created or deleted in response to the invocation of methods in the interface of the FGS. A pre-defined service in a FGS has a proper WSDL interface that is customized once the sub-service is built by the FGS. Thus, an URL is associated with it and it can be automatically registered to the MDS. Destruction of sub-services can be automatic (i.e. when the activity of the sub-service terminates) or it can be terminated by the creating user by invoking the method of the FGS.
The Globus Resource Allocation Manager service (GRAM) is a built-in FGS of GT4. It is basically composed of a FGS ($ManagedJobFactoryService$) that manages job executions on demand.
In order to execute a new job, the the use provides the GRAM service with a description of the job. The $ManagedJobFactoryService$ identifies the user, builds a proper instance ($ManagedJobService$) that corresponds to an execution environment, returns the URL of the instance to the user, and finally registers the new service to the MDS.
At this point, the any user can access the $ManageJobService$ instance autonomously through its own interface, as any other Grid Service. The Factory Service of the GT4 GRAM is depicted in Fig. 3.
Each FGS identifies any requesting GU and binds her identity to the instance created. The instance is initially accessible by the GU that has required the creation of the instance to the FGS.

2) Permissions in GT4

In GT4 each GU is associated with a public key certificate issued by a given Certificate Authority (CA). Without loss of generality we assume that each GU has an unique identifier included in her certificate.
The access of a GU to a VR is based on the use of temporary public key certificates called *proxies* [9]. Proxies are generated by GUs by using their own main certificates and are signed by the GUs themselves instead of the CA. For security reasons the life-time of proxies is considerably shorter than that of the main certificates. (The default life-time of proxies is set to 5 days in GT4.) Proxies are mainly used for delegation as any principal possessing a proxy is granted the same permissions as the GU who issued it.
All accesses in GT4 are made through valid proxies. For instance, in order to access a VR VR_y a GU GU_x builds and

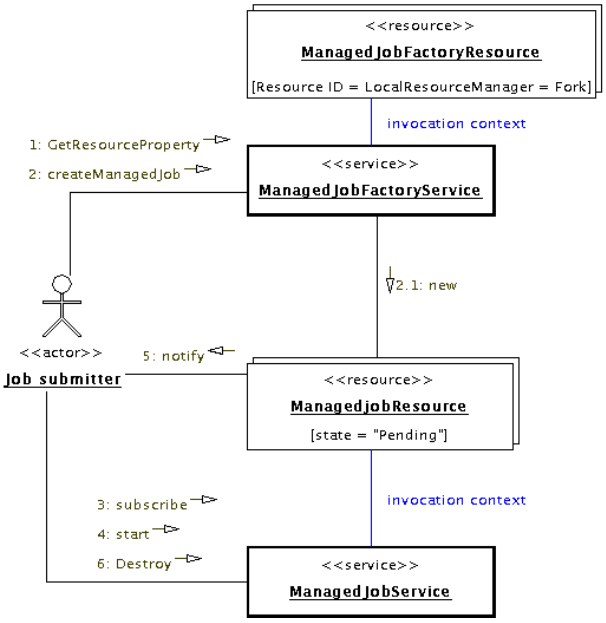


Figure. 3: The Factory Service of the GT4 GRAM

(self-)signs a new proxy, and uses it for accessing VR_y. In the following, we write $Px(GU_x)$ to denote a valid proxy generated by GU_x.
From a general perspective, the use of proxies characterizes GT4 as an identity-based authorization middleware, i.e. each VR keeps the list of the GUs that are authorized to access it. A proxy $Px(GU_x)$ identifies the user GU_x that signed it. Thus, any other user that possesses $Px(GU_x)$ is automatically authorized to access the same resources that GU_x is permitted to access.

B. Group-based Access Control as a FGS with Proxies

The previously analyzed characteristics of GT4 allow us to implement the group-based access control policy presented in Sect. III as a VR through the definition of a FGS, say GroupManagerService (GMS).
We assume that a GMS service is implemented in a VO supporting our Group-based access control model. Proxies provide an efficient way to share accesses to resources among group users.

1) Building GMS and Groups Services

At first, the GMS is built as a FGS in GT4 with an initial internal status containing the meta-policy, the WSDL interface and information describing the state of the service. All these data are considered public.
The meta policy is defined by the administrator of the VO, namely the *globus* user. The WSDL interface contains only two methods for creating and destroying groups that directly implements the $CreateGroup(info)$ and $DeleteGroup(grp)$ operations related to the GAC^+. Other information are related to the active groups.
The invocation of $CreateGroup(info)$ by a GU, yields a $DynamicGroup$ instance by the GMS. To this end, the user

must provide information on the characteristics of the group (e.g. the group policy) in the $info$ parameter. If the meta-policy is satisfied, then the $DynamicGroup$ Service is created and the URL of the group service is returned to the user and registered to both the MDS.
The URLs of the active $DynamicGroup$ instances are also stored in the GMS. Thus, other GUs can discover active $DynamicGroups$ services by querying the GMS and the MDS. The information provided by these services on each $DynamicGroup$ includes the group policy, the list of participating users, and the shared resources.
The $DynamicGroup$ interface contains the four methods we defined in Sect. III (namely $AddUserToGroup(grp, gu)$, $AddResourceToGroup(grp, vr)$, $RemoveResourceFromGroup(grp, vr)$, and $RemoveUserFromGroup(grp, gu)$ and a few extra methods for changing the group policy and permissions exchange that we discuss below. Note that GUs are identified by their proxies and VRs are identified by their URLs.
The removal of a group is made through the $DeleteGroup(grp)$ method of the GMS, where grp is the URL of a $DynamicGroup$ instance. After the verification of the meta policy, a successful removal destroys the $DynamicGroup$ instance and deletes the associated information from the GMS as well as the subscriptions from the MDS. All the proxies kept in the group are similarly destroyed. The management of $DynamicGroups$ by the GMS is depicted in Fig. 4.

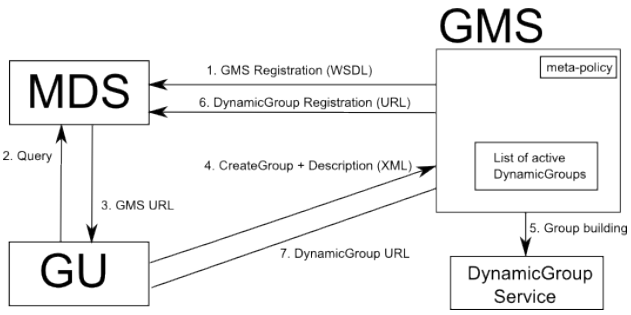


Figure. 4: The GMS and dynamic groups

We notice that unlike an FGS like GRAM, which requires non-negligible resources in term of computational power, memory and disk space for executing jobs, the GMS acts only as an information service (customized for managing information on dynamic groups) of GT4. Single $DynamicGroup$ instances acts in the same way, as index services.
Thus, the GMS and all $DynamicGroup$ instances exploit the resources allocated for the GT4 middleware, like the MDS, and they do not requires specific resources that are external to the middleware.
In particular, wrt the basic model in Section II, this means that at the Physical layer, no new PRs are required for supporting GMS and $DynamicGroup$.

2) Group Services and Permissions

In order to share the access to resources in an effective way, the operations related to the removal and addition of users in a group are implemented by the passing or removal of proxies. In particular, the $AddUserToGroup(grp, gu)$ stores a proxy of gu in the $DynamicGroup$ service instance.
Subsequent invocations of $AddResourceToGroup(grp, vr)$ made by gu, build an *association list* of resources accessible through the proxy of gu (i.e. $Px(gu)$). Dually, invocations of $RemoveResourceFromGroup(grp, vr)$ eliminate elements from this list.
Thus, in order to access a resource shared in a dynamic group, a user gu' must acquire from the group a proxy generated by the user who own the resource. To this end, an explicit method $AskPermission(grpid, vr)$ is included in the $DynamicGroup$ interface that governs the provision of permissions to legal group users.
The invocation of this method by a gu' allows the user to ask a proxy from the group identified by $grpid$ for accessing the resource vr. As a result of this invocation, the $DynamicGroup$ instance checks the association lists that contain vr and provides back to the user a suitable proxy.
Once gu' gets back a proxy $Px(gu)$, she uses it to access vr in gu's stead.
It must be noted that since proxies have a relatively short validity period, GUs have to regularly provide a new proxy through the $AddUserToGroup$ operation as soon as the old expires.
The use of proxies can thus limit the usability of dynamic groups, since the provisioning of fresh and up-to-date proxies to the group can be cumbersome for the GU.
However, since the management of proxies is common and widely recognized problem in Grid environments, some proposals have already been put forward for automating the generation and provisioning of proxies. For instance, in [10] a proxy renewal service is presented. It automatically renews expired proxies following guidelines and constraints provided by the GU. Moreover, in [11] a framework for proxy revocation is defined. It allows to automatically revoke proxies in GU's stead.
The choice between the manual provisioning and the automatic renewal of proxy is related with the level of control the GU requires on permission shared of the group and it is up to the strategies and requirements of the single GU.
From a group point of view, the provision of an invalid proxy as well as the omission of a renewal could correspond to a temporary violation of the group policy.
In these cases GUs can be *revoked* for a while and readmitted as soon as the group policy is satisfied again. During the revocation period, the GU still belongs to the group (i.e. its permissions are still shared) but it is banned from the use of other group permissions.

3) Using Groups

We show how the proposed implementation can support an effective exploitation of resources shared through groups, by using again the scenario in Sect. III as an example, but assume that GU_1 wants to build a group and that she has discovered the existence of a GMS through the MDS. GU_2 can be given access to VR_1 by GU_2 in the following way (see Fig. 5):

1. GU_1 queries the GMS for information and obtains the meta policy.

2. After evaluating the meta policy, GU_1 builds a group policy where it requires, for instance, that any other user in her groups must share the access to at least a resource in order to exploit the shared resources. Thus, GU_1 invokes the $CreateGroup(info)$, providing the group policy; let suppose that the request satisfies the conditions in the meta policy, thus a new $DynamicGroup$ (DG_1) is built and registered to GMS and MDS with URL=url_{GU_1}.

3. GU_1 signs a proxy $Px(GU_1)$ and invokes the method $AddUserToGroup(url_{GU_1}, Px(GU_1))$ on the newly-created DG_1 service to add its identity to the group.

4. GU_1 invokes $AddResourceToGroup(url_{GU_1}, VR_1)$ on DG_1, adding the sharing of VR_1 to the group. DG_1 adds VR_1 to the list of resources associated with $Px(GU_1)$.

5. GU_2 queries the MDS in order to discover dynamic groups in which the access to VR_1 is shared and she discovers DG_1.

6. GU_2 queries DG_1 for information and gets the group policy.

7. GU_2 evaluates conditions in the group policy and then chooses to share the access to VR_2 in order to gain access to VR_1. Consequently, GU_2 signs a proxy $Px(GU_2)$ and invokes $AddUserToGroup(url_{GU_1}, Px(GU_2))$ on DG_1.

8. Then, GU_2 invokes $AddResourceToGroup(url_{GU_1}, VR_2)$ on DG_1. DG_1 associates VR_2 to $Px(GU_2)$.

9. GU_2 invokes $AskPermission(url_{GU_1}, VR_1)$ on DG_1. Since GU_2 meets the conditions in the group policy of DG_1, $Px(GU_1)$ is returned to GU_2.

10. GU_2 accesses and exploits VR_1 through $Px(GU_1)$.

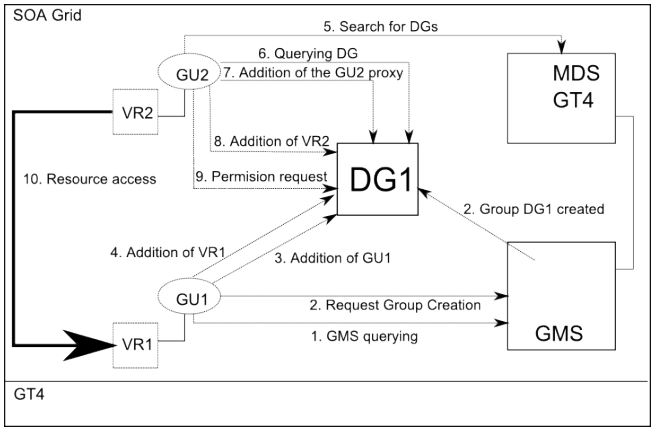


Figure. 5: Building and using of dynamic groups in GT4.

V. Related Work

The research on Grid access control has been carried out in two complementary directions, namely (1) *the porting of existing access control models* to the Grid paradigm and (2) the definition of solutions for supporting *the management of the AC models on Grid middleware*, independently from the single model and the definition of rules.

Regarding the first point, the most interesting proposals regard the porting of Role-Based Access Control model on Grid. In [18], the RBAC model is ported to a generic, pre-WS Grid through a properly defined architecture, designed for working over the middleware. Roles can be defined on any Grid entity and a simple language to define rules and role acquisition is provided.

Differently from the previous proposal, in [19] a fully OGSA-compliant porting of the RBAC model on a Service Oriented Grid is provided. The proposed approach is based on the integration of RBAC using Shibboleth as authorization service in the OGSA-DAI framework, resulting in a reduction of the overhead due to the native[2] mapping of Grid users on physical resource permissions.

Among RBAC, other access control models (not natively embedded and supported in Grid middlewares) have been ported to Grid through proper over-middleware frameworks. For instance, the Attribute-Based Access Control (ABAC) model have been proposed in [20]. In this approach, a framework for managing ABAC on GT4 is provided. This approach simplifies the management of access control rules for Grid administrators, by granting a flexible and scalable methodology for the definition and modification of rules.

Beyond dynamism of rules, another issue in the native Grid access control system is the difficult in defining fine-grained access control for specific resources or users. To this aim, in [21] a language and a framework for supporting fine-grained definition of access control policies is proposed. Through such extension it is possible to define security constraints related to access control.

Independently from the access control model, solutions for managing access control on Grid can be broadly classified in three categories: (1) *access rules definition tools*, aimed at simplifying the definition of access rules at administrative level; (2) *authorization services* focused on provisioning signed certificates to users with their attributes and permissions; (3) *decision-making services* devoted to determine if a given user, provided with credentials, can access a resource; the decision is generally made by considering both the user attributes and the resource policies.

The Prima system [12] falls in the first category. It provides tools to the Grid administrator (i.e. the Globus user) that simplify the access control management.

The Community Authorization Service (CAS) [13] is an authorization service that supports authorization at the PO layer (i.e. Physical Layer) and at VO site (i.e. Middleware layer). PERMIS is another example of cross-layering authorization infrastructure: rules at the Physical Layer are stored in LDAP servers and defined by PO admins. It supports RBAC and other group-based policies, but they are defined by a central authority and cannot be flexibly changed by GUs. Dif-

[2] in the OGSA-DAI framework

ferently from CAS and PERMIS, Cardea [14] is a framework for managing authorizations coming from administration systems of different domains. The virtualization over the authorization systems provided by Cardea grants a more dynamic management of authorization, but still based on single administrative permission assignment.
Akenti [15] provides an access control decision function that grants/denies access by taking into account both the rules defined by the resource administrator and the user identity; it provides signed capabilities to authorized Grid users.
The EU-DataGrid Security Infrastructure [16] constitutes a complete solution, implementing services for supporting the definition of access rules, an authorization authority (VOMS) and decision functions operating on the available certificates.
Existing proposals therefore enjoy interesting features but they also share a common drawback: changes in the access control policy require the involvement of the administrators. Our group-based access control model allows Grid users to exchange access permissions to VRs without the intervention administrators.

VI. Conclusions

We have presented a group-based access control model for Grid environments that allows users to share access permissions to VRs without the intervention administrators. Our approach increases the flexibility of the access control model offered by state-of-the-art Grid platforms, without requiring changes in the middleware. We have shown that our solution can be defined on top of the access control mechanisms offered by state-of-the-art Grid middleware and that it can be implemented as a service in a service-oriented Grid environment.
The proposed approach is suitable in trust environments like scientific Grids, where each GU has a trust relationship with the others in the VO, or in environments where the exploitation of resource does not involve high-security requirements applications (e.g. personal information, financial transactions, and so on).
To this regard, the future work will be focused on defining security solutions that allow the secure use of the proposed model on insecure Grids.

References

[1] I. Foster and K. Kesselmann. The Grid 2: Blueprint for a New Computing Infrastructure. Elsevier, 2003. (book style)

[2] I. Foster. Globus toolkit version 4: Software for service oriented systems. Journal of Computer Science and Technology, 21-4:513520, July 2006. (journal style)

[3] gLite Middleware. http://glite.web.cern.ch/glite/.

[4] J. Dongarra. Urgent Computing: Exploring Supercomputings New Role, volume 4, 1. Cyberinfrastructure Technology Watch, P. Beckham, Ed., March 2008. (book style)

[5] S. Gorlatch, A. Ploss F. Glinka, J. Muller-Iden, R. Prodan, V. Nae, and T. Fahringer. Enhancing grids for massively multiplayer online computer games. Technical Report CoreGRID Technical Report - TR-0134, June 17, 2008. (technical report style)

[6] D. Talia C. Comito and P. Trunfio. Grid services: principles, implementations and use. International Journal of Web and Grid Services, 1, 1:48 68, 2005. (journal style)

[7] W3C. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. http://www.w3.org/TR/wsdl20/

[8] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. 2002. (conference style)

[9] N. V. Kanaskar, U. Topaloglu, and C. Bayrak. Globus security model for grid environment. SIGSOFT Softw. Eng. Notes, 30(6):1 9, 2005. (conference style)

[10] D. Kouril. A credential renewal service for long-running jobs. In GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing, pages 6368, Washington, DC, USA, 2005. IEEE Computer Society. (conference style)

[11] S. Zhao, A. Aggarwal, and R. D. Kent. A framework for revocation of proxy certificates in a grid. In SNPD '07: Proceedings of the Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, pages 532-537, Washington, DC, USA, 2007. IEEE Computer Society. (conference style)

[12] M. Lorch and D. G. Kafura. Supporting secure ad- hoc user collaboration in grid environments. In GRID 02: Proceedings of the Third International Workshop on Grid Computing, pages 181-193, London, UK, 2002. Springer- Verlag. (conference style)

[13] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A community authorization service for group collaboration. In POLICY02: Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY02), page 50, Washington, DC, USA, 2002. IEEE Computer Society. (conference style)

[14] R. Lepro. Cardea: Dynamic access control in distributed systems. Technical report, 2003. (technical report style)

[15] M. Thompson, A. Essiari, and S. Mudumbai. Certificate- based authorization policy in a pki environment. ACM Trans. Inf. Syst. Secur., 6(4):566 588, 2003. (journal style)

[16] R. Alfieri, R. Cecchini, V. Ciaschini, A. Gianoli, F. Spataro, F. Bonnassieux, P. Broadfoot, G. Lowe, L. Cornwall, J. Jensen, D. Kelsey, . Frohner, D. L. Groep, W. Som De Cerff, M. Steenbakkers, G. Venekamp, D. Kouril, A. Mcnab, O. Mulmo, M. Sil, and J. Hahkala. Managing dynamic user communities in a grid of autonomous resources. In CHEP 2003, La Jolla, pages 24 28, 2003. (conference style)

[17] Y. Wen, C. Chen, S. Wang. Crossing Heterogeneous Grid Systems with a Single Sign-On Scheme Based on a P2P Layer, Asia-Pacific Services Computing Conference, 2008. APSCC '08. IEEE , pp.45-51, 9-12 Dec. 2008 (conference style)

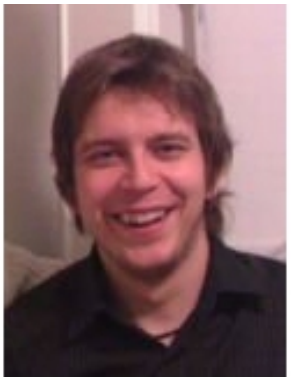
[18] W. Qiang, J. Hai, S. Xuanhua, Z. Deqing and H. Zhang. RB-GACA: A RBAC Based Grid Access Control Architecture, in Grid and Cooperative Computing, LNCS, pp 487-494, 2004 Springer Berlin / Heidelberg. (book chapter style)

[19] V. Muppavarapu, A. Pereira, and S. Chung. Role-based access control for a Grid system using OGSA-DAI and Shibboleth, in The Journal of Supercomputing, Vol. 54, Issue 2, pp. 154-179, Springer Netherlands. (journal style)

[20] B. Lang, I. Foster, F. Siebenlist, R. Ananthakrishnan, and T. Freeman. A Flexible Attribute Based Access Control Method for Grid Computing, in Journal of Grid Computing. (journal Style)

[21] J. Wu, C. B. Leangsuksun, V. Rampure, and H. Ong. Policy-Based Access Control Framework for Grid Computing. In Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID '06). IEEE Computer Society, Washington, DC, USA, 391-394. (conference style)

Author Biography



Alessio Merlo received his Ph.D. in Computer Science from University of Genoa (Italy) where he worked on performance and access control issues related to Grid Computing. He is currently serving as a research scholar at University of Genoa, Department of Engineering, where he is working on a FP7-funded project concerning automatic security testing of Web applications. His research interests are focused on performance and security issues related to Web and distributed systems.