CERIAS Tech Report 2004-70

TOWARDS SUPPORTING FINE-GRAINED ACCESS CONTROL FOR GRID RESOURCES

by E. Bertino, P.Mazzoleni. B.Crispo, S.Sivasubramanian, E.Ferrari

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

Towards supporting fine-grained access control for Grid Resources

Elisa Bertino, Pietro Mazzoleni
Dipartimento di Informatica e Comunicazione
University of Milan, Italy
bertino,mazzolen@dico.unimi.it

Bruno Crispo, Swaminathan Sivasubramanian
Computer Science Department
Vrije Universiteit
crispo,swami@cs.vu.nl

Elena Ferrari
Dipartimento di SSCCFFMM
University of Insubria, Italy
elena.ferrari@uninsubria.it

Abstract

The heterogeneous nature and independent administration of geographically dispersed resources in Grid, demand the need for access control using fine-grained policies. In this paper, we investigate the problem of fine-grained access control in the context of resource allocation in Grid, as we believe it is the first and key step in developing access control methods specifically tailored for Grid systems. To perform this access control, we design a security component (to be part of a meta-scheduler service) that finds the list of nodes where a user is authorized to run his/her jobs. The security component is designed in an effort to reduce the number of rules that need to be evaluated for each user request.

We believe such a fine-grained policy-based access control would help the adoption of Grid to a higher extent into new avenues such as Desktop Grids, as the resource owners are given higher flexibility in controlling access to their resources. Similarly, Grid users get a higher flexibility in choosing the resources in which their jobs must execute.

1 Introduction

A Grid provides an abstraction for resource sharing and collaboration across multiple administrative domains. Resources in this environment span across multiple geographical locations, usually heterogeneous in nature and administered individually by different resource owners. Such an environment requires the possibility of different access control policies for each resource, instead of one global policy uniformly applied to all Grid resources.

In this paper, we are interested in the problem of fine-grained access control (FGAC) in the context of job scheduling, as we believe this is the *first and key step* in resource access control. Grid in its current form, implements security as a separate subsystem [5] and job scheduling/resource allocation is executed oblivious of security implications. Such an allocation process can lead to allocation of resources which the user is not authorized to access. This would introduce the problem of re-submitting the request until the meta-scheduler finds a set of resources that will authorize the user's request.

We propose a way to integrate fine-grained access control in the resource allocation process, so that the scheduler *allocates only those resources* that the user is authorized to access. To accomplish this, as a first step, we propose a security component (part of meta-scheduler) that finds the resources that authorize a user's request by evaluating the policies associated with each resource.

In our system, resource owners specify the access control policy for the resources using a well defined access control language (like XACML [10] and SAML [2]). Similarly, the Grid users specify their identity and security constraints in the same language. We view resource allocation as a two-stage process consisting of: (i) matching the access control policy of resources with a user's request and (ii) the resource scheduling - finding the best set of resources for a given user's request.

In this paper, we treat the resource scheduler as a black-box and discuss in detail the process of finding the set of authorized nodes that authorize a user's request and the interaction between our security component and the scheduler.

The contributions of this paper are follows: (i) We propose a Grid resource management architecture supporting fine-grained access control, where the security component is integrated with the resource broker and (ii) propose a novel method of organizing nodes' access control policies

Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'04)
0-7695-2118-5/04 $20.00 © 2004 IEEE

and user identities in an effort to reduce the number of rules evaluated for each user request.

The rest of the paper is organized as follows: Section 2 presents related work. Section 3 presents the architecture of our system model. Section 4 introduces the basics of access control policies, while Section 5 discusses the architecture of our security component. Section 6 and 7 discuss the node/user subscription and user job submission phases respectively. Section 8 concludes the paper.

2 Related Work

Usually, the process of resource access control in Grid is simplified using the concept of *Virtual Organization* (VO) [4]. VOs consists of users (possibly geographically dispersed) working towards a common set of problems. The VO defines who its members are, assigns roles or credential to members, and also regulates the access to the resources. Users only have the choice to accept or reject the set of access rules fixed by the VO.

We believe that a greater flexibility can be achieved, if in addition to the global access policy set by the VO, individual resource providers were also allowed to refine the access control policies for their resources, by means of their own *local policies* (LP). This approach is motivated by our assumption that there might be cases when resource owners are not willing to share or lease the ownership of their resources but they rather prefer to *choose* the policy according to in which their resource can be shared.

For the past several years, a great amount of research has been devoted to the problem of resource scheduling in distributed environments and in Grid [6]. Systems such as Condor [8], uses *class-ads*, to express resource attributes and perform matches between job requests and available resources. While the primary focus of these approaches has been on developing resource management algorithms, our focus is on building a security component that performs match-making between grid user requests and FGAC policies of resources in the context of job scheduling.

The problem of policy-based resource matching is addressed in [7] where the authors present a framework for matching the policies of resources with that of the user request to find a suitable set of resources which a user can run his job. The system finds the set of resources that matches the user's request by matching the user's classads with that of resources. To expedite the process of evaluation, they adopt an approach of grouping nodes with *same* policies into clusters (or gangs). Thus, the matchmaker performs matching at cluster-level instead of node-level. This improves the performance of the matchmaker when it has to find a set of resources for a user. However, the approach of aggregating nodes with same policies is well suited only for match-making of resource attributes. Using such a clustering scheme for FGAC policies can result in a large number of clusters each having only one node, as it is unlikely to find resources with same security policies in Grid supporting FGAC.

Recently, [1] proposed an algorithm for integrating the generic concept of trust into schedulers. The algorithm assumes that security competence (or node trustability) can be abstracted into a numerical value. The scheduler uses this value along with scheduling-related parameters such as completion time, execution time, as parameters of a single cost function, it aims to minimize. Then the scheduler picks the set of nodes that yields the best cost. However, such a scheme can pick a node with better resource characteristics (e.g., faster machine) compared to nodes with better trustability. Modelling security constraints according to such approach allows the scheduler to trade resource quality for poor security and vice versa, which is not acceptable for the resource providers and grid users. In our system, we take a different approach where security constraints of the nodes and users are expressed as policy rules instead of numerical values and the scheduler finds the set of nodes that authorize and match a user's job requests.

The concept of fine-grained authorization for resources is also analyzed in [9], where the Akenti tool is used to perform the access control. Their work focusses on the policy evaluation of nodes that have already been selected by the scheduler. Unfortunately, performing policy authorization using a separate system (different from the scheduler) to determine whether the resources selected by the scheduler is accessible or not, can lead to scheduling of resources the user is not authorized to access.

3 System Model

In this paper, we assume that the Grid consists of a large number of Grid nodes spread across the Internet. Each Grid node (resource) is assumed to have an owner responsible for its administration. Each Grid node can participate in more than one Grid projects and each project has Grid users with large number of job requests. Examples of such Grid include Desktop Grids that serve a computing platform for more than one projects.

For running a job, a Grid user (participating in a Grid project) submits a job request describing its requirements, along with his identity profile, to the scheduler. The security component of the scheduler matches the policy of the resources with the job request to see if the nodes authorize user's request or not. If so, the security component passes the set of authorized nodes to the scheduler. Subsequently, the scheduler selects the best set of nodes from this subset based on the resource constraints set by the user and resource availability.

4 Authorization Policies

Authorization Policies, or *Policy Rules*, are sets of logical expressions, expressing the constraints defined by a resource owner to restrict access to certain information (i.e., resources) for certain users (or requests).

By abstracting the syntactic details of the selected policy language, each policy rule can be seen as a composition of logical expressions representing the three basic elements of a rule: *Target*, *Effect*, and *Condition*. *Target* denotes the set of access requests a policy is intended to evaluate. It includes a set of services (*resources*) (e.g., computational power), the entity (*subject*) who issues the request (e.g., users from university of Milan), and the type of requested jobs (*actions*) (e.g., execute a script). *Condition* is a set of expressions (built by combining a set of predicates) which further specify when a policy applies (e.g., requests coming between 8am and 6pm). Finally, *Effect* represents the action (i.e., deny or permit) resulting from the request when the policy rule applies.

With respect to the policy language that can be used to define rules, several alternatives have been proposed so far as standards. As an example, XML-based languages such as XACML, SAML and Author-X [10, 2, 3], are rich in semantics and very flexible, thus allowing user to specify very complex access control rules. However, the solution proposed in this paper does not rely on a specific formalism to express policies but it aims to be applicable over policies specified in any available language. Therefore, in the following, we do not refer to any of the above; instead, we consider a policy rule being defined as a combination of one or more logical expressions, called *policy expressions*, built using one of more attributes (*policy attributes*) among the ones available to define constraints. In other words, we assume available a comprehensive set of policy attributes (e.g., Subject.Name, Subject.Affiliation, Resource.Type, Resource.Amount, Condition.Time) a resource owner can select to create authorization constraints (e.g., User.Name="IBM" AND Resource.Amount<300) which apply to requests issued against to his/her shared resources (or against all resources shared in a project in case of Global policies).

To understand how global and local policies can be formalized, consider as an example a simple GRID where resource owners share only one type of resource *X* (e.g., computing resources) with users belonging to a Grid project, called MyGrid. Suppose moreover that the MyGrid administrator specifies a Global Policy stating that the resources shared among the project can be accessed by users only between 5.00pm and 9.00am. According to the above formalism, such policy can be modelled by the logical expression: *MyGrid_policy=(Subject.Project="MyGrid" AND Condition.Time=5.00pm...9.00am)*.

Within the MyGrid project, we assume three resource owners: *Al, Ben,* and *Carl*, who are willing to join the MyGrid project, but who are not completely satisfied with the MyGrid Global Policy. According to our approach, resource owners are given the possibility of refining GP (using only AND operations) by specifying an additional set of rules which apply only to the resources they share within the network. Assume the following scenario:

1. *Al* does not specify further constraints.
2. *Ben* refines the global policy by specifying in his local policy base that users from IBM are authorized to access his shared resources only if the request (Request.Amount) is lower than 400 resource-units. Formally, *Ben_policy=My_Grid_policy AND (User.Affiliation=IBM AND Request.Amount<400)*;
3. *Carl* adds in his local policy base a policy stating that HP users can access his shared resources only between 8pm and 8am Formally, *Carl_policy=My_Grid_policy AND (User.Affiliation=HP AND Request.Amount<400)*

Since policies state constraints on user properties, the user must submit the information necessary to evaluate policies when issuing an access request. We call *user-profile* the set of attribute name/value pairs needed to perform policy evaluation. If the information in a user-profile validates the Global Policies as well as the Local Policies defined by a resource owner, the request is authorized, otherwise it is rejected. Based on the nature of the information collected into a user profile and the type of requests issued by the user, we distinguish:

- *Static profiles*: They contain attributes whose values do not change for the requests issued by the user. Examples of such attributes might be User.Name, User.Affiliation, Request.Resource_type and so forth.
- *Dynamic profiles*: They contain attributes whose values may change for each request (e.g., Condition.Time, User.IP_Connect, Resource.Amount, etc.)

5 Security System Architecture

In this section, we present the architecture of our security component designed to reduce the number of rules that need to be evaluated for each user requests to identify the compliant set of resources which authorize the job execution.

5.1 Design rationale

In designing the architecture of our security component, we took as starting point the following observations:

1. VOs organize resource owners by common goals, such as for instance the type of resource shared or the Grid project they participate, assuming similar policies for each member of the VO. In a FGAC Grid, VOs organize nodes by common authorization policies. However, if the number of resources owners with different local policies increases, the system has to deal with a huge number of different VOs (each with only one member) losing the advantages gained by such organization. Therefore, we need to support a flexible solution with two levels of VOs where the first level represents Global policies, and the second level represent the local policies specified by each single node[1]. Moreover, to avoid explosion of VOs, the system should group nodes using same policy attributes into the next level VO. For example, local policies $lp_1 = (Time = 8pm \ldots 8am)$ and $lp_2 = (Time = 10pm \ldots 12am)$ should be stored together into the same second-level VO (i.e., the one grouping LPs defined using the policy attribute $Time$) instead of being stored separately. The basic intuition behind grouping nodes using attributes instead of grouping nodes with same policies are the following: (i) it might be difficult to find two nodes having the same LP and (ii) attribute-based grouping helps pruning evaluations thereby decreasing number of evaluations performed (explained in the 3^{rd} point).

2. The rule evaluation process for policies based on static information should be executed *only once*. So far, a user submitting requests with information only from his/her static user profile, always requires an exhaustive rule evaluation process. A more efficient solution should cache some rule evaluation results the first time they are obtained and reuse them for all the subsequent requests issued by the same user. We propose the use of tables storing the set of nodes that authorize a user's request just based on his static profile. This process will reduce the turn-around time of scheduler.

3. The rule evaluation process of policies based on dynamic profile information must be executed for each user request. Unlike the static case, it is not possible to cache some rule evaluation results as in this case values of policy attributes may change with each request. In this case, we structure policies into chains of policy expressions and we propose a *pruning* technique that evaluates a policy expression only if all the policy expressions already evaluated for the same rule, have authorized the request. For instance, given the local policy $lp_3 = (User.Affiliation =' IBM' And Time = 8pm \ldots 8am)$, the evaluation of the policy expression $Time = 8pm \ldots 8am$ will be restricted to requests whose user Affiliation is 'IBM'.

[1]This two-level policy hierarchy can be easily extended to any number of levels, if required. However, for the sake of simplicity we consider only a two-level policy hierarchy in the rest of the paper

5.2 Secure Virtual Organizations

The cornerstone of our solution is the concept of *Secure Virtual Organization*, *SVO*, used to organize GRID nodes regulated by the same access control policies. Specifically, we assume that Grid nodes can be grouped into three types of SVOs: Project Secure Virtual Organizations (*PSVOs*), Static Secure Virtual Organizations (*SSVOs*), and Dynamic Secure Virtual Organizations (*DSVOs*), defined as follows.

A *Project Virtual Organization* groups nodes sharing the same global policies enforced by the VO administrators. Each Grid node can participate to more than one Grid project; therefore the same node will be grouped into more than one PSVO. Nodes that refine global policies by adding their own set of rules can be further grouped into SSVOs and DSVOs.

A *Static Secure Virtual Organization* groups all the nodes such that their local policy rules comprise a common attribute a_i. Thus, each rule is decomposed into a set of *policy expressions*, each one separately stored into the corresponding SSVO. For the sake of simplicity, in this work we do not consider any hierarchical organization of policy attributes, instead we assume one SSVO statically generated for each policy attribute to which a rule refers. Finally, a *Dynamic Secure Virtual Organization* groups nodes of a SSVO whose associated policies contain the same policy expression p_i. For instance, given a SSVO grouping nodes whose policies contain the attribute User.Affiliation, possible DSVOs for that SSVO could group nodes having User.Affiliation=IBM, User.Affiliation≠ HP or User.Affiliation=All_users as policy expressions. Unlike SSVOs, DSVOs are dynamically built by the system according to the LPs stored in each SSVO. We assume a new DSVO is dynamically created when the number of nodes referring to the same policy expression p_i in a SSVO goes beyond a defined threshold. Consider again the scenario introduced in Section 5.2. According to such taxonomy, the Grid consists of one PSVO (i.e., MyGrid project) three SSVOs, each one referring to a different policy attribute (i.e., User.Affiliation, Request.Amount, and Condition.Time), and by a set of DSVOs whose number (and distinguishing expressions) results from the occurrence of the same policy expressions into a SVO. Figure 1 shows a graphical representation of the SVOs (static and dynamic) generated in our scenario. Notice that Figure 1 has been provided as a reference for the rest of the paper; we do not expect all the components in the figure be completely clear at this point.

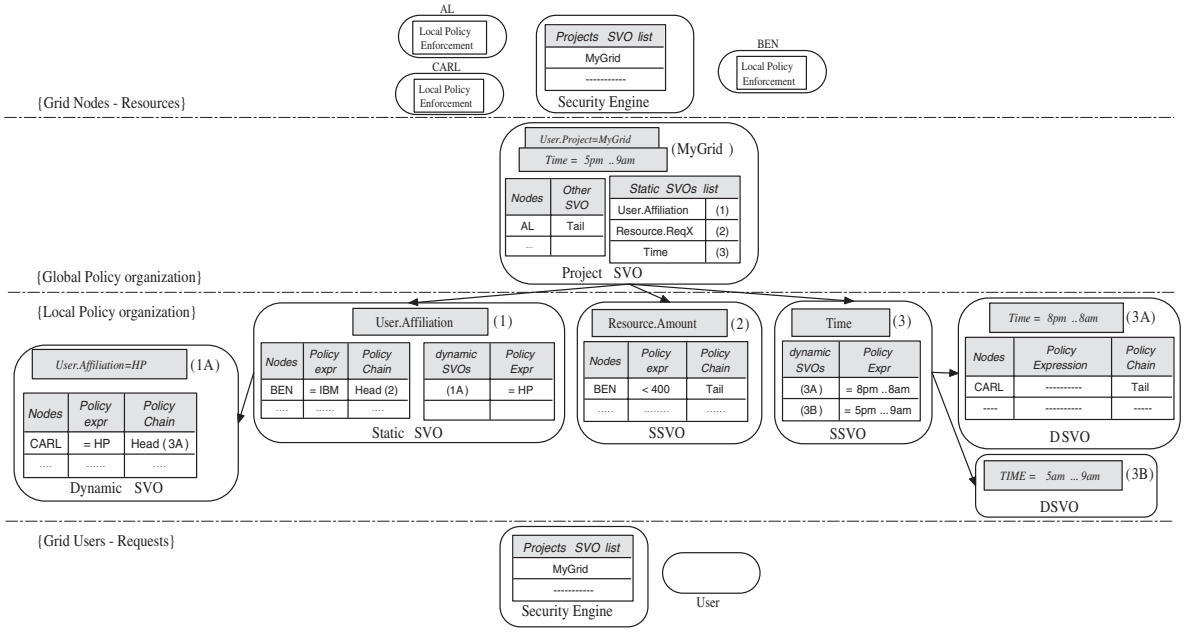

Figure 1. SVO organization example

5.3 System Components

In this section, we describe the overall architecture of the proposed security component, which support scheduling of computations in a FGAC (Fine-Grained Access Control) Grid computing network. The architecture of our security system is given in Figure 2.

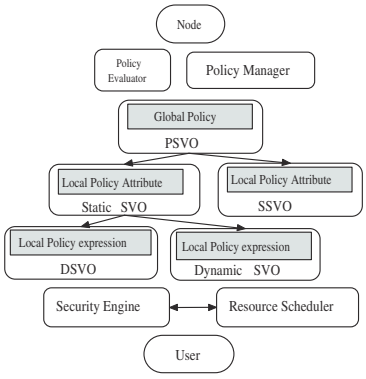

Figure 2. System architecture

In addition to the hierarchical organization of nodes among SVOs, the other important components of our system architecture are the following:

- *Node* - It is the subject who makes his/her resources available to the Grid Community.
- *Policy Manager* - It manages the association of new nodes and local policy updates.
- *Policy Evaluator* - It is the evaluation engine for the logical expressions stored into a SVO.
- *Security Engine* - It deals with new users joining the Grid as well as resources requests issued by registered users.

The methodology used to identify the nodes which are willing to authorize a job request consists of two main phases: the *Subscription phase* and the *Job-submission phase*.

1. *Subscription Phase*: The goal of this phase is to set up the system the first time a user or a node registers to the Grid. Specifically, when a node joins the grid, the Policy Manager collects policy rules and decomposes them into single policy expressions which are then separately organized into the SVOs. Similarly, when a user registers to the Grid, the Security Engine collects information about his/her profile and uses it to make a preliminary evaluation of the policy rules.
2. *Job-submission phase*: This phase is executed for each job request issued to the Grid. It takes as input the user profile and returns the list of nodes which are willing to authorize the execution of the request.

6 Subscription Phase

The *subscription phase* is executed the first time a new user or node registers to the Grid (i.e., by downloading the

software for resource sharing or by asking for a personal login to run jobs within the Grid). This phase deals with organizing policies rules (global and local) within the SVO structure presented in Section 5.2. Moreover, this phase performs a pre-evaluation of the policy expressions in order to reduce, as much as possible, the number of rules to be evaluated at run-time by the system for each future requests issued by a specific user. Subscription is organized into two separate subphases: *Node subscription phase* and *User subscription phase*.

6.1 Node subscription phase

When a node expresses its intention to share resources among the Grid, it must subscribe to the Policy Manager for specifying its LPs other than the project(s) it aims to participate. In brief, the Policy Manager collects node LPs, evaluates their consistency with global policies enforced by the VO to which the resource belongs, decomposes each policy rule into policy expressions and distributes them into the SVOs they belong according to the criteria illustrated in the previous section. For instance, the *Ben_policy* introduced in Section 4, will be decomposed into two policy expressions p_1=*User.Affiliation='IBM'* and p_2=*Resource.Amount<400* which are stored into the SSVOs identified by User.Affiliation and Resource.Amount attribute respectively.

Furthermore, we assume policy expressions composing a rule and stored into different SVOs, to be chained according to an order $\prec$ defined upon SSVOs distinguishing attributes. We call this structure as *Node_policy_chain* and consider $\prec$ to be a simple lexographical order. As example, Ben_policy_chain links the policy expression p_1, stored into User.Affiliation SSV, to the policy expression p_2 given that 'Resource.Amount' follows 'User.Affiliation' in $\prec$. Figure 1 shows examples of policy chains where "Head" and "Tail" are used to indicate the first and the last term of a policy_chain respectively. Policy_chains, will be used in the following to establish an order on evaluation among expressions to support the pruning technique as introduced in Section 5.1.

6.2 User subscription phase

The User subscription phase, executed when a new user joins the Grid, aims at collecting the user profile and performing a preliminary evaluation of policy expressions to reduce, as much as possible, the number of constraints to be evaluated for each future request issued by the same user. In this phase a user specifies a pair ⟨variable,value ⟩ for each attribute into the static profile whereas the attributes into the dynamic profile are simply listed with no values (which can change for each requests). Based on these information, two lists of nodes can be associated with a single user:

1. *Statically_Evaluated_Nodes*. It contains nodes whose LPs consist only of policy expressions that can be completely evaluated using information available into the static user profile.

2. *Dynamically_Evaluated_Nodes*. It contains nodes whose LPs have at least one policy expression defined using an attribute which belongs to the dynamic user profile.

Nodes into the *Statically_Evaluated_Nodes* list will authorize (or reject) all requests from the user without further rule evaluation at job submission phase. Therefore, such lists can be cached into a table (referred as Statically_Evaluated_Table in the following) The table maintains an entry for each user registered to the Grid and for each possible resource owner. The cell $\langle User_x, Node_y \rangle$ contains 1 if $Node_y$ belongs to requirements evaluated as Statically_Evaluated_Nodes for $User_x$'s profile; 0 otherwise. The table, storing only a restricted set of information for each user, scales well with large number of users and nodes and it can be stored centrally or replicated over different hosts.

On the other hand, nodes into *Dynamically_Evaluated_Nodes* list require additional evaluations in order to decide whether a user request can be authorized or denied access to a shared resource. As an example consider a node n_i whose policy rule is composed of four policy expressions A, B, C, and D. Moreover, assume u_j to be a user such that his/her static profile suffice for evaluating A and C, whereas B and D are dynamic. In this case, n_i belongs to set Dynamically_Evaluated_Nodes for u_j because the authorization may vary upon each single user request.

However, instead of postponing the evaluation of all the policy expressions at job-submission phase, we adopt a two-phases evaluation process in which expressions related to static attributes (A and C in our example) are evaluated at subscription phase, whereas the evaluation of expressions related to dynamic attributes (i.e., B and D) is left to the job-submission phase. Given a policy rule, the list of policy expressions requiring evaluation at job-submission phase will be chained into Node_policy_chain similar to the one proposed in the User_submission_phase. Unlike previous phase, where policy_chains are built to organize all logical expressions defining a LP, here the chains are subsets of the original LPs and apply only to requests from a specific user (i.e. specified with a particular user-profile). In our example, $n_i.policy\_chain$ derived from u_j user profile will keep a link from the expression B to the SVO containing policy expression D.

7 Job-Submission Phase

The job submission phase is executed each time a job is issued by a user to the Grid and it is in charge of extracting the list of compliant resources using which the task will run. For each request issued to the Grid, the user submits his/her profile (i.e., including a value for each attribute into the dynamic profile) to the Security Engine which executes two main activities: (i) it extracts the list of nodes which authorize all requests from the Statically_Evaluated_Table created during the user-subscription phase and (ii) it completes the evaluation of the nodes whose policies require values belonging to the dynamic part of the profile.

The first task is straightforward. It does not require any rule evaluation but it simply queries the Statically_Evaluated_Table to extract the compliant nodes which authorize all the requests submitted by the user independently from the values of the attributes in the dynamic user profile.

The second task comprises the evaluation of nodes whose LPs authorize or deny a request based on information stored into the dynamic user profile. During the User-submission phase, LPs have been evaluated for the policy expressions using static attributes. In this phase, the evaluation is completed for the part involving attributes into the dynamic user profile. To accomplish this task, the policy chains will be evaluated according to the pruning technique introduced in Section 5.1. In our example, n_i LP has been partially evaluated for user u_j at User subscription phase (i.e., for the policy expressions A and C) and the remaining expressions (B and D) combined as a policy_chain which is evaluated for each request issued by u_j based on the values specified for the dynamic attributes.

8 Conclusions and Future Work

In this paper, we explored the need of a security component supporting fine-grained authorization policies of resources in a Grid environment. Specifically, we designed this component as a part of the resource allocation, as we believe this is a key component in resource access control. In our model, we propose a novel way of organizing LPs which reduces the cost of policy evaluation, using dynamic grouping of attributes and maintaining user profiles.

As a next step, we plan to integrate our security component with a well known existing schedulers or resource brokers. We are also working on performance analysis of our approach and comparing it with that of Condor's matchmaker for evaluating security-related attributes. Further, we plan to extend the system such that user can specify security constraints for their requests to filter which nodes he/she is willing his/her jobs to be running.

Acknoledgment

We thanks prof. Thilo Kielmann and prof. Andy Tanenbaum for reading the draft of the paper and providing useful feedbacks.

References

[1] Farag Azzedin and Muthucumaru Maheswaran, *Integrating trust into grid resource management systems*, In International Conference on Parallel Processing (ICPP'02), Vancouver, B.C., Canada , August 2002.

[2] OASIS Security Services Technical Committee, *Assertions and protocol for the oasis security assertion markup language (saml)*, Sept 2003.

[3] Elena Ferrari Elisa Bertino, Silvana Castano, *On specifying security policies for web documents with an xml-based language*, SACMAT01 ACM Symposium on Access Control Models and Technologies (2001).

[4] Ian Foster, Carl Kesselman, and Steven Tuecke, *The anatomy of the Grid: Enabling scalable virtual organizations*, Lecture Notes in Computer Science **2150** (2001).

[5] Ian T. Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke, *A security architecture for computational grids*, ACM Conference on Computer and Communications Security, 1998, pp. 83–92.

[6] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran, *A taxonomy and survey of grid resource management systems for distributed computing*, Software, Practice and Experience **32** (2002), no. 2, 135–164.

[7] Rajesh Raman, Miron Livny, and Marvin Solomon, *Policy driven heterogeneous resource co-allocation with gangmatching*, 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03) (2003).

[8] Todd Tannenbaum, Derek Wright, Karen Miller, and Miron Livny, *Condor – a distributed job scheduler*, Beowulf Cluster Computing with Linux (Thomas Sterling, ed.), MIT Press, October 2001.

[9] M.R. Thompson, A. Essiari, K.KIeahey, V. Welch, S. Lang, and B. Liu, *Fine-grained authorization fo job and resource management using akenti and the globus toolkit*, (2003).

[10] OASIS Security Services Technical Committee XAMCL, *extendible access control markup language (xacml) committee specification 1.0*, Feb 2003.