Access Control Policy Combinations for the Grid Using the Policy Machine

Vincent C. Hu, David F. Ferraiolo, and Karen Scarfone
National Institute of Standards and Technology, Gaithersburg, Maryland 20899-8930, USA
Email: {vhu, dferraiolo, karen.scarfone}@nist.gov

Abstract

Many researchers have tackled the architecture and requirements aspects of grid security, concentrating on the authentication or authorization mediation instead of authorization techniques, especially the topic of policy combination. Policy combination is an essential requirement of grid, not only because of the required remote (or global) vs. local interaction between grid members, but also the dynamic scalability nature of handling the joining and leaving of grid membership. However, evolving from the general security requirements of grid, the independency of a grid member's access control system is critical and needs to be maintained when the access decision is determined by the combination of global and local access control policies. The Policy Machine (PM) provides features which not only can meet the significant independency requirement but also have better performance, easier management, and more straightforward policy expression than most of the popular policy combination techniques for grid.

1. Introduction

Grid computing has become closer to reality due to the maturity of the current computing technologies [1]; however, grid systems have greater challenges compared to non-grid systems with infrastructure security issues such as access control (authorization), directory services, and firewalls/VPN. Of these challenges, grid access control is the most crucial and difficult, because the management of access control on a multi-organization grid and the grid-mapfile does not scale well, and it works only at the resource level, not the collective level [2]. The difficulties lie under the usual case that the resources are available both locally and conditionally grid-wide; to not violate the principle of reference monitor, both the local and grid access control policies should be integrated under one access control management system.

Many have researched the criticality and requirements [3] for the interaction between grid and local access control policies, but few have discussed practical approaches for solving the problems. One reason is that most access control mechanisms and models are not flexible enough to arbitrarily combine access control policies [4]. NIST has proposed the idea of combining access control systems using the universal *Policy Machine* (*PM*) [5], which, by composing the relations of the basic access control elements (i.e., subjects, operations, objects, and attributes), can combine not only well-known access control policies but also novel ones. The technique for combining policies provides practical solutions for the integration issues of grid and local policies.

This paper contains six sections. Section 1 introduces the motivation for the research. Section 2 defines terms applied to this paper. Section 3 discusses the grid access control requirements that evolve to the requirements for and definitions of policy combinations. Section 4 introduces the *Policy Machine* (*PM*) and demonstrates how *PM* combines grid and local access control policies. Section 5 compares *PM* with XACML and its related work. Section 6 is the conclusion.

2. Terms

Because all the authorization for the combined policies is performed at the local system, we define a set of terms named from the perspective of an individual grid member. Figure 1 illustrates the relations of the following terms.

- **Grid member** (others call it participant, note, element, or computer) refers to a system that participates as a member of a grid.
- **Local system** refers to an individual grid member and its trust domain. A local system sees other grid members as **remote systems**.
- **Resource** is hardware and software accessible by any grid member; it is **local** if only available through the local system, and **global** otherwise.
- **User** is a registered principal of a grid. A user is **local** if the resource the user is requesting to access

is located on the same system as the user login, and **remote** otherwise.

- **Subject** is the delegation of a remote or local user when a resource access assigned by the system is activated.
- **Trust domain** is a collection of systems managed under one security policy. The users and objects under the same trust domain are mutually trusted. The local trust domain is managed by local access control policy, and grid trust domains are governed by the matching grid access control policies.
- **Combined policy** is created by incorporating local and grid access control policies. There may be multiple grid policies in regard to different trust domains in the grid; therefore, there might be multiple grid-local combined policies in a local system. We only use one in this paper for simplicity in explaining concepts, but the principle can be applied to multiple policy combinations.

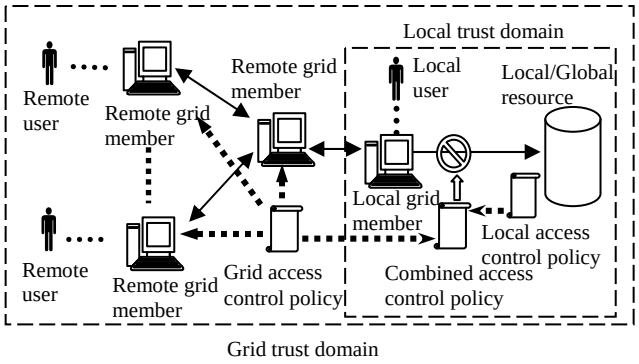


Figure 1 – Grid from a local system's point of view

3. Grid Access Control Requirements

Besides general security functions, including authentication, integrity, privacy, and nonrepudiation, grid requires a standard access control protocol between grid members. To support that, many requirements have been proposed [3,6,7]. We summarize these in four major categories: Infrastructure, Authentication, Independency, and Combination. The Infrastructure category contains the security requirements, which are no different than the requirements for most non-grid computing systems. The other three categories contain requirements specific to grid. The following describes the requirements without listing details to the clause level, for our purpose is to abstract the concepts for the main topic – policy combination.

Infrastructure requirements – This category contains the fundamental security requirements that also apply to non-grid computing systems, including networks and standalone systems. The requirements include forgery prevention (user and server), wire-tapping and tamper prevention, intrusion detection, resource abuse prevention, and event logging. These requirements protect the underlying communication and system infrastructures of the grid.

Authentication requirement – As trust influences access control enforcement [8], the operations between grid members covered by different trust domains require mutual authentication [7]. Thus, grid users must be grid-wide recognized, and a delegation for a remote user is created to access the local system. So, a program or process as a subject acts on behalf of the remote user and can be delegated a subset of the local user's rights.

Independency requirement – A trust domain is defined by the trust among grid members; it should be maintained such that other trust domains have limited or no influence over the local one. In other words, we can neither require that local security policy be replaced, nor are we allowed to override local policy decisions when interacting with the global environment. Further, no additional security operations or services are imposed on the local security mechanisms. It is impractical to modify local resources to accommodate the global environment. Consequently, a combined policy must focus on controlling the interdomain interactions and the mapping of interdomain operations into local access control policy [7], which might require the separation of access control policies and mechanisms.

Combination requirement – All access should be controlled by the combination of local and grid policies if the resource is available for both local and remote access. There are four basic types of combinations: merge two policies by returning their intersection, merge two policies by returning their union, restrict a policy by eliminating all accesses in a second policy [8], and mutually exclude two policies. The configuration of combinations should allow a remote system to update and specify invocation to the grid policy. The authorization should accommodate various access control models and implementations to allow remote systems and a local system to dynamically exchange security policy information (as well as other information) to negotiate a security context [6].

Policy combinations

As previously discussed, the *combination requirement* is one of the critical functions of grid, not only because of the inevitable complexity of remote vs. local interaction between grid members, but also the dynamic scalability nature of handling the joining and leaving of grid membership. Many researchers

[6,7] have tackled the architecture and requirement aspects of this issue, mostly concentrating on the authentication or authorization mediation instead of the authorization techniques, especially on the topic of policy combination.

Policy combination can be as diverse as its application. The building blocks of all varieties of combinations are intersection, union, elimination, and exclusion of policies. The complexity degree of a combination is dictated by the number of access control policies included. Many have concluded that to support policy combination, grid mechanisms must enforce the following: 1) users have to be community recognizable (authentication requirement), such as through the management of an LDAP system; 2) users' roles have to be globally vs. locally meaningful (independency requirement); and 3) the local repository resolves conflicts between global and local policies [8]. Providing the three required characteristics, we can now define the basic combinations as the following with examples. We leave the formal definitions (by *PM*) to Section 4.

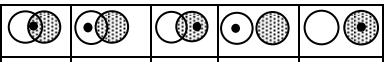
For the following definitions, we denote an access request as a tuple (*subject*, *operation*, *object*) such that *subject* requests to access an *object* of a resource with operation *operation*.

Definition 1 - ***Policy Intersection***: An access request is granted only if the request is granted by both policies *A* and *B*, as shown in Figure 2, where a dot represents a request, a circle represents the trust domain for an access control policy (clear circle is policy *A*, and shadowed is policy *B*), which is a set of permitted requests, and an X means a request is granted. For example, in addition to being authorized for an access request by local policy *A*, a multilevel policy that ensures every user has the proper security classification, a user also needs to have matching role assignments assigned by role-based access control policy *B*, which authorizes certain roles for remote users.

Definition 2 - ***Policy Union***: An access request is granted only if the request is granted by either policy *A* or *B*. For example, a grid is formed by affiliated stores such that grid members help each other by fulfilling orders if an item is not available locally. Any grid member should accept orders from remotely or locally registered grid users. Thus, in the local system, the products (resources) are available for both local customers (managed under policy *A*) and remote customers (managed under policy *B*).

Definition 3 - ***Policy Elimination***: An access request is granted only if the request is granted by policy *A* but not policy *B*. For example, to prevent a conflict of interest (COI), the local system grants a user access to read information through policy *A* only if the same information cannot be obtained from policy *B*, so that a user cannot work in the same capability for company *B*, for which users can access the same information through access control policy *B*. Note that COI-related resources are globally managed such that policy *B* is updated grid-wide.

Definition 4 - ***Policy Exclusion***: An access request is granted either by policy *A* or *B* but not both (mutual exclusion, which is different from *policy elimination* that does not grant the request that is granted by policy *B*). For example, users can claim their rebate either through purchasing from retail stores (managed by local policy *A*) or through purchasing from wholesalers (managed by remote policy *B*) but not both, therefore preventing being double-rebated. Note that once a rebate is sent to the user, the user will be removed from policy *A* and *B*.

	
Intersection	X
Union	X
Elimination	
Exclusion	

X	X	X	X	X
	X		X	
	X	X	X	X

Figure 2 – Policy Combination

4. Policy Machine

NIST has initiated a project in pursuit of a standardized access control mechanism referred to as the *Policy Machine* (*PM)* [4,5,9]. It is a practical approach to implementing the four basic combination functions. *PM* is based on the principle that the separation of access control policies from mechanisms [10] allows enforcement of multiple policies within a single, unified system so that access control rules from different authorities may be integrated with each other [8].

As shown in Figure 3, the *PM* architecture is composed of the *Policy Engine (PE)*, which includes *PE* processes and the *PE* database, and the *General Policy Management System* (*GPMS*). *PE* is the processing unit that receives user requests and performs the authorization process by referencing information from the *PE* database; it then generates a Boolean value (*grant* or *deny*) as a result. *GPMS* is the interface for *PM* administrators to configure and compose policies and to manage the *PE* database.

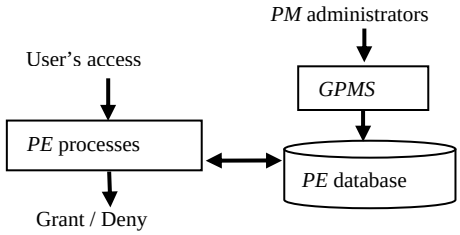


Figure 3 – *PM* schema

PM categorizes users, objects (grid resources) and their attributes into policy classes, and appropriately enforces subsets of the policies in response to a user's access request. The following fundamental data sets for *PM* processing are stored in the *PE* database:
U: The set of *PM* users under the *PM*'s control
UA: The set of user attributes of *U*
OP: The set of operations (access rights) permitted by the *PM*
O: The set of objects under the *PM*'s control
OA: The set of object attributes of *O*
PC: The set of policy classes the *PM* is implementing

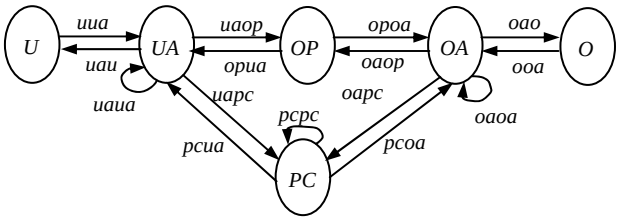


Figure 4 – *PE* database model

Figure 4 shows the *PE* database model, and Table 1 lists the functions *PE* uses for authentication.

Table 1 – Functions associated with the *PE* database

Function	Description – mapping relation of
$uua(u) = UA_u \subset UA$	a user *u* to a set of user attributes UA_u that *u* is assigned to
$uau(ua) = U_{ua} \subset U$	a user attribute *ua* to a set of users U_{ua} that *ua* is assigned to
$uaua(ua) = UA_{ua} \subset UA$	a user attribute *ua* to a set of inherited user attributes; a user assigned to *ua* will inherit the privileges of the user attributes in UA_{ua} *(note, $ua \in UA_{ua}$)*
$uaop(ua) = OP_{ua} \subset OP$	a user attribute *ua* to a set of operations OP_{ua} that *ua* has the privilege to perform
$opua(op) = UA_{op} \subset UA$	an operation *op* to a set of user attributes UA_{op} that has privilege *op*
$opoa(op) = OA_{op} \subset OA$	an operation *op* to a set of object attributes OA_{op} that can be accessed by *op*
$oaop(oa) =$	an object attribute *oa* to a set of
$OP_{oa} \subset OP$	operations OP_{oa} that can operate on *oa*
$pcpc(pc) = PC_{pc} \subset PC$	a policy class *pc* to a set of inherited policy classes (note, $pc \in PC_{pc}$)
$uapc(ua) = PC_{ua} \subset PC$	a user attribute *ua* to a set of policy classes PC_{ua} that covers *ua*
$pcua(pc) = UA_{pc} \subset UA$	a policy class *pc* to a set of user attributes UA_{pc} that are covered by *pc*
$oapc(oa) = PC_{oa} \subset PC$	an object attribute *oa* to a set of policy classes PC_{oa} that covers *oa*
$pcoa(pc) = OA_{pc} \subset OA$	a policy class *pc* to a set of user attributes OA_{pc} that are covered by *pc*
$ooa(o) = OA_o \subset OA$	an object *o* to a set of object attributes OA_o that *o* is assigned to
$oao(oa) = O_{oa} \subset O$	an object attribute *oa* to a set of objects O_{oa} that *oa* is assigned to
$oaoa(oa) = OA_{oa} \subset OA$	an object attribute *oa* to a set of inherited object attributes; an object assigned to this attribute will inherit all the privileges of the object attributes in OA_{oa} (note, $oa \in OA_{oa}$)

Note: all the mapping functions in Table 1 have 1 to $m \geq 0$ domain to range relation.

PM can allow inheritance relations among user attributes, object attributes, and policy classes such that an element inherits the privileges from the elements that it is inherited from. The inheritance relation has to be in the partial order to be legitimate. A set of members in an inheritance relation from one function to another function can be formally described by the union transitive closure of the two functions: $\cup_{y \in a(x)} b(y) = Z$ with the symbol "$x \rightarrow_{a,b}{}^*$". For example, all inherited user attributes UA_u of a user *u* can be denoted by $u \rightarrow_{uua,uaua}{}^*$, and all inherited object attributes OA_o of an object *o* is $o \rightarrow_{ooa,oaoa}{}^*$. We also denote by $x \rightarrow_y z$ that there are mapping relations from *x* to *y* to *z*.

4.1. Atomic authorization process of *PM*

Based on the above model and notation, the following definitions describe the *PE* authorization process and the states of *PM*.

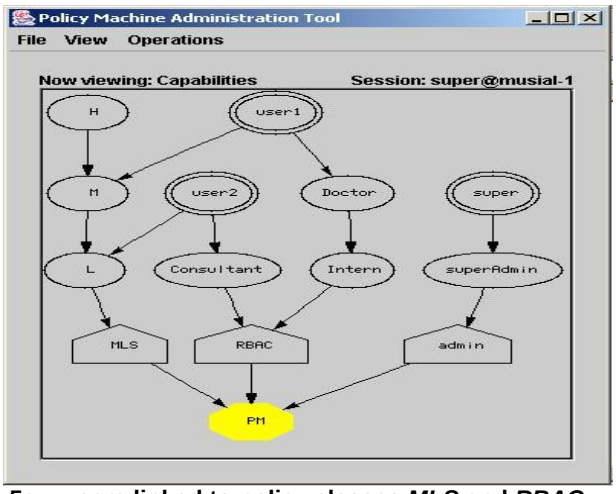
Definition 5 - A tuple **<*u, ua, op, pc, oa, o*>** $\in U \times UA \times OP \times PC \times OA \times O$ is an instance of the current configuration of a *PM*.

Definition 6 - *Grant_instance_of_policy*(*u, op, o, pc*), function decides if an access request (*u*, *op*, *o*) is satisfied in a *PM*, i.e. if there exists a policy class *pc* link (maps to) a *ua* and *oa* that the *u* and *o* are a member of, respectively, formally,

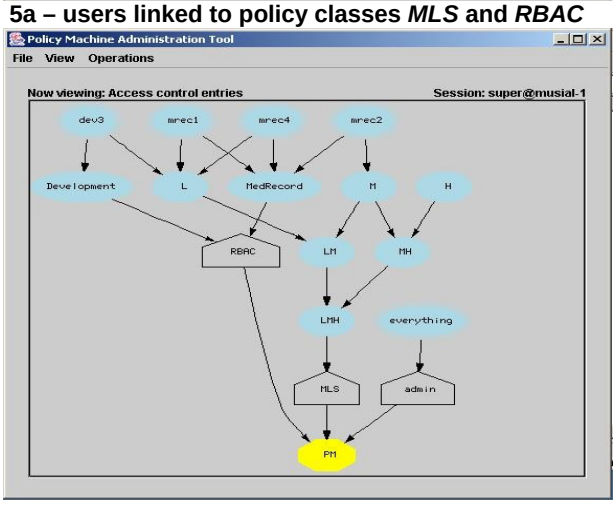
For $u \in U$, $op \in OP$, $o \in O$, $pc \in PC$,
Grant_instance_of_policy(u, op, o ,pc) = True $\Leftrightarrow$
$\exists\ ua \in UA$ and $\exists\ oa \in OA$, such that
1. $ua \in (u \rightarrow_{uua,uaua}*)$,
2. $oa \in (o \rightarrow_{ooa,oaoa}*)$,
3. $ua \rightarrow_{op} oa$,
4. $pc \in ua \rightarrow_{uapc,pcpc}*$, and
5. $pc \in oa \rightarrow_{oapc,pcpc}*$.

4.2. Cases of policy multi-coverage

Through the links to the policy class, more than one policy can be implemented for the same sets of users, user attributes, operations, objects, and object attributes. An example of NIST's implementation is shown in Figure 5: 5a shows *user1* and *user2* linked to policy classes *MLS* and *RBAC*, and in 5b the two policy classes are shown to be linked to the objects *dev3*, *mrec1*, *mrec2*, and *mrec4*. For simplicity, we only show the combination of two policies.



5a – users linked to policy classes *MLS* and *RBAC*



5b – objects linked to policy classes *MLS* and *RBAC*

Figure 5 - Example *PM* implementation from NIST

As shown, one request can be covered by <u, ua, op, pc_A, oa, o> and <u, ua, op, pc_B, oa, o>, i.e. *Grant_instance_of_policy*(u, ua, op, pc_A, oa, o) = *Grant_instance_of_policy*(u, ua, op, pc_B, oa, o). In other words, a user's access request can be authorized by more than one access control policy. Such conditions can be realized by four basic *PM* states, as described in the following states 1 to 4**.**

State 1, **multi-coverage by independent assignments**. As shown in Figure 6, this is a straightforward *PM* state where a user's request (u, op, o) is covered by two different, unrelated policies (for example, pc_M and pc_N). The following conditions are satisfied for such a state:
1.*Grant_instance_of_policy*(u, op, o, pc_M) = *Grant_instance_of_policy*(u, op, o, pc_N) = True,
2. $\{pc_M, pc_N\} \not\subset \{pcpc(pc_M) \cap pcpc(pc_N)\}$, and
3. $uua(u) \in pc_M \rightarrow_{pcpc,pcua}* = UA_A$, $uua(u) \in pc_N \rightarrow_{pcpc,pcua}* = UA_B$, $u \notin UA_A \cap UA_B$
Condition 1 above says that there is multi-coverage of (u, op, o). Condition 2 shows that the two unrelated (no inherit relation) policies pc_M and pc_N both cover the (u, op, o) request. Condition 3 makes sure that there are no common user attributes mapping for user u.

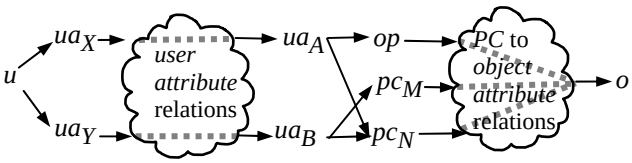


Figure 6 - Individual policy class assignment

State 2, **multi-coverage by hierarchical user attribute relation**. As shown in Figure 7, the user is assigned directly or indirectly to a user attribute ua_A by policy class pc_M, and ua_A is inherited (directly or indirectly) by other user attributes, say ua_B, which is covered under a different policy class pc_N. Both pc_M and pc_N have the same access right op to the same object o the user is request to access. The following conditions must be satisfied for such an access instance:
1.*Grant_instance_of_policy*(u, op, o, pc_M) = *Grant_instance_of_policy*(u, op, o, pc_N) = True,
2. $\{pc_M, pc_N\} \not\subset \{pcpc(pc_M) \cap pcpc(pc_N)\}$, and
3. $uua(u) \in pc_M \rightarrow_{pcpc,pcua}* = UA_A$, $uua(u) \in pc_N \rightarrow_{pcpc,pcua}* = UA_B$, $UA_B \not\subset UA_A$, $UA_A \subset UA_B$.

Condition 1 and 2 are the same as in State 1. Condition 3 makes sure that there is an inheriting relation, for example, from ua_A to ua_B.

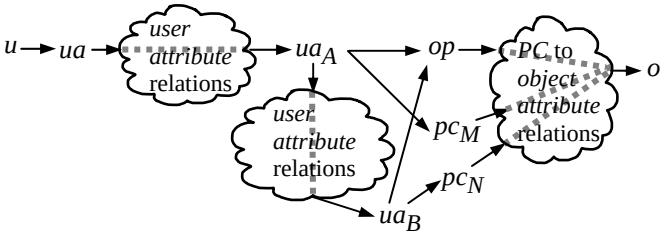


Figure 7 - Hierarchical user attributes coverage

State 3, **multi-coverage by hierarchical policy classes**. As shown in Figure 8, the user is covered by a policy class that inherits another policy class, which is authorized the access right to the object *o* the user is requesting. The following conditions have to be satisfied for such a request (*u, op, o*):

1. *Grant_instance_of_policy*(*u, op, o,* pc_I) = True, and

2. $pc_I \in \{pcpc(pc_M) \cap pcpc(pc_N)\}$, $pc_M \notin pcpc(pc_I)$, $pc_N \notin pcpc(pc_I)$

Condition 1 above says that the (*u, op, o*) request is covered by the policy, pc_I. Condition 2 shows that there are inheriting relations from pc_I to pc_M and from pc_I to pc_N.

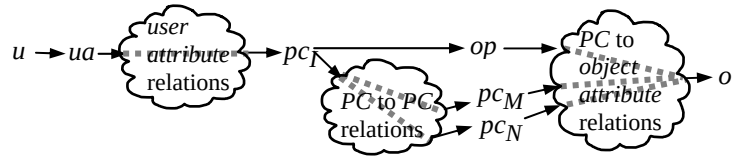


Figure 8 - Hierarchical policy class coverage

The above three basic states can be arbitrarily combined to form new types of states. Thus,

Grant_instance_of_policy(*u, op, o,* pc_1) =

Grant_instance_of_policy(*u, op, o,* pc_2) =……= *Grant_instance_of_policy*(*u, op, o,* pc_n) = True,

where *n* is the number of policies that are included in the combined state.

4.3. *PM* implementation of combination

As the data sets and functions described in Section 4.2 satisfy the *Combination requirement*, *PM* allows adding and deleting policy classes (therefore, the policies) without affecting the existing policy (classes), thus meeting the *Independency requirement*. It is easy to identify the policies that cover a user's access request, even with dynamic adding and deleting of policies. The authorization of user access requests via the four basic types of policy combinations becomes straightforward in the context of logic expressions of the atomic *PE* processes:

Policy Intersection is enforced for the application that requires both the grid and local policies to grant the user access request (*Definition* 1). If the function $Grant_{G \cap L}(u, op, o, pc_{grid}, pc_{local})$ is to decide the request (*u, op, o*) through the intersection of grid and local policy implemented by policy classes pc_{grid} and pc_{local} respectively, then the following should be true:

$Grant_{G \cap L}(u, op, o, pc_{grid}, pc_{local})$ =

Grant_instance_of_policy(*u, op, o,* pc_{grid}) ∧ *Grant_instance_of_policy*(*u, op, o,* pc_{local}).

Considering cases of multi-coverage states in Section 4.2, assume $pc_{grid} = pc_M$ and $pc_{local} = pc_N$, or $pc_{grid} = pc_N$ and $pc_{local} = pc_M$. The user's request will be granted for all cases.

Policy Union is enforced for the application that requires either the grid or local policy to grant the user access request (*Definition* 2). If the function $Grant_{G \cup L}(u, op, o, pc_{grid}, pc_{local})$ is to decide the request (*u, op, o*) through the union of grid and local policy implemented by policy classes pc_{grid} and pc_{local} respectively, then the following should be true:

$Grant_{G \cup L}(u, op, o, pc_{grid}, pc_{local})$ = *Grant_instance_of_policy*(*u, op, o,* pc_{grid}) ∨ *Grant_instance_of_policy*(*u, op, o,* pc_{local}).

As with Policy Intersection, all cases of multi-coverage states in Section 4.2 will be granted.

Policy Elimination is enforced for the application that grants a user's request only if the request is authorized by one policy but not the other one (*Definition* 3), so if the function $Grant_{G\text{-}L}(u, op, o, pc_{grid}, pc_{local})$ is to decide the request (*u, op, o*) through the elimination of the local policy pc_{local}, then the following should be true:

$Grant_{G\text{-}L}(u, op, o, pc_{grid}, pc_{local})$ = *Grant_instance_of_policy*(*u, op, o,* pc_{grid}) ∧¬ *Grant_instance_of_policy*(*u, op, o,* pc_{local}).

Or in reverse, if the grid policy is eliminated, then

$Grant_{L\text{-}G}(u, op, o, pc_{grid}, pc_{local})$ = *Grant_instance_of_policy*(*u, op, o,* pc_{local}) ∧¬ *Grant_instance_of_policy*(*u, op, o,* pc_{grid}).

Considering cases of multi-coverage states in Section 4.2, assume $pc_{grid} = pc_M$ and $pc_{local} = pc_N$, or

$pc_{grid} = pc_N$ and $pc_{local} = pc_M$. The user's request will be denied for all cases.

Policy Exclusion is enforced for the application that grants a user request only if the request is covered by either one of the policies but not both (*Definition* 4), so if the function $Grant_{G \oplus L}(u, op, o, pc_{grid}, pc_{local})$ is to decide the request (u, op, o) through the mutual exclusion of grid and local policy implemented by policy classes pc_{grid} and pc_{local} respectively, then the following should be true:

$Grant_{G \oplus L}(u, op, o, pc_{grid}, pc_{local}) =$
$Grant\_instance\_of\_policy(u, op, o, pc_{grid}) \oplus$
$Grant\_instance\_of\_policy(u, op, o, pc_{local})$.

Or, expressed by AND operations of *Policy Elimination*:

$Grant_{G\text{-}L}(u, op, o, pc_{grid}, pc_{local}) \wedge Grant_{L\text{-}G}(u, op, o, pc_{grid}, pc_{local}) =$ True

As with Policy Elimination, all cases of multi-coverage states in Section 4.2 will be denied.

Extending the above elementary combinations, various complex combinations can be expressed by composing propositional Boolean logic. For example, three trust domains in a grid are controlled such that a user request can be granted only if it is granted by both trust domains *x* and *y*, or local domain l, and has to be excluded from trust domain *z*. The expression for such a policy is:

$Grant_{xylz}(u, op, o) = (Grant_{G \cap L}(u, op, o, pc_x, pc_y) \vee$
$Grant\_instance\_of\_policy(u, op, o, pc_l)) \wedge \neg$
$Grant\_instance\_of\_policy(u, op, o, pc_z)$

5. Related Works

XACML [11] provides a flexible and mechanism-independent representation of access rules that vary in granularity, allowing the combination of different authoritative domains' policies into one policy set for making access control decisions in a widely distributed system environment.

Several systems that support emerging grid computing use XACML for their authorization mechanism, which can be integrated into the GSI authorization framework [12] as authorization services. For example, Shibboleth [13] is considering XACML to replace its current access control system; accessed resources in Cardea [14] are protected by local access control policies in XACML syntax; PRIMA [15] applies XACML to allow more flexible specification of access control rules in privileges and policies; the PERMIS [16] project is investigating XACML to replace parts of its proprietary policy language; and the Akenti [17] team will investigate using XACML for the representation of distributed policies and the applicability and effectiveness of the policy combining mechanism [18].

The flexibility and expressiveness of XACML make it complex and verbose. It is hard to work directly with the language or policy files. Supporting XACML in a heterogeneous environment also calls for fully specified data type and function definitions that produce a highly verbose document even if the actual policy rules are trivial: an example is manual creation of XACML policies by ordinary users in PRIMA. In general, platform-independent policies expressed in an abstract language are difficult to create and maintain by resource administrators [18]. *PM* is not a language, so it is free from syntactic and semantic complexity. It employs the Boolean logics of AND, OR, NOT, and EXCLUSION to express the combination of policies, in contrast to XACML, which has only the AND operation with six algorithm varieties. When describing hierarchical relations between attributes or policies, *PM* only requires adding links between them; in XACML, relations need to be inserted in precise syntactic order. *PM* also has a WYSIWYG graphic user interface that visually aids in the management of policy documents; administrators can "see" how the managed access control attributes are related to each other, as well as the policy under which the attributes are covered.

Depending on the granularity of access decisions, there is significant overhead for XACML, especially for enterprise systems. For example, handling an HTTP request to render a portal page that aggregates many resources may require many access decisions. Handling this with XACML would require many requests to be generated since each may only contain a single action element. The resulting processing time and delay in collecting all access decisions may not be acceptable [19]. In contrast to XACML multiple policies, which have to be bounded by the combining algorithm in the same PolicySet, *PM* can independently implement multiple policies by assigning them to different policy classes, allowing policy decisions to be processed independently. This avoids the time delay due to the sequence of overhead algorithms, thus *PM* has better performance than XACML. This feature is essential for meeting the *Independency requirement* (Section 3) of grid security, especially when adding and deleting policies in a policy combination.

6. Conclusion

This paper began by introducing general grid security requirements, which led to the integration of

requirements for the combination of global and local access control policies for grid systems. According to the requirements, four fundamental mechanisms were defined – intersection, union, elimination, and exclusion – in various policy combinations.

We then presented a formal model of universal access control mechanism, *PM*, through data set and mapping functions. Based on the model, basic states of multi-policy coverage cases for a user request were explained. Those states alone with separated coverage states were used to demonstrate the solution that *PM* provides for combining global and local policies in grid. At the end, we briefly introduced the most studied policy combination approach, XACML, discussed its applications and limitations, and showed the advantages of *PM*: easier policy expression and management, and better performance.

Even though the focus of this paper is on the grid application, the proposed policy combination mechanism can also be applied to distributed operating environments, so that each processing unit can locally manage its own resources as the local system presented in this paper did, improving enforceability [20] due to the network wide access control configuration.

7. References

[1] The 451 Group, "Grid Technology User Case Study", *Grids 2004 report*, JP Morgan Chase, New York, 2003.

[2] Chien A. A., "Globus Grid Security", *CSE225 Lecture note #13*, University of California, San Diego, 2004.

[3] Enterprise Grid Alliance Security Working Group, "Enterprise Grid Security Requirements", 2005.

[4] Hu C. V., "The Policy Machine For Universal Access Control", *Dissertation, Computer Science Department, University of Idaho*, 2002.

[5] Hu C. V., Frincke D. A., and Ferraiolo D. F., "The Policy Machine For Security Policy Management", *Proceeding ICCS Conference*, San Francisco, 2001.

[6] Nagaratnam N., Janson P., Dayka J., Nadalin A., Siebenlist F., Welch V., Foster I., and Tuecke S., "The Security Architecture for Open Grid Services", *Global Grid Forum, Open Grid Service Architecture Security Working Group (OGSA-SEC-WG)*, 2002.

[7] Foster I., Kesselman C., Tsudik G., and Tuecke S., "A Security Architecture for Computational Grids", *5th ACM Conference on Computer and Communication Security*, 1998.

[8] Coetzee M., and Eloff J. H. P., "Virtual Enterprise Access Control Requirements" *Proceedings of SAICSIT*, pp. 285-294, 2003.

[9] Ferraiolo D. F., Gavrila S., Hu C. V., and Kuhn D. R., "Composing and Combining Policies under the Policy Machine", ACM SACMAT, 2005.

[10] Jajodia S., Samarati P., and Subrahmanian V. S., ``A Logical Language for Expressing Authorizations," *Proc. IEEE Symp*, Oakland, Calif., 1997.

[11] OASIS, "Extensible Access Control Markup Language (XACML), TC", http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

[12] Lang B., Foster I., Siebenlist F., Ananthakrishnan A., and Freeman T., "A Multipolicy Authorization Framework for Grid Security", *Proc. Fifth IEEE Symposium on Network Computing and Application*, Cambridge, USA, 2006.

[13] Erdos M., and Cantor S., "Shibboleth Architecture v5", Internet2/MACE, 2002.

[14] Lepro R., "Dynamic Access Control in Distributed Systems", *NAS Technical Report NAS-03-020*, 2003.

[15] Lorch M., et al, "The PRIMA System for Privilege Management, Authorization and Enforcement in Grid Environments", communicated to the 4th Ind. Workshop on Grid Computing- Grid 2003.

[16] Chadwick D. W., and Otenko A., "The PERMIS X.509 Role Based Privilege Management Infrastructure", *7th ACM Symposium on Access Control Models and Technologies*, 2002.

[17] Thompson M., Essiari A., and Mudumbai S., "Certificate-based Authorization Policy in a PKI Environment", *ACM Transactions on Infomation and System Security (TISSEC)*, Vol. 6 No. 4, pp. 566-588, 2003.

[18] Lorch M., et al, "First Experiences Using XACML for Access Control in Distributed Systems", *ACM Workshop on XML Security*, Fairfax, Virginia, 2003.

[19] Griffin P., "Introduction To XACML", *DeV2Dev*, http://dev2dev.bea.com/pub/a/2004/02/xacml.html.

[20] Hu C. V., Kuhn D. R., and Ferraiolo D. F., "The Computational Complexity of Enforceability Validation for Generic Access Control Rules", *Proceeding IEEE SUTC2006 Conference*, Taichung, Taiwan, 2006.