Journal of Information Assurance and Security
ISSN 1554-1010 Volume 6 (2011) pp. 141-148

Attribute Relations Specifications and Constraints Using Attribute Based Mechanism of Policy Machine

Vincent C. Hu[1], David D. Ferraiolo[2], Serban Gavrila[3]

[1]National Institute of Standards and Technology
Gaithersburg. Maryland, USA
vhu,@nist.gov

[2]National Institute of Standards and Technology
Gaithersburg. Maryland, USA
dferraiollo@nist.gov

[3]National Institute of Standards and Technology
Gaithersburg. Maryland, USA
gavrila@nist.gov

***Abstract*: Attribute relations in access control mechanisms or languages allow accurate and efficient specification of some popular access control models, they can also be regulated for some constraints of an access control polices. However, most of the access control systems including today's de-facto access control protocol and specification language, XACML, does not provide sufficient syntactic and semantic support for the specification or constraint of attribute relations in their scheme. In this paper, we show the deficiencies of XACML in specifying and constraining such capabilities in the implementations of the Multilevel Security, Hierarchical Role Based, Separation of Duty, and Safety requirements of access control systems. In comparison, we then demonstrate the mechanisms for the capabilities provided by a relation-based access control mechanism – the Policy Machine.**

Keywords: access control, access control model, access control policy, authorization, privilege management, XACML

I. Introduction

A critical capability of an access control (AC) system is to allow an AC administrator to specify relations between AC attributes. With this capability, an AC system is able to maintain hierarchical orders (i.e. the inheritance relation of access privileges) of the attributes of the AC elements (subjects, actions, objects). The relations of privilege inheritance is specified by attribute relation (AR), which is essential for many popular AC models such as **Bell-La Padula** [1] (BLP) and **Biba** (BI) [2] of **Multileve Security (MLS)**[3], and **Hierarchical Role Based Access Control** (HRBAC) [4] as well as constraint policies such as **Separation Of Duty** (SOD) [5].

The syntactic and semantic supports of attribute relation specifications in AC mechanisms or languages allow not only accurately specifying but also efficiently enforcing the relation-based AC models and policy constraints. The specific advantages of such capabilities include:

- Specifying hierarchical relations for the inheriting or inherited privileges of subjects, actions, and objects in AC policies by attribute relations. For example, if subject *X* is **related** to subject *Y* then subject *X* **inherits** all the access privileges of subject *Y*.
- Efficient management of AC rules, such that AC policy administrators can modify privileges based on attribute groups and relations without making errors. And it is possible to display all the linkages of existing related attributes through GUI (Graphic User Interface), thus providing a complete view of the current privilege assignments.
- Enforcing SOD policies, which require an AC system to render all the inherited privileges of the targeted subjects.
- Performance enhancement for evaluating access requests, because an AC system does not have to go through all the AC rules to collect attribute information for the grant decision if higher level attributes of the request can be found to match the rule.
- Constraining the action and object pairs of an access privileges of particular attributes for the assurance of safety, which guarantees no leakage of information.

eXtensible Access Control Markup Language (XACML) [6,7] is today's de-facto protocol and specification language scheme for AC implementations. XACML provides a

flexible and mechanism independent representation of access rules that vary in granularity; it allows the combination of different authoritative domains' policies into one policy set for making access control decisions in a widely distributed system environment. However, XACML does not provide a scheme for specifying ARs; instead, ARs can be implemented in one of its architectural components (e.g., PEP) by ad-hoc applications.

In this paper, we first show the deficiencies of XACML in specifying and constraining ARs. We then demonstrate the virtues of an AR mechanism from a relation-based AC mechanism – *Policy Machine* (*PM*) [8, 9, 10], which embeds access control policies in standard access control attributes. *PM* includes a server engine called *Policy Server (PS)* and a policy management system, called *General Policy Management System (GPMS)*. *PS* and *GPMS* together enable enforcement of multiple access control policies within a single, unified system. *PM* is highly extensible, since it composes and combines access control policies from a relatively small set of atomic properties completely expressed with mappings and interrelationships of the ARs on three basic elements – *Subject Sets*, *Object Sets*, and *Operation Sets*. Mappings and interrelationships of ARs are enforced with a database and a fixed set of functions.

This paper contains six sections. Section 1 introduces the AR of AC policies. Section 2 explains AR implementation in the popular XACML AC scheme. Section 3 introduces the architecture and functions of *Policy Machine* (*PM*). Section 4 demonstrates *PM*'s mechanism for specifying ARs for AC models and constraints for safety requirements. Section 5 compares *PM* with related work. Section 6 is the conclusion.

II. Attribute Relations in XACML

XACML provides an AC policy specification language in an XML scheme as well as generic architecture components: Policy Decision Point (PDP), Policy Evaluation Point (PEP), Policy Information Point (PIP), and Policy Administration Point (PAP) for the AC enforcement functions. The regular expressions of XACML Version 2 are listed below:

(1) PS: T+ PS + P + PCA + O
(2) T: S + R + A + E
(3) P: T + RL +RCA + O
(4) RL: T + C + E

where *PS* is the *Policy Set*, *T* is the *Target*, *P* is the *Policy*, *PCA* is the *Policy Combination Algorithm*, *O* is the *Obligation*, *S* is the *Subject*, *R* is the *Resource*, *A* is the *Action*, *E* is the *Environment*, *RL* is the *Rule*, *RCA* is the *Rule Combining Algorithm*, *C* is the *Condition*, and *E* is the *Effect* of the XACML language scheme.

Regular expressions (*2*) and (*4*) are used for composing AC rules by the basic AC elements: subjects, resources, actions, and environment variables. Regular expressions (*1*) and (*3*) are for associating (*2*) and (*4*) in two different levels. There is no grammar for the expression of ARs in these four regular expressions unless specified by enumerating every relation between attributes. Additionally, XACML allows functions to be implemented to handle ARs in a PEP or an extended function. And those two methods are ad-hoc efforts without formal and structural definition in the scheme. In comparison, we will introduce an AC mechanism that provides a well-defined framework for the specification of attributer relations in Section 3.

Note that even though XACML Version 3 has more (and concise) elements in the language scheme than Version 2, for the purpose of explaining the ARs by the basic AC elements (i.e., subject, action, and object), we use XACML Version 2. The issues discussed in this paper apply to Version 3 as well.

A. Specification of MLS and HRBAC Policies

BLPB models for MLS policies require assigning **classes** (ranks) attributes to subjects and objects. Formal definitions are $Rs = \{\ldots(Sa_i, Sa_j)\ldots..\}$ and $Ro = \{\ldots(Oa_i, Oa_j)\ldots\}$, where *Rs* is a set of ARs for subject classes: for instance, Sa_i is the "Top Secret" class and Sa_j is the "Secret" class. *Rs* defines the "no read up" property of BLPB. In the same manner, Oa_i and Oa_j define the object classes and property. Instead of classes, HRBAC model uses Sa_i and Sa_j to define the hierarchical AR of privilege inheritance from Role Sa_j to Role Sa_i; for example, Role "Professor" inherits all Role "Teaching Assistant" privileges in a grading system. To specify and enforce these relations in XACML, AC policy authors need to specify all the possibilities including direct and indirect relations between the classes or roles. In the worst case, it requires $O(n^2)$ number of (*2*) type of statements from the regular expression to describe the relations for *n* number of classes or roles in the policy. Further, there is no semantic support for checking the correctness (e.g., cyclic assignment) of the specifications.

B. Specification of Separation of Duty Policies

When required to enforce SOD polices to prevent conflicts of interest or to control business processes, the access state of the AC system is dynamically dictated by some system variables. For example, a SOD policy restricts a subject's action and object pairs of privilege not to exceed a predefined number, so that no subject should be assigned to more than *k* number of such capabilities. Another SOD policy guarantees that no less than *k* number of subjects can perform all of a set of action and object pairs of privilege (i.e., requires at least *k* number of subjects to perform all of them). To specify and enforce these SOD policies, XACML needs to maintain counters for monitoring the number of such action and object pairs consumed by each subject currently in the system. Thus, the XACML's *obligation* and *environment* elements are used to update and retrieve (read

in) the external counters, respectively. And to compose SOD policies, statements in regular expression (*4*) are needed for referencing the environment variables (e.g., external counters) and statements (*3*) are used to store updated variables. However, the challenge is to accurately maintain the constraint variables (the number *k* in our examples), because a subject's access request can be granted from more than one (*4*) statement. And (*4*) may be encompassed in (*1*) (*2*) and/or (*3*) statements, which provide no syntax for maintaining the ARs between (*4*)s. For example, a subject may be granted access both from Role *X* and Role *Y* to the same object, since there is no way to specify the fact that *X* inherits *Y*, therefore, the *k* number of the action and object pairs for this subject is counted twice (which is supposed to be once) from both *X* and *Y* attributes in the same access session. It is hard to ensure a SOD policy is implemented without errors in XACML, because even though ARs can be specified in the language, there are no syntactic and semantic supports for the correctness of the specification unless by custom application through functions in PEP or PDP.

C. Specification of Attribute Contraints

Some policies require to constrain a specific access privilege only be accessible from specific subject attributes; In other words, there is no inheritance relation allowed from or to these attributes. On the contrary, some policies regulate that a specific subject attribute can only access to a specific action and object pair through other assigned subject attributes other than itself. Such constraints are for safety enforcement, which ensures no leak of privileges from one subject attribute to other unauthorized ones. Even though XACML supports default deny of access request in its rule scheme, the specifying and enforcing of such requirements have to be implemented in PEP by application.

III. Attribute Relations in Policy Machine

NIST has initiated a project in pursuit of a standardized access control mechanism referred to as the *Policy Machine* (*PM*) [8, 9, 10]. *PM* is based on the principle that the separation of access control policies from mechanisms [11] allows enforcement of multiple policies within a single, unified system so that access control rules from different authorities may be integrated with each other [12].

As shown in Figure 1, the *PM* architecture is composed of the *Policy Server* (*PS*) for PDP and PEP. *PS* includes *PS* processes and the *PS* database, and the *General Policy Management System* (*GPMS*). *PS* receives subject requests and performs the authorization process by referencing information from the *PS* 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 *PS* database.

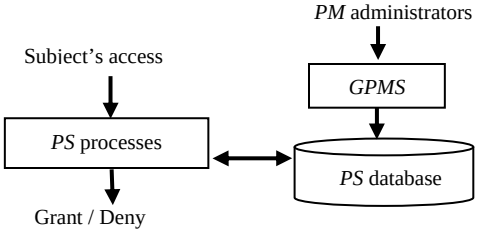

Figure 1. *PM* architecture

PM categorizes subjects (users), objects (resources), and their attributes into policy classes, and appropriately enforces subsets of the policies in response to a subject's access request. The following fundamental data sets for *PM* processing are stored in the *PS* database:

S: The set of *PM* subjects (users) under the *PM*'s control
SA: The set of subject attributes of *S*
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

Table 1 lists the *PS* authentication functions, and Figure 2 shows the *PS* database model.

Function	Description – mapping relation of
$ssa(s) = SA_S \subset SA$	a subject (user) *s* to a set of subject (user) attributes SA_S. i.e., *s* is assigned to those attributes
$sas(sa) = S_{sa} \subset S$	a subject attribute *sa* to a set of subjects S_{sa} that *sa* is assigned to
$sasa(sa) = SA_{sa} \subset SA$	a subject attribute *sa* to a set of inherited subject attributes; a subject assigned to *sa* will inherit the privileges of the subject attributes in SA_{sa} *(note,* $sa \in SA_{sa}$*)*
$saop(sa) = OP_{sa} \subset OP$	a subject attribute *sa* to a set of operations OP_{sa} that subjects in *sa* may perform
$opsa(op) = SA_{op} \subset SA$	an operation *op* to a set of subject attributes SA_{op} that may perform operation *op*
$opoa(op) = OA_{op} \subset OA$	an operation *op* to a set of object attributes OA_{op} that can be accessed by *op*
$oaop(oa) = OP_{oa} \subset OP$	an object attribute *oa* to a set of operations OP_{oa} that can be performed on *oa*
$sapc(sa) = PC_{sa} \subset PC$	a subject attribute *sa* to a set of policy classes PC_{sa} that covers *sa*. i.e. *sa* is covered by each policy class *pc* in that set
$pcsa(pc) = SA_{pc} \subset SA$	a policy class *pc* to a set of subject attributes SA_{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 subject 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

Function	Description – mapping relation of
$oao(oa) = O_{oa} \subset O$	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.

Table 1 PM Functions

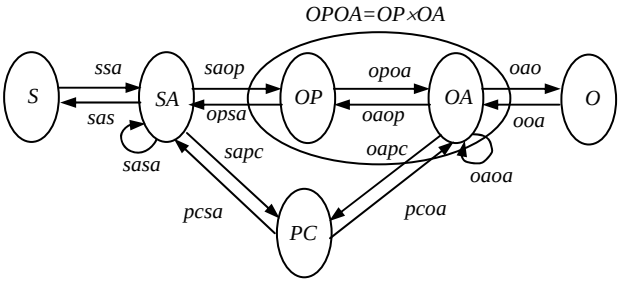

Figure 2. Set relations and functions of *PM*.

PM allows inheritance relations among subject attributes, and object attributes such that an element inherits the privileges from the elements that it is inherited from. The inheritance relation must not have cycles to be legitimate. A set of elements in an inheritance relation from one function to another function can be formally described by the union transitive closure of the two functions: $\cup_{y \in a(x)} b(y)$ denoted by the symbol "$x \rightarrow_{a,b}*$". For example, all inherited subject attributes SA_S of a subject *s* can be denoted by $s \rightarrow_{ssa,sasa}*$, 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*.

The atomic authorization process of *PM* is based on the above model and notation; the following definitions describe the *PS* authorization process and the states of *PM*.

Definition 1 – A tuple $T = \langle s, sa, op, pc, oa, o \rangle \in S \times SA \times OP \times PC \times OA \times O$ is an instance of the current configuration of a *PM*.

Definition 2 – The ***Grant_in_policy*(*s, op, o, pc*)** function decides if an access request (*s*, *op*, *o*) is satisfied in a policy class of *PM*, i.e. if there exists a tuple *T* as in Definition 1 such that *sa* and *oa* in the policy class *pc*, *s* is a member of *sa*, *o* is a member of *oa*, and *op* is in *saop*(*sa*), and *oa* is in *opoa*(*op*). Formally,

For $s \in S$, $op \in OP$, $o \in O$, $pc \in PC$, *Grant_in_policy*(*s*, *op*, *o*, *pc*) = True $\Leftrightarrow$
$\exists\ sa \in SA$ and $\exists\ oa \in OA$, such that
1) $sa \in (s \rightarrow_{ssa,sasa}*)$,
2) $oa \in (o \rightarrow_{ooa,oaoa}*)$,
3) $sa \rightarrow_{op} oa$,
4) $pc \in sa \rightarrow_{sapc,pcpc}*$, and
5) $pc \in oa \rightarrow_{oapc,pcpc}*$.

By Definition 1 and 2, *PM* only requires mapping the relations between elements to decide the permission of a subject's request. Through this mechanism, *PM* provides syntactic and semantic support of the AR specification.

IV. Application of attribute relations in AC models

This section demonstrates how *PM* specifies and enforces the MLS, HRBAC, SOD policies, and safety constraints by the AR assignments from the *PS* database and relation mapping functions. Subsection *A* demonstrates the implementation of a simple BLP and BI model, Subsection *B* shows the specification of SOD requirements, and Subsection *C* illustrates the enforcement of safety constraints by examples as described in Section 2.

A. Specification of MLS and HRBAC Policies

Information in MLS policy is typically controlled by assigning an AC element a *security class* (*label*) used to indicate privilege flow from the security class of *a* to the security class of *b*, which means subjects with security class *a* can also have the privileges of security class *b* (or *a* **dominates** *b*). In terms of object, object class *x* inherit object class *y* means any access privilege to class *x* can be also applied to class *y*.

PM can emulate MLS models by using its subject and object ARs. The subject security classes (labels) can be represented in *PM*'s *subject attributes*. Further, the objects security classes (labels) can be represented in *PM*'s *object attributes*, and the *subject attributes* are linked to the *object attributes* through *operations*. For example, to implement the BLP model, *PM may* construct two sets of relations for each of the *subject attributes* and *object attributes* as shown in the simple example of Figure 3. (For clear demonstration, different from Figure 2, we omit the *PC*, which should be linked by every subject and object attributes in the figure. We also omit *S*, and *O* sets, which can be any subjects and objects) The attribute with lower-case *r* in the attribute labels of *subject* and *object attributes* are for the read actions, which are required for the basic **confidential rule**. The attributes with lower-case *w* in the attribute labels are for the **star** property of BLP. In Figure 3, *TS* is *subject/object attribute* label for "*Top Secret*" subject/object class, *S* is for "*Secret*" class, and *C* is for "*Confidentia*l" class. *W* is for write action, *R* is for read action for each class (for example, *TSR* or *CW*). Each *subject/object belonging* to a class is assigned to both labels *w* and *r subject/object attribute* (for example, *TSr* and *TSw*). Assume that class *TS* dominates class *S*, and class *S* dominates class *C*; Subjects with the *Cw subject attribute* can write objects with the *object attribute Cw*, *Sw* and *TSw*. *Sw* can write *Sw* and *TSw*. *TSw* can only write *TSw*. *TSr* can read *TSr*, *Sr*, and *Cr*. *Sr* can read *Sr* and *Cr*. *Cr* can only read *Cr*. Note that a subject/object must be

assigned to the same *r* and *w* group of *subject/object attributes* (*TS*, *S* or *C)*. For example, a subject should be assigned to the *Cw subject attribute* if she was assigned to the *Cr* subject attribute and vice versa.

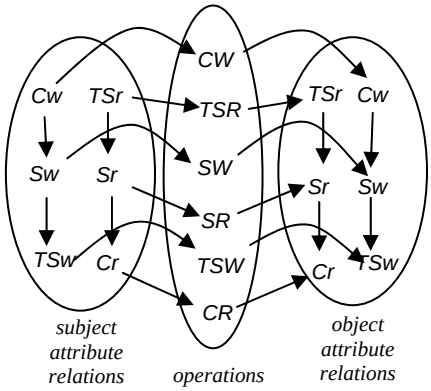

Figure 3. Simple Bell-La Padula Implementation.

There is no fundamental difference between the BI and BLP models. Both models are concerned with information flow in a lattice of security classes, with information flow allowed only in one direction in the lattice. The BLP model allows information flow upward in the lattice, whereas the BI model allows it downward. Since direction is relative, a system that can enforce one of these models can also enforce the other; this only requires a relatively straightforward remapping of attribute labels to invert the dominance relations needed[1].

A main feature of HRBAC is to allow a subject Role to inherit (therefore, dominate) access privilege from other roles. Similar to BLP and BI models, the hierarchy of privilege inheritance for HRBAC can be directly specified by the *subject attributes* of *PM*, such that if *subject attribute x* dominates *subject attribute y*, then subject with role *x* inherits all the access privilege of subjects with role *y*.

Figure 4.1 and 4.2 show example attribute assignments of MLS and HRBAC of a *PM* system states from subjects and objects point of views respectively. As the relation need only be assigned to directly related attributes, it only requires O(*n*) relation assignments if there are *n* classes for BLP or BI, or role inheritance relations for HRBAC. Thus, the complexity is many times more efficient compared to O(n^2) assignment statements in Section 2 *A*. Note that in this paper, we only focus on the efficiency and accuracy in specifying the AR required AC models and constraints. The process complexity (efficiency) for the enforcement of these models and constraints is either inevitable (e.g., collecting all the ARs in SOD models such as the examples in the next Subsection B) or algorithm/application dependent, thus, not discussed in this paper.

[1] It is often suggested that the BLP and BI model can be combined in situations where both confidentiality and integrity are of concern.

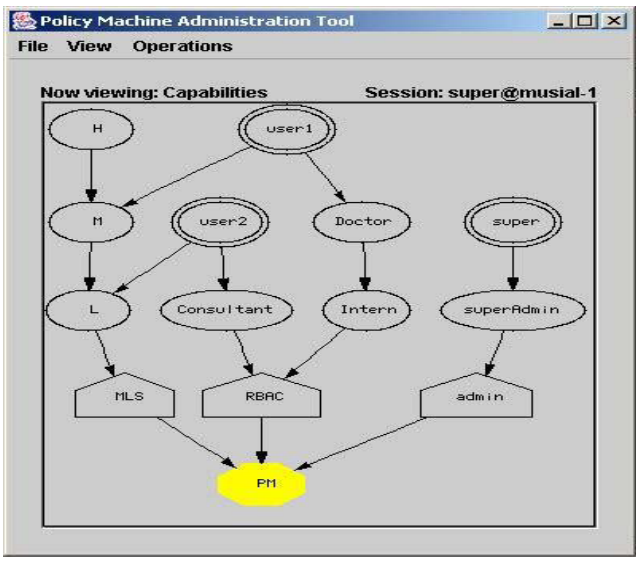

Figure 4. 1. Sample attribute relation assignments in PM from subjects' point of view

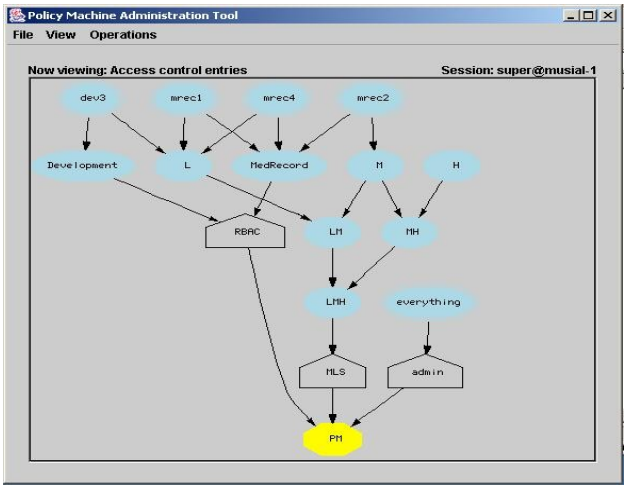

Figure 4. 2. Sample attribute relation assignments in PM from objects' point of view

B. Specification of Separation of Duty (SOD) Policies

SOD policies define constraint requirements for an AC system to ensure no access state can exceed some predefined system limitations. To enforce SOD without leaking access privileges as described in Section 2, it is necessary to maintain all subject/object attribute relations for any subject or object if multiple attribute assignments are allowed. Hence, in order to specify SOD policies in addition to the basic relation mapping functions in Table 1, *PM* needs to have the following extended functions to retrieve current mappings of ARs in the system:

- *sa_opoa(sa)* returns all (*op, oa)* pairs mapped to the *sa*.
- *opoa_sa*(*op*, *oa*) returns all *sa* that mapped to the (*op*, *oa*) pair.

Note that all the functions in Table 1 are direct mapping between attributes; functions return no attributes when the attribute is used only as a connection node for a dominate or

inheritance relation. The following examples *a)* and *b)* illustrate the specifications of the AC rules in *PM* for enforcing the two SOD policies samples described in Section 2 *B*.

a) The SOD constraint specifies that no subject should be assigned to more than *k privileges* of a given set. Note that when $k = 1$, this policy is a Privilege to Privilege Conflicts Policy (PPC), i.e. a set of privileges ($OP \times OA$) should not be assigned to the same subject. *PM* implements this policy by calculating the number of subject attributes the requesting subject is dominating or inheriting associated with the constrained privileges, and the number cannot exceed *k*. To implement this policy, before granting the access request by *Grant_in_policy*(*s*, *op*, *o*, *pc*), rules must be qualified is formally specified by *sa_opoa* and functions in Table 1 as the following:

$SoD_{PM} = \langle OPOA, k \rangle$, $OPOA = \{(op_1, oa_1), \ldots (op_n, oa_n)\}$, $1 \le k \le |OPOA|$, and

$$\forall s \in S \left(\left| (\cap\, sasa(sa \in ssa(s))\ sa\_opoa(sa)) \cap OPOA \right| \le k \right)$$

Tuple SoD_{PM} contains the set of restricted privileges *OPOA*, and limited number of privileges *k*. $\cap$ used in $sa\_opoa(sa) \cap OPOA$ is because a subject may be assigned to duplicated privileges through different ARs.

b) The SOD constraint specifies a set of privileges ($OP \times OA$) that no less than k number of subjects can perform all of them.(i.e., requires at least *k* number of subjects to perform all of them). *PM* specifies the rules for the SOD constraint by calcuating the minimum number of subjects who have (are associated with) all the privileges in the $OP \times OA$ set, which should equal or exceed k. To implement this policy, before granting the access request by *Grant_in_policy(s, op, o, pc*), rules must be qualified are formally specified by opoa_sa and Table 1 functions as the following:

$SoD_{PM} = \langle OPOA, k \rangle$, $OPOA = \{(op_1, oa_1), \ldots (op_n, oa_n)\}$

$MIN_{t=1..|COVERAGE|} |(unique(coverage_t), coverage_t \in COVERAGE = \{ S_1 \times \ldots S_i \ldots \times S_n$, *such that*

$$\forall (op_i, oa_i) \in OPOA, SA_i = opoa\_sa(op_i, oa_i) \left(\forall sa_{gi} \in SA_i, SA_{gi} = sasa(sa_{gi}) \left((\forall sa_{qgi} \in SA_{pi}, S_i = \cup\, sas(sa_{qgi})) \right) \right) \} | \ge k$$

sasa(*sa*) returns all the subject attributes dominated or inherited from *sa* including *sa* itself. Tuple SoD_{PM} contains the set of restricted privileges and the restricted *k* number of subjects. $S_1 \times \ldots \times S_n$ denotes the product set from S_1 to S_n. $MIN_{t=1..n}\, f(t)$ returns the minimum value from the functions *f* applied to different variables 1 .. *n*. The function *unique*(*D*) (or cardinal set of (*D*)) returns unique (non-duplicate) elements of the set *(D)*. For example, (op_1, oa_1) is accessible by subjects *a*, *b*, and *c*; (op_2, oa_2) is accessible by subjects *c*, and *d*; and (op_3, oa_3) is accessible by subjects *b*, *c*, and *e*. Thus, the all possible simultaneous accesses to all *OPOA* privileges are enumerated in *COVERAGE* = {(*a*,*c*,*b*), (*a*,*c*,*c*), (*a*,*c*,*e*), (*a*,*d*,*b*), (*a*,*d*,*c*), (*a*,*d*,*e*), (*b*,*c*,*b*), (*b*,*c*,*c*), (*b*,*c*,*e*), (*b*,*d*,*b*), (*b*,*d*,*c*), (*b*,*d*,*e*), (*c*,*c*,*b*), (*c*,*c*,*c*), (*c*,*c*,*e*), (*c*,*d*,*b*), (*c*,*d*,*c*), (*c*,*d*,*e*)}, where (*a*,*c*,*b*) = $coverage_1$, ……and here (*c*,*d*,*e*) = $coverage_{18}$, and $unique(coverage_1)$ = (*a*,*c*,*d*), $unique\_subject(coverage_2)$ = (*a*,*c*), and $unique\_subject(coverage_{14})$ = (*c*). And $MIN_{t=1..18}$ | $coverage_t$ |= | $coverage_{14}$ | = 1.

Examples *a* and *b* shows the SOD rule specifications by the *PM*'s standard *PS* functions based on the ARs. Without these functions, the complexity in specification is nontrivial.

In addition to enforce at run-time when computing the access decision, the inheritance constraints can also be implemented in the *PM*'s *GPMS* (in Figure 1), which restricts the assignments of attribute relations as required in this section.

C. Specifying Attribute Constraints for Safety

One of the safety requirements of AC systems is to ensure that the protected information can only be accessible by direct assignment to a group of permitted users. In *PM*, such requirement can be enforced by constraining the protected privilege of the information so that it can only be accessible through assigned subject attributes [13], which can neither inherit nor be inherited by other subject attributes. This is illustrated in Figure 5, where the protected privilege assigned to sa_y is not allowed to be inherited by sa_z, nor can sa_y inherit privilege from sa_x.

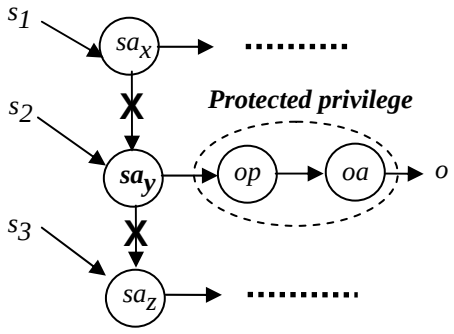

Figure 5. Example of no inheritance allowed

To enforce such constraint, the ***Grant_in_policy*** function in **Definition 2** needs to be extended to include three more parameters: sa_c, which represents the permitted subject attribute, op_c is the action, and oa_c is the object attribute of the protected privilege. All other parameters from the **Definition 2** remains the same. Thus, the modified function is *Grant_in_policy*(*s*, *op*, *o*, *p*, sa_c, op_c, oa_c) = True with the additional conditions

1.1) $sasa(sa_c) = \varnothing$,

1.2) $\forall\, sa_i \in SA,\ sa_c \notin sasa(sa_i)$

after rule *1*), and

6) saop(sa_C) = {op_C},

7) opsa(op_C) = {sa_C},

8) opoa(oa_C) = {op_C},

9) oaop(op_C) = {oa_C}.

after rule 5), thus, constraint that only sa_C is allowed to access oa_C by op_C.

Contrary to *A* above, another example of inheritance constraint is that a subject attribute sa_X has to inherit a specific attribute sa_Y in order to access any resources that sa_Y is permitted to access. The rational for such constraint is the case that sa_X (e.g. team member) is allowed to read a document after the sa_Y (e.g. team leader) is assigned to work on the document. Figure 6 illustrates an example for such assignments in *PM*, where subject attribute sa_X is not allowed to access *oa* by *op* except inherit the privilege from the subject attribute sa_Y.

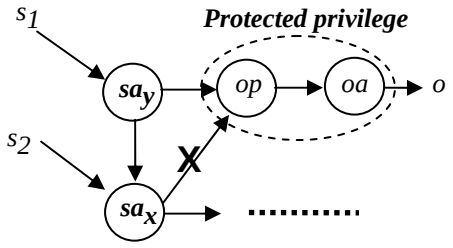

Figure 6. Example of specific inheritance required

To enforce such constraint, the ***Grant_in_policy*** function in **Definition 2** needs to be extended to include four more parameters sa_X, sa_Y, op_C, and oa_C, where op_C, and oa_C are the protected action and object attribute. The rest of the parameters from **Definition 2** remain the same. The modified function is *Grant_in_policy*(*s*, *op*, *o*, *p*, sa_X, sa_Y, op_C, oa_C) = True with three additional conditions to ensure that 1, sa_X inherits sa_Y, 2, there is no assignment from sa_X to op_C, and 3, op_C is assigned to oa_C as following:

6) $sa_Y \in sasa(sa_X)$,

7) $sa_X \rightarrow_{op} oa$ = False,

8) $oa_C \in opoa(op_C)$.

Note that the additional conditions in the modified *Grant_in_policy* function enforce the policy by checking the conformance of the safety policy. The policy can also be enforced by implementing these conditions in the *GPMS* of *PM*.

V. Related Work

In general, AC models and mechanisms can be expressed by *graph-based approaches* or *logical approaches.* In *graph-based approaches*, access control models are modeled through graphs whose state changes upon the application of graph transformations. In *logical approaches*, such AC elements are expressed through logic programs according to the semantics chosen for these programs. Although the two approaches have almost the same expressive power, they are complementary with respect to the purpose of use [14].

[15] proposed a Flexible Access Control Model (FACM) as a graph approach to simplify the specification and the verification of safety via constraints, that is, with expressions able to specify the safety requirements of any access control configuration. The proposed model provides user-friendly notation and presentation of ARs and constraints. However, the main usage of the graph representation is to help in the specification, design, rather than as a pure computational model, unlike *PM*, which provides computational functions in the *PS* server, and allows policy authors to specify AC rules by directly mapping ARs into rules semantic.

[12] proposed a logical approach named A Logical Framework for Reasoning about Access Control Models (ACMP) based on the C-Datalog program with the expressive power that can model a variety of access control policies. The framework is flexible in representing subjects, objects, privileges, hierarchies, and sessions, as well as positive and negative authorizations, which provide a precise mathematical foundation for reasoning about ARs. However, in addition to its logical programs are not being intuitive to most users, ACMP does not provide views of access instance and relations between attributes, unlike *PM*, which allows administrators to check/filter the relations at the point of view of any selected access element. This capability otherwise requires tracing through AC rules, and it is hard to achieve with the increased number of entries in the ACMP program.

VI. Conclusion

The flexibility and expressiveness of XACML make it complex to work directly with some AC mechanisms. Specifying ARs in XACML calls for completely specified relations for each and every directly or indirectly related attribute, thus produces a highly verbose document even if the actual policy rules are trivial. (In general, AC policies expressed in an abstract language are difficult to create and maintain by AC policy administrators [16].). Because, *PM* is not a language, it is free from the syntactic and semantic complexity of a language. When describing hierarchical relations between attributes or policies, *PM* only requires adding links between them, therefore, avoiding the time delays due to the sequence of overhead algorithms. In supporting the enforcement of SOD policy constraint rules, *PM* provides an infrastructure that allows the efficient specification of rules to collect the attributes for the policy.

In addition, ACPT allows the flexibility in define access granting constraints to support safety requirement of an AC system. As *PM* possesses both the graphical and logical properties, it not only provides the syntactic and semantic support for the implementation of AR based policies, thus simplify the required attribute engineering in some AC policies, but also has a WYSIWYG (What You See IS What You Get) graphic user interface (Figure 4) 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. This feature is especially important when adding and deleting rules in the AC policies.

References

[1] Bell D.E. and Lapadula L. J., "Secure Computer Systems: Mathematical Foundations and Model," M74-244, MITRE Corp., Bedford, Mass., 1973 (also available as DTIC AS-771543).

[2] Biba K. J., "Integrity Considerations for Secure Computer Systems," ESD-TR-76-372, USAF Electronic Systems Division (also MTR3153, MITRE Corp.), Bedford, Mass., April 1977.

[3] NCSC, "Trusted Computer System Evaluation Criteria," National Computer Security Center, 1985.

[4] Ferraiolo D. F., Cugini J. A., and Kuhn D. R., "Role-Based Access Control (RBAC): Features and Motivations," Proc. of the 11th Annual Conference on Computer Security Applications, IEEE Computer Society Press, Los Alamitos, Calif, pp 241-248, 1995.

[5] Jajodia S., Samarati P., and Subrahmanian V. S., "A logical language for expressing authorizations," Proc. IEEE Symp. On Research in Security and Privacy, Oakland, Calif, pp 31- 42, May 1997.

[6] OASIS, "Extensible ACCess Control Markup Language (XACML), TC," http://www.oasisopen.org/committes/tc_home.php?wg_abbrev=xacml

[7] Yague, I. M. "Survey on XML-Based Policy Languages for Open Environments", Jouranl of Information Assurance and Security (JIAS), Volume 1, Issue 1, March 2006.

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

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

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

[11] Jajodia S., Samarati P., and Subrahmanian V. S., "A Logical Language for Expressiing Authorizations," Proc. IEEE Symp, Oakland, California, 1997.

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

[13] Hu C. V., Kuhn R., Xie T., Hwang J., "Model Checking for Verification of Mandatory Access Control Models and Properties", International Journal of Software Engineering and Knowledge Engineering (IJSEKE) regular issue volume V20N5, September 2010.

[14] Bertino et al, "A Logical Framework for Reasoning about Access Control Models," ACM Transactions on Information and System Security, Vol. 6, No. 1, pp 71–127, February 2003.

[15] Jaeger, T and Tidswell, "Practical Safety in Flexible Access Control Models". ACM Trans. Inform. Syst. Secu. 4, 2 pp 158-190, May, 2003.

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

Author Biographies

Vincent C. Hu is a computer scientist in the computer security division of the National Institute of Standards and Technology (NIST). His interests include access control, grid systems, and quantum computing. He designed and developed the Access Control Policy Tool (ACPT) by applying model verification and combinatorial array testing techniques; He currently works on the evaluation metrics for access control systems. Vincent received his Ph.D. degree in computer science from the University of Idaho at Moscow, Idaho in 2002.

David D. Ferraiolo is a computer scientist and acting manager of System and Emerging Technologies Security Research Group in the computer security division of the National Institute of Standards and Technology (NIST). His primary technical interests are in information security, and access control systems. He co-developed the role based access control model (RBAC) used throughout industry, and currently is leading the effort of developing the Policy Machine. David received a combined B.S. in computer science and mathematics from the State University of New York at Albany in 1982.

Serban I. Gavrila is a computer scientist with the Computer Security Division of National Institute of Standards and Technology (NIST). His primary technical interests are in access control systems. He enjoys Java programming and has co-developed distributed systems for controlling access to computer resources, among them the Policy Machine. Serban received his B.S. degree in computer science from the University of Bucharest at Bucharest, Romania in 1972.