

# Using Semantics for Automatic Enforcement of Access Control Policies among Dynamic Coalitions\*

Janice Warner  
MSIS Department and CIMIC  
Rutgers University  
janice@cimic.rutgers.edu

Vijayalakshmi Atluri  
MSIS Department and CIMIC  
Rutgers University  
atluri@rutgers.edu

Ravi Mukkamala  
CS Department  
Old Dominion University  
mukka@cs.odu.edu

Jaideep Vaidya  
MSIS Department and CIMIC  
Rutgers University  
jsvaidya@rbs.rutgers.edu

## ABSTRACT

In a dynamic coalition environment, organizations should be able to exercise their own local fine-grained access control policies while sharing resources with external entities. In this paper, we propose an approach that exploits the semantics associated with subject and object attributes to facilitate automatic enforcement of organizational access control policies while resource sharing occurs among coalition members. Our approach relies on identifying the necessary attributes required by external users to gain access to a specific organizational object (or service). Specifically, it consists of extracting user attribute sets that semantically match with the attributes of the objects for which a role has permissions. This relies on a closer examination of why a user is assigned a specific role. These attribute sets are first pruned based on their significance in characterizing a role, which are then checked against those submitted by an external user to decide whether to allow or deny access to the specific object. While our goal in this paper is to support coalition based access control, the proposed approach can also aid in automating the process of role engineering.

## Categories and Subject Descriptors

D.4.6 [Operating Systems]: Security and Protection—*Access controls*; H.3.5 [Information Storage and Retrieval]: On-line Information Services—*Data Sharing*

## General Terms

Security

\*The work is supported in part by the National Science Foundation under grant IIS-0306838.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SACMAT'07, June 20-22, 2007, Sophia Antipolis, France.  
Copyright 2007 ACM 978-1-59593-745-2/07/0006 ...\$5.00.

## Keywords

Access Control, Semantics, RBAC, Coalitions, Collaboration

## 1. INTRODUCTION

Today, there is an increasing need for dynamic, efficient and secure sharing of resources among organizations. This is driven by a number of applications including emergency and disaster management, peace keeping, humanitarian operations, or simply virtual enterprises for commercial purposes. Typically, resource sharing is done by establishing alliances and collaborations, also known as *coalitions*. Due to the nature of the applications, the coalitions are often *dynamic* where entities may join or leave the coalition in an ad-hoc manner.

In a dynamic coalition environment, participants (e.g., users, systems) of an organization may need to gain access to resources (both data and services) of other organizations to accomplish the task at hand. However, this should not mean that access to resources is now freely available to everyone across all organizations. Even if the coalition was formed in a quick and dynamic manner, organizations should be able to exercise their own fine-grained access control policies while sharing resources with external entities. Although this may be accomplished by means of traditional access control and authentication mechanisms, they are administratively difficult when the partnerships and interactions are short-lived and constantly changing. This is because, using traditional access control would require explicit specification of authorizations for individual users within each member organization and changes would need to be painstakingly administered.

Access control policies define who has access to what resources (e.g., data and services) and what activities (e.g., read, write, execute) they can perform on those resources. In closed systems, “who” is defined on the basis of a system user id. For dynamic, ad-hoc coalitions, “who” must be defined more abstractly based on attributes of the user rather than a user id since there is no *a priori* knowledge of the specific individuals in the coalition.

Let us assume that organizations use *role-based access control* (RBAC) as their basic policy framework. Given a

set of security policies, organizations implement them by creating roles (nothing but a set of permissions, which in turn are object-privilege pairs) and then associating users to roles. A closer examination of why a role is given a permission and why a user is assigned to that role reveals that there are some inherent semantics associated with the assignments. Specifically, given an object and a set of users who have permission to access that object, we can examine the semantics of the object and user attributes and determine what user attributes are *semantically relevant* to the object attributes and thus to the job function being performed. Although these required attributes are not explicitly specified in defining roles or specifying user-role mapping, job functions within an organization and skills of the users are implicitly used. Schaad et al. [16] presented a case study of Dresdner Bank where job function is explicitly used. Each job function at Dresdner is given a specific code. The role to which a person is assigned at Dresdner is based on their job function, their official position (level within the organization), and their organizational unit.

In an ad-hoc coalition, if a user of another organization (say, external user) would like to access resources within this organization (internal resources), one has to ensure that none of the security policies of this organization are violated. (Note that we assume that access by external individuals is limited to the “read” operation.) In order to accomplish this, one must ensure that the external user possesses the same (or semantically similar) attributes possessed by an internal user that are required to gain a permission through a role. Our earlier work [2, 19] proposed a coalition based access control model, which examines all of the credential attributes of all the users of a role. This, we believe, is overly restrictive since all the attributes are not relevant for gaining a permission. Instead, we believe that one can exploit semantics of user and object attributes in identifying relevant attributes. Many of these attributes are implicitly associated with job functions which in turn are associated with a role. For example, an auditor John is allowed to access a file containing Acme’s financial data because he is assigned to the task of “auditing of Acme’s accounting data,” and John possesses credentials that show he has a degree in accounting and is CPA certified. There is an inherent semantic relationship between the attributes of John (including the task to which he is assigned) and the object semantics. Furthermore, this is true not only for John, but also for the other users – those who have access to the object also have similar attributes.

In support of this idea, we first formalize the notion of the necessary semantics of the attributes. We define the attributes and identify the relationships that facilitate mapping between user attributes, job functions or roles and object attributes. We then specify how attributes can be matched. After determining if the candidate attributes are *semantically relevant*, we ensure that attribute requirements are also *significant* or unlikely to be held by those that should not be assigned to a role. Once all relevant and significant attributes are identified, we use the role hierarchy to test whether users with a certain set of credential attributes are given maximal access to the objects they need while restricting others from accessing objects not needed for their roles. Finally, we describe how external user attributes presented as credentials can be compared with the attribute requirements to determine whether the external user should

be given the requested access.

Not only does the proposed approach aid in forming ad-hoc coalitions, but it can also be useful in automating assignment of internal users to roles during the process of role engineering. Essentially, it suggests a way to assign users to roles based on their attributes. This is a first step toward automating role assignment. A NIST report [7] suggests that this is one of the expensive tasks before RBAC can be deployed and found that it is a very difficult task where organizations spend substantial effort (time and money).

This paper is organized as follows. Section 2 presents the necessary framework comprising of user and object attributes, role hierarchies and concept hierarchies. Section 3 describes how semantically relevant user attribute requirements for the assignment to a role are extracted. Section 4 describes how the attribute requirements are used in the process of providing access control in an ad-hoc coalition. Section 5 reviews the related work and compares it to our approach. Finally, Section 6 concludes the paper.

## 2. FRAMEWORK

This section introduces the preliminaries required for our semantics based approach to coalition based access control. In particular, we review coalitions, role based access control, user and object attributes, semantic concepts and their hierarchies, as well as semantic linking among attributes.

### 2.1 Coalition

Organizational entities can form an ad-hoc coalition. An instance of a coalition is represented by  $\chi$  and can consist of two or more members. Every organizational entity is represented by its entity identifier, which must be globally unique. For any coalition instance  $\chi_i$ , the member organizations are represented by  $\chi_i = \{eid_1, \dots, eid_n\}$  where  $n$  is the number of collaborating organizational entities. Within the organizational entities, users will have user identifiers,  $u_j$ . However, these user identifiers will not be globally understood and thus do not need to be globally unique.

EXAMPLE 1. Let us say that a coalition comprising of two companies HelpU Inc. (HU) and LM Systems (LM), represented by  $\chi_{47095} = \{HU433, LM978\}$ , exists. HU is a software sub-contractor to LM. Assume HU has two software engineers, identified internally as lwwerner and jmarin, working on component 1’s software for LM’s Project Blue.

### 2.2 Role Based Access Control (RBAC)

Since our framework assumes that RBAC is in place within each organization participating in a coalition, we review the definition of RBAC.

DEFINITION 1. [RBAC] RBAC has been formally defined by NIST[6] as follows:

- $U, R, OPS$  and  $O$  are sets of users, roles, operations and objects respectively.
- $UA \subseteq U \times R$  is a many to many user to role assignment relation.
- $P$  (the set of permissions)  $\subseteq \{(op, o) | op \in OPS \wedge o \in O\}$
- $PA \subseteq P \times R$  is a many to many permission to role assignment relation.

- $assigned\_users(r) = \{u \in U | (u, r) \in UA\}$ , the mapping of role  $r$  onto a set of users.
- $assigned\_permissions(r) = \{r \in R | (u, r) \in UA\}$ , the mapping of role  $r$  onto a set of permissions.
- $assigned\_objects\_per\_permission(r, p) \rightarrow \{o \in O | p = (op, o)\}$ , the permission-to-object mapping which gives the set of objects associated with permission  $p$  for a given role.
- $assigned\_objects(r) \rightarrow \cup\{assigned\_objects\_per\_permission(r, p) | p \in assigned\_permissions(r)\}$ , the permission-to-object mapping which gives the set of all objects associated with any of the permissions assigned to role  $r$ .
- $RH \subseteq R \times R$  is a partial order on  $R$  called the role hierarchy or role dominance relation.

## 2.3 Attributes

We assume that both users and objects are associated with specific attributes. An *attribute* consists of a name and a value pair,  $(a_i : v_i)$  and is referred to by its name,  $a_i$ . The set of all attributes defined for an organization is  $\Lambda$ .

### 2.3.1 User Attributes

User attributes that may be semantically relevant to objects describe what the user is capable of doing, has done or is assigned to do. Some of these attributes may be drawn from *certifiable* credentials [8] possessed by the users indicating, for example, that a user has completed a degree or a training program. Others may be explicitly assigned internally by the organizations to indicate, for example, experience the user has had or their current assignments. The set of all user attributes is denoted as the *user attribute base* (UAB). We use  $ua_i = \{(a_i : v_i), (a_j : v_j), \dots\}$  to denote all the attributes associated with a specific user  $u_i$ .

EXAMPLE 2. Tom, who works at LM, has the following attribute set:  $ua_{Tom} = \{(hasDegree:bachelors), (performsJob:software), (assignedTo:projectBlue), (hasExpertiseIn:java), (officeLoc:NVC1)\}$ .

### 2.3.2 Object Attributes

Objects attributes are either explicitly defined or automatically derived using text analysis as in [15]. Object attributes might be classified by keywords or content, in terms of their type (e.g., executable, spreadsheet), attributes of their author/owner, or in relationship with each other. Object attributes include object id ( $o_i$ ), attribute name ( $a_i$ ) and attribute value ( $v_i$ ). The set of all object attributes is denoted as the *object attribute base* (OAB). We use  $oa_i = \{(a_i : v_i), (a_j : v_j), \dots\}$  to denote the set of object attributes associated with a particular object  $o_i$ .

EXAMPLE 3. LM has an object “Component 1 software” ( $C1$ ) that has the following attribute set:  $oa_{C1} = \{(hasContent:java), (hasContent:financial), (hasContent:software), (createdUnderProject:Blue)\}$ .

## 2.4 Concept Hierarchy

We employ concept hierarchies to *link* the different attributes and compare their values. A concept hierarchy is

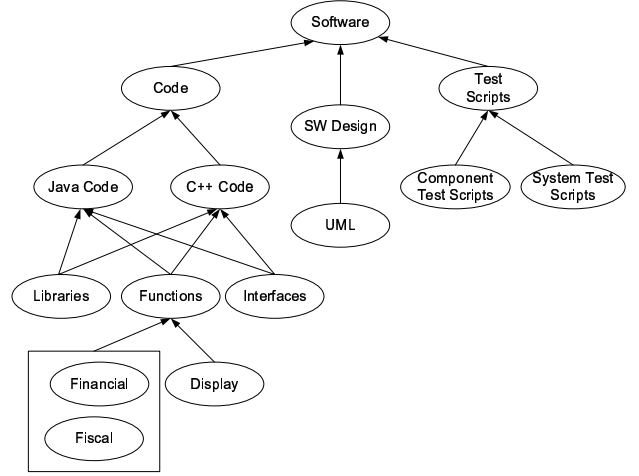


Figure 1: Example Concept Hierarchy for Concept Software

a graphical notation for representing knowledge using interconnected nodes and arcs. In a concept hierarchy, a more specific concept(s) is represented as a descendant(s) of its more general concept(s).

A concept hierarchy,  $C_i$ , consists of a partially ordered ( $\prec$ ) set of concepts. Given two concepts  $c_1, c_2 \in C_i$  we consider the following four possible relationships between them:  $subClass(c_1) = c_2$  for the relationship where  $c_2$  is a more specialized concept;  $eq(c_1) = c_2$  where  $c_1$  and  $c_2$  are equivalent concepts (synonyms);  $sup(c_1) = c_2$  where  $c_2$  is the next more general concept than  $c_1$  (i.e., separated by only one link in the concept hierarchy); and  $com(c_1) = (c_2)$  where  $sup(c_1) = sup(c_2)$ . This latter relationship means that  $c_1$  and  $c_2$  are compatible. Note that several other relationships among concepts are defined in [1]. In this paper, we utilize only the relationships specified above. Note that the concepts can be either the attribute names or their values.

Figure 1 shows an example concept hierarchy for the general concept **software**. From the figure,  $sup(\text{Financial}) = \text{Functions}$  and  $eq(\text{Financial}) = \text{Fiscal}$  are examples of relationships. Note that when two concepts are synonyms they are enclosed in a box. Finally,  $com(\text{Financial}) = \text{Display}$  indicating that these two concepts representing have a common parent and thus represent more specific aspects of the same general concept.

DEFINITION 2. We say an attribute  $a_i$  is associated with (or belongs to) a concept hierarchy  $C_j$ , denoted as  $a_i \rightarrow C_j$  if its value  $v_i$  is an included concept in  $C_j$ .

We assume that every attribute in  $\Lambda$  with a non-numerical attribute value must be associated with one or more concept hierarchies. For example, the values for user attributes **performsJob** and **hasExpertiseIn** are concepts in the concept hierarchy for the concept **software**. Likewise, the values for the object attributes **hasContent** or **hasKeyword** are also concepts in the concept hierarchy for the concept **software**. Note that the degree of specialization of a node increases with its distance from the root. For example in Figure 1, **Java Code** is more specialized than **Code** which in turn is more specialized than **software**.

## 2.5 Attribute Linking

In order to compare two attributes and their associated values, we need to know if the attributes use the same vocabulary. We do this by associating attribute names with concept hierarchies where the association indicates the concept hierarchy from which the attribute values are drawn. Attributes are comparable if they are associated with attribute values that are from the same concept hierarchy. We say that such comparable attributes are *linked*.

We distinguish two types of linking: (1) *referential linking* that links a user attribute name and an object attribute name when associated with the same concept hierarchy; (2) *synonymous linking* that links two user attribute names that are synonyms.

Whether it is referential or synonymous linking, we use  $\top(a_i, a_j)$  to represent the linking between two attribute names  $a_i$  and  $a_j$ . The linking relationship is transitive. That is, if  $\top(a_i, a_j)$  and  $\top(a_j, a_k)$ , then  $\top(a_i, a_k)$ .

**DEFINITION 3.** We say  $\top(a_i, a_j)$  is a *referential linking* if  $a_i$  is a user attribute and  $a_j$  is an object attribute such that  $a_i \rightarrow C_i \wedge a_j \rightarrow C_i$ .

**DEFINITION 4.** We say  $\top(a_i, a_j)$  is a *synonymous linking* if both  $a_i$  and  $a_j$  are user attributes such that  $eq(a_i) = a_j$ .

We use referential linking to compare user and object attribute values to extract the necessary attribute-value pairs for a role, which will be further described in Section 3. Synonymous linking is used to link a user attribute-value requirement to an attribute-value received from an external user in a credential when the attribute name in the credential is not exactly that used by the resource owning organization but is a synonym of an attribute name that is used. This will be further described in Section 4.

## 3. GENERATING REQUIRED USER ATTRIBUTES FOR ROLES

As discussed in Section 1, our approach to facilitate coalition based access control is based on determining the set of required user attributes to access a specific object. In this section, we present our approach to extracting that required set of user attributes. Essentially, our approach is to determine user attribute-value pairs that characterize a role. This set is the required *candidate attributes of users* (*cau*) for membership in a role. It is our assertion that users should have attributes that represent their job functions and that these attributes should match attributes of objects needed to perform their job function. Specifically, we say that user attributes should be semantically matched to object attributes. We define such a *semantic match* as follows:

**DEFINITION 5.** [Semantic Match] There exists a *semantic match* between a user attribute  $a_i$  and an object attribute  $a_j$  iff  $(\top(a_i, a_j)) \wedge ((eq(v_i) = v_j) \vee (subClass(v_j) = v_i))$ .

**EXAMPLE 4.** For example, the object “Component 1 Software” has the attribute **hasContent** which draws its values from the concept hierarchy for **software** shown in Figure 1. A user, Elise, has an attribute **hasExpertiseIn** whose values are also drawn from the concept hierarchy for **software**. Thus, the object attribute and the user attribute are linked.

---

### Algorithm 1 Determining Required User Attributes for a Role

---

**Require:**  $OAB, UAB, r$

- 1:  $cao^r \leftarrow \{\cup aoi | o_i \in assigned\_objects(r)\}$
- 2: **for** each  $a_i \in assigned\_users(r)$  **do**
- 3:    $cau_i^r \leftarrow au_i$
- 4: **end for**
- 5: **for** each  $cau_i^r$  **do**
- 6:   **while** more attributes  $a_j \in cao^r \wedge (\text{SemanticMatchFound} = \text{FALSE})$  **do**
- 7:     **if**  $(\top(a_i, a_j)) \wedge ((v_i = v_j) \vee (v_i = sup(v_j)))$  **then**
- 8:        $\text{SemanticMatchFound} = \text{TRUE}$
- 9:     **end if**
- 10:   **end while**
- 11:   **if**  $\text{SemanticMatchFound} = \text{FALSE}$  **then**
- 12:     Remove  $a_i$  from  $cau_i^r$
- 13:   **end if**
- 14: **end for**
- 15:  $cau^r \leftarrow \cup \{cau_i^r\}$
- 16: Let each element  $i$  in  $cau^r$  be  $as(i)$
- 17: **for**  $i = 0$  to  $n$ ,  $i++$  **do**
- 18:   **for**  $j = i + 1$  to  $n$ ,  $j++$  **do**
- 19:     **if**  $(\text{all } a_i \in as(i)) = (\text{all } a_j \in as(j))$  **then**
- 20:       **for** all  $v_k \in as(i) \wedge$  all  $v_l \in as(j)$  **do**
- 21:         **if** all  $v_k = com(v_l) \vee$  all  $v_k = sup(v_l)$  **then**
- 22:         add  $v_k$  to  $as(j)$
- 23:         remove  $as(j)$  from  $cau^r$
- 24:       **end if**
- 25:       **if** all  $v_k = com(v_l) \vee$  all  $v_l = sup(v_k)$  **then**
- 26:         add  $v_l$  to  $as(i)$
- 27:         remove  $as(j)$  from  $cau^r$
- 28:       **end if**
- 29:     **end for**
- 30:   **end if**
- 31:   **end for**
- 32: **end for**
- 33: **for** each attribute-value set  $as(i) \in cau^r$  **do**
- 34:   calculate  $\phi_c$
- 35:   **if**  $\phi_c < 100^1$  **then**
- 36:     remove  $as(i)$  from  $cau^r$
- 37:   **else**
- 38:     **for** each attribute-value pair  $(a_i : v_i) \in as(i)$  **do**
- 39:       calculate  $\phi_a$
- 40:       **if**  $\phi_a < 5^2$  **then**
- 41:         recalculate  $\phi_c$  for  $as(i) - (a_i : v_i)$
- 42:         **if**  $\phi_c > 100$  **then**
- 43:         Remove  $a_i$  from  $as(i)$
- 44:       **end if**
- 45:     **end if**
- 46:   **end for**
- 47:   **end if**
- 48: **end for**
- 49: return( $cau^r$ )

<sup>1</sup> The attribute set threshold was chosen to be 100.

<sup>2</sup> The attribute threshold was chosen to be 5.

---

Now suppose the value of the object attribute, **hasContent**, is **functions**. Suppose also that the value of Elise’s **hasExpertiseIn** attribute is **financial**, then there is a semantic match between the two attributes because Elise’s attribute value is a subclass of the Component 1 Software’s attribute value in the software concept hierarchy. Note that

if Elise’s attribute value for `hasExpertiseIn` had also been `functions`, there would have also been a semantic match between the two attributes. However, if Elise’s attribute value for `hasExpertiseIn` had been `software`, there would be no semantic match because `subClass(function) ≠ software`. The inclusion of `subClass` is to allow, for example, users with `hasExpertiseIn` in Java code to access objects with attributes `Code` or `Software`.

However, there is more to the process than simply performing semantic matches. To determine attribute requirements for a role, we use the following four step process, where the first step is the semantic matching described above.

**STEP 1. Discovering User Attributes that are Semantically Related to Object Attributes:** For each user who is currently a member of the role, consider each of their attribute-value pairs contained in the UAB. This is the *candidate attribute-value pair set* for that user. For each attribute-value pair, determine if there is a semantic match between it and an attribute of any of the objects to which the role has privileges. If there is no semantic match, remove the attribute-value pair from consideration by taking it out of the candidate-value pair set for that user.

**STEP 2. Merging candidate attribute-value pair sets:** This part of the process involves combining user candidate attribute-value pair sets that are the same or similar. The similarity is determined by finding pairs that have the same attribute names but different attribute values. The values are compared and if there is a relationship as per the concept hierarchy associated with the values, they may be merged.

**STEP 3. Pruning attribute-value pair sets by assessing significance of attribute-value pairs:** For each candidate attribute value pair set, determine if it is *significant* for that role. An attribute-value pair is significant if it is a characteristic of a high percentage of members of the role and is a characteristic for only a small number of non-members of the role. Only those attribute-value pairs that are significant for the role are kept for further consideration. The result is a set of candidate attribute-value pair sets that are *significant* to the role.

**STEP 4. Checking Attribute Requirements Across Roles:** Finally, the candidate attribute-value pair sets of all roles are tested if they obey certain rules when considering the role hierarchy, which involves ensuring that related roles meet certain attribute requirements rules. This is a semi-automated process. Note that new attribute-value pairs may be added during this step.

The details of the above four steps are presented in the following subsections.

### 3.1 Semantic Matching of User Attributes to Object Attributes

This step involves extracting and semantically matching user attributes (and their values) for role members to object attributes (and their values) for objects for which the role has some permission. Lines 1 to 15 of Algorithm 1 show how this is done and is explained in this subsection.

To derive the attribute requirements for membership in role  $r$ , we need the set of objects for which permissions are assigned to the role,  $assigned\_objects(r)$  and the set of users assigned to the role,  $assigned\_users(r)$  as per Definition 1.

**DEFINITION 6. [Candidate Object Attributes]** The *candidate object attribute set* for a role  $r$ ,  $cao^r = \{\cup o_i | o_i \in$

$assigned\_objects(r)\}$ .

Essentially, for each of the object  $o_i \in assigned\_objects(r)$ , we extract from the OAB all attributes associated with that object,  $ao_i$ . The union of all the attributes for all the objects in  $assigned\_objects(r)$  make up the candidate object attribute set  $cao^r$  for role  $r$ .

**DEFINITION 7. [Candidate User Attributes]** The *candidate user attribute set* for user  $u_i$  of role  $r$ ,  $cau_i^r = \{a_j | a_j \in au_i \wedge u_i \in assigned\_user(r) \wedge a_j \text{ has a semantic match to some } o_k \in cao^r\}$ .

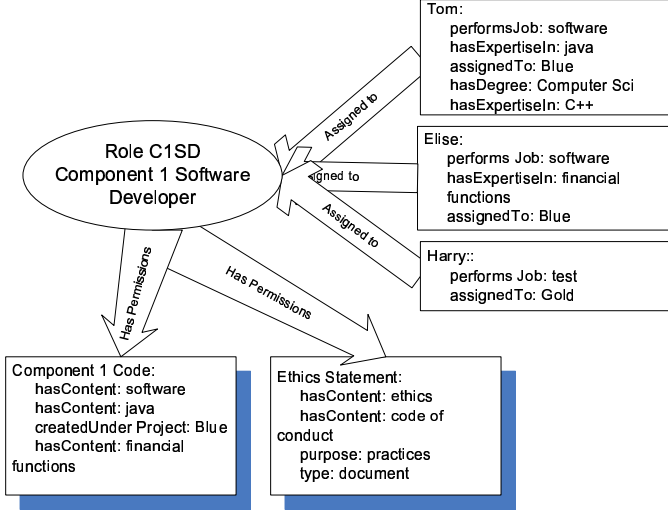
For each of the users  $u_i \in assigned\_users(r)$ , we extract a user attribute set,  $au_i$ . For each of the attributes  $a_i \in au_i$ , we check whether there is a semantic match (as per Definition 5) between it and any of the object attributes in  $cao^r$ . If there is a semantic match, the attribute  $a_i$  and the value  $v_j$  of the matching object attribute are added to the candidate user attribute requirement set for user  $u_i$  under role  $r$ ,  $cau_i^r$ . If a user has no semantic matches to any of the object attributes, that user is flagged since they may be incorrectly assigned to the role. Likewise, if there is an object whose attributes have no semantic matches to any of the users assigned to the role, it is also flagged as an inappropriate object for the role.

**DEFINITION 8. [Candidate Role Attributes]** The *candidate role attribute set* for role  $r$ ,  $cau^r = \{\cup cau_i^r | u_i \in assigned\_users(r)\}$ .

Thus, we have a set of candidate attribute-value pairs,  $cau_i^r$  for every user  $\{u_i, u_j, u_k \dots\}$  who is assigned to role  $r$ . The union of these sets is the candidate role attribute set  $cau^r$  for role  $r$ . Some of the individual attribute-value pairs sets will be exactly the same. That is, some users assigned to role  $r$  are likely to have exactly the same attribute-value pairs,  $cau_i^r$ . To eliminate duplicates, the resulting  $cau^r$  has only  $n$  elements in it where  $n$  is the number of unique attribute-value pair sets and  $n$  is less than or equal to the number of users who are members of role  $r$ . To distinguish these sets within the  $cau^r$ , we identify them by the notation  $as(i)$  where  $i$  goes from 1 to  $n$ .

**EXAMPLE 5.** Figure 2 shows a role, *Component 1 Software Developer*, which is a specialization of the role *Software Developer* (see Figure 3). This role has three members, Tom, Elise, and Harry and is assigned permission for all files for *Component 1 code*. The attributes of Tom, Elise, and Harry as well as the *Component 1 Code* object are shown in the figure. When comparing Tom’s attributes to those of *Component 1 Code*, there are referential links between the user attribute `performsJob` and the object attribute `hasContent`, the user attribute `hasExpertiseIn` and the object attribute `hasContent`, and the user attribute `assignedTo` and the `createdUnderProject` object attribute. Therefore, the candidate attribute set when just looking at Tom as the first user assigned to the software developer role would be  $cau^{SD} = \{(\text{performsJob:software}), (\text{assignedTo:Blue}), (\text{hasExpertiseIn:java})\}$ . Tom’s attributes, `hasDegree: Computer Science` and `hasExpertiseIn:C++`, were removed from consideration since there is no semantic match between those attributes and attributes of the objects.

For Elise, there is a match between her attribute `performsJob` and the object attribute `hasContent`, the user



**Figure 2: Component 1 Software Developer Role, Members, and Permissions**

attribute `hasExpertiseIn` and the object attribute `hasContent`, the user attribute `assignedTo` and the `createdUnderProject` object attribute.

Harry has no attributes which match the attributes of the objects. This should be a concern since it appears that Harry may not belong to the role. Harry’s lack of semantically matched attributes should be flagged for further analysis. The empty set is not added as a candidate.

Therefore, the candidate attribute user sets for the *Component 1 Software Developer role (C1SD)* can be expressed as:  $cau^{C1SD} = \{ \{ (performsJob:software), (assignedTo: Blue), (hasExpertiseIn:java) \}; \{ (performsJob:software), (hasExpertiseIn:financial\ functions), (assignedTo:Blue) \} \}$ . Note that the “;” represents “or”. A set of users in the role have either of the sets of attribute-value pairs.

Now suppose the role *Component 1 Software Developer* also has access to an `ethics statement`, also shown in the figure. This file has no attributes that are linked to the user attributes of Tom, Elise or Harry. As such, it does not add or detract from the candidate attribute requirements for the role. However, it can be flagged as an object that may not belong to the role since it does not appear to be semantically related.

### 3.2 Merging Candidate Attribute-Value Pairs

The second step for deriving the user attribute requirements for a role involves merging the sets of unique attribute-value pairs,  $as(i)$ , contained in  $cau^r$  if there is more than one set of attribute-value pairs. If all the users in the role had exactly the same attribute-value pairs, there would only be one unique set. However, it is very likely that different users have different attribute-value pairs. This step is found in lines 16 to 32 of Algorithm 1. In Step 1, we have a set of attribute-value pairs of every user who is a member of a role. We want to reduce the number of sets such that only members of a role who were assigned for distinct reasons are represented by different sets. This is done through merging of the candidate attribute-value pairs when users have the

same attributes but different values for these attributes.

Merger begins by comparing every pair of sets within  $cau^r$ ,  $as(i)$  and  $as(j)$ , that have exactly the same attribute names but different attribute values. If two sets have all but one attribute-value pairs where the values are equal, then one pair associated with attribute  $a_k$  where the values are not equal is considered. Let the unequal values be  $v_i$  and  $v_j$ . If  $v_i = sup(v_j)$  or  $v_j = sup(v_i)$  then the sets can be merged as one set with the equal pairs being included along with either  $v_i$  or  $v_j$  respectively. If  $com(v_i) = v_j$ , and if  $v_i$  and  $v_j$  represent all the children of  $sup(v_i)$ , then the value of  $a_k$  is replaced with  $sup(v_i)$ . If  $v_i$  and  $v_j$  do not represent all the children of  $sup(v_i)$ , then the value of  $a_k$  is replaced with a concatenation of the values  $v_i$  and  $v_j$ , ( $a_k : [v_i, v_j]$ ) which represents that either  $v_i$  or  $v_j$  are acceptable values for attribute  $a_k$ .

**EXAMPLE 6.** An example would be  $as(1)$  and  $as(3)$  where  $as(1) = (a_1 : v_3, a_3 : v_5)$  and  $as(3) = (a_1 : v_1, a_3 : v_5)$ . Both  $as(1)$  and  $as(3)$  have the same attributes ( $a_1$  and  $a_3$ ) but the values of  $a_1$  are different. The merger proceeds pairwise between sets of this type. Let us say that  $com(v_3) = v_1$ , then  $as(1)$  and  $as(3)$  can be merged with a new set that includes the multiple values for  $a_1$  that are acceptable. For example,  $as(3)$  is removed from  $cau^r$  and  $as(1)$  is replaced by  $as(1) = (a_1 : [v_1, v_3], a_3 : v_5)$ .

Continuing on, suppose  $as(5) = (a_1 : v_2, a_3 : v_5)$ . Let us further say that  $v_2 = sup(v_1) = sup(v_2)$ . The merger would result in the new  $as(1)$  being removed, leaving only  $as(5)$ .

If two sets have more than one attribute value pairs where the values are unequal, merger can only occur if all the values for one user’s attributes have the `subClass` or compatible relationship to those of the other user’s attributes. For example, if  $as(2) = \{ a_6 : v_6, a_7 : v_8 \}$  and  $as(4) = \{ a_6 : v_9, a_7 : v_7 \}$ , we have two sets that have the same attributes but both attributes have different values. If  $subClass(v_6) = v_9$  and  $subClass(v_8) = v_7$ , then we can merge the two by keeping only the more general attribute values as in  $as(2)$ . There are some users assigned to the role that have only the more general attribute values and if they are assigned correctly, then those values must be acceptable for this role. The reverse is also true. If  $subClass(v_9) = v_6$  and  $subClass(v_7) = v_8$ , then we can merge the two by keeping only the more general attribute values as in  $as(4)$ . However, if  $subClass(v_6) = v_9$  and  $subClass(v_7) = v_8$ , then we cannot merge the two because the attribute value pairs may represent two different types of users who may be assigned to the role - one with a more general value for  $a_6$  and a more specific value for  $a_7$  and another with a more specific value for  $a_6$  and a more general value for  $a_7$ .

Once the first pair of attribute sets are merged, all additional mergers occur with the merged set(s). After all mergers are made, any set that is a superset of any other set can be discarded. Since each of the sets in  $cau^r$  represent all the semantically relevant attribute-value pairs that one or more role members possess, these pairs must be sufficient to associate them with the role. Users who have a superset of these attributes have extra pairs. These extra pairs must not be essential for the role in question or there would not be role members who only have the subset. At the end of this step, we are left with a final candidate role attribute set  $cau^r$ . This set may be a set of sets where each set,  $as(i)$  within  $cau^r$  is separated with a “;” which

represents “or”. There would be more than one set of required candidate user attributes for a role if there are different qualifications that might still allow a user to perform the functions of a role. For example, suppose we have a *Project Manager* role for **Project Blue**, a software project. Some members might be part of that role because they have a college degree and project management experience while others might be part of the role because they have project management certification and have worked on software projects. The attribute and value pairs, (**hasDegree:bachelors**) and (**hasExpertiseIn:project management**) are different from (**hasCertificate:Project Management Institute**) and (**hasExpertiseIn:software**) and cannot be combined or merged.

EXAMPLE 7. For LM, the *Component 1 Software Developer* role, after merger, would have the following attribute sets,  $cau^{C1SD} = \{(\text{performsJob:software}), (\text{assignedTo:Blue}), (\text{hasExpertiseIn:[java,financial]})\}$ . The more general *software developer* role (see the partial role hierarchy in Figure 3) has the attribute set,  $cau^{SD} = \{(\text{hasDegree:[bachelors,masters]}), (\text{performsJob:software}), (\text{assignedTo:[Blue,Gold,Red]}), (\text{hasExpertiseIn:[UML,Code]})\}$ .

### 3.3 Pruning the Required Candidate User Attributes by Assessing their Significance

To assess the significance of required candidate user attributes, we look at the significance of each attribute-value set as a whole and each attribute-value pair individually in order to judge what combination of attributes would be the most likely to represent the users who belong in the role and not those who do not have the qualifications of the role. This step is given in lines 33 - 48 of Algorithm 1.

Attribute-value sets  $as(i) \in cau^r$  are *significant* if they are unique to role  $r$  and uncommon outside of role  $r$ . The significance of an attribute-value set  $as(i) \in cau^r$  can be measured by using the number of users in role  $r$  that possess all the attributes and values in  $as(i)$  versus the number of users that are not in  $r$  but still possess the attributes and values in  $as(i)$ . We compute the Attribute-Value Set Significance Factor  $\phi_c$   $as(i) \in cau^r$ , as follows:

$$\phi_c = \frac{(|\psi_{as(i)}^r|/|assigned\_users(r)|)}{(|\psi_{as(i)}^{\neg r}|/|(U-assigned\_users(r))|)},$$

where  $\psi_{as(i)}^r = \{u_j | u_j \in (assigned\_users(r)) \wedge \forall (a_i : v_i) \in as(i), a_i \text{ is an attribute of } u_j\}$ , and  $\psi_{as(i)}^{\neg r} = \{u_j | u_j \in (U - (assigned\_users(r))) \wedge \forall a_i \in as(i) | (a_i : v_i) \text{ is an attribute of } u_j\}$ .

One may set a threshold for  $\phi_c$ , and if an attribute set,  $as(i)$  is significant if the significance factor for  $as(i)$  is greater than the threshold and not significant otherwise. An insignificant attribute-value set is removed from the  $cau^r$ . Thresholds are parameters that may vary depending upon the size of the organization and of the candidate role as compared to the number in the organization.

EXAMPLE 8. For the software developer role, let us say that our final merged candidate attribute set is:  $cau^{SD} = \{(\text{performsJob:software}), (\text{assignedTo:[Blue,Gold,Red]}), (\text{hasExpertiseIn:[UML,Code]})\}$ . LM has 500 employees and 20 are currently assigned to the software developer role. All of those assigned to the role have attributes that match the candidate attribute-value pair set and 4/480 users who

are not assigned to the software developer role have the attributes and their required values. The significance factor is thus  $(20/20)/(4/480) = 120$ . If we set the threshold value of  $\phi_c$  to 100 (it is a hundred times more likely for a software developer to have the attributes than for someone other than a software developer to have those attributes), then the attribute set is relevant.

Individual attribute significance can also be assessed to trim down the attribute requirements. Like the significance of  $as(i)$ , the significance of an attribute  $a_i$  in  $cau^r$  can be measured by using the number of the users in role  $r$  that possess this attribute  $a_i : v_i$  versus the number of users that are not in  $r$  but still possess this attribute  $a_i : v_i$ .

We compute the Attribute-Value Significance Factor  $\phi_a$  of  $a_i$  in  $cau^r$  as follows:

$$\phi_a = \frac{|\psi_i^r|/|assigned\_users(r)|}{|\psi_i^{\neg r}|/|(U-(assigned\_users(r)))|},$$

where  $\psi_i^r = \{u_j | u_j \in assigned\_users(r) \wedge (a_i : v_i) \in cau^r \wedge a_i \text{ is an attribute of } u_j\}$ , and  $\psi_i^{\neg r} = \{u_j | u_j \in (U - (assigned\_users(r))) \wedge (a_i : v_i) \in cau^r \wedge a_i \text{ is an attribute of } u_j\}$ .

We may set a threshold value for  $\phi_a$  to determine  $a_i$ 's attribute-value pair significance. An attribute-value pair,  $(a_i, v_i)$  is significant if the significance factor  $\phi_{a_i}$  is greater than a pre-specified threshold, and not significant otherwise. An insignificant attribute-value pair is removed from the  $cau^r$  if it does not negatively impact the significance factors for the attribute sets in which it was included. That is, once the attribute  $a_i$  is removed, recalculate  $\phi_c$ . If  $\phi_c$  falls below the threshold value, return  $a_i$  to the attribute value set.

EXAMPLE 9. Table 1 shows the significance factors for individual attribute-value pairs within the role and outside the role. From the table, it appears that **assignedTo:[Blue,Gold,Red]** may not be that significant. However, if we go back to assessing the significance of the overall attribute-value set for the role minus this attribute, we find that the number of users who have the other two attributes in  $as(1)$  of  $cau^{SD}$  is now 7/480. Thus the new attribute set significance factor is  $1/(7/480) = 68.6$  which is less than  $\phi_c$ , making the attribute set less significant than when the **assignedTo:[Blue,Gold,Red]** attribute value pair is included.

Attribute and its Values	$\psi_i^r$	$\psi_i^{\neg r}$	$\phi_a$
<b>performsJob:software</b>	20/20	52/480	9.23
<b>hasExpertiseIn:[UML,Code]</b>	20/20	55/480	8.73
<b>assignedTo:[Blue,Gold,Red]</b>	20/20	130/480	3.69

Table 1: The significance factor

### 3.4 Checking Requirements Across Roles

The final step is a post-processing step that evaluates the usability of the required candidate user attributes,  $cau^r$ . In steps 1 to 3 discussed above, we ensure that attribute-value pairs are semantically relevant and significant for the role. This last step would best be done in a semi-automated fashion using the expertise of a security administrator to determine if the attribute requirements are appropriate.

Given a role hierarchy,  $RH$ , if two roles,  $r_i$  and  $r_j$  are such that  $r_i$  is a more specialized role than  $r_j$ , then (1)  $cau^{r_i} \neq$

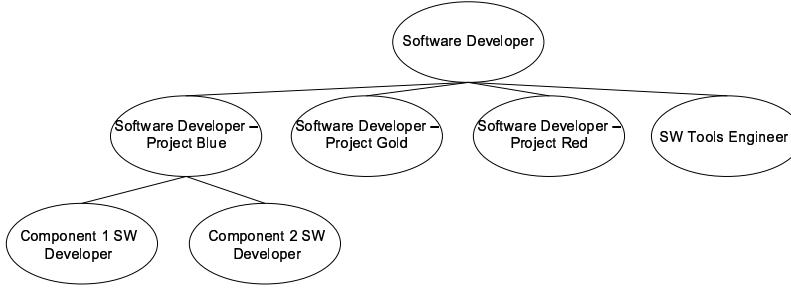


Figure 3: Partial Role Hierarchy for LM Systems

$cau^{r_j}$  and (2) the values of attributes held in common in  $cau^{r_i}$  and  $cau^{r_j}$  must have the relationship  $SubClass(v_j) = v_i$  where  $v_i$  is an attribute value of a required user attribute for  $r_i$  and  $v_j$  is an attribute value of a required user attribute for  $r_j$ .

The above is a formal way of expressing the following. A member of a more specialized role should have a more specific range of values for an attribute. For example, if an electrical engineering degree is needed to have the role *electrical engineer*, then a stricter requirement that can be imposed for the role *senior electrical engineer* is that a person has a masters degree in electrical engineering.

For those roles,  $r_k$  and  $r_l$  that are siblings or are unrelated,  $cau^{r_k} \neq cau^{r_l}$ . This means that the required user attributes for each role should be unique to that role.

If the above stated rules governing the relationships among the required user attributes are not met, the administrator may need to add more required user attributes for the relevant roles. Such addition would require defining new internal attributes and their values, and adding them to the members of the role. To do so requires determining if the users legitimately should be associated with the attribute and value, and repeating Step 3 to determine its significance with respect to that role.

EXAMPLE 10. Figure 4 shows how the concepts from this step are applied. The more specialized *C1SD* role has more stringent set of required user attributes than that of the *Software Developer - Project Blue* role (i.e., the *C1SD* Role must have expertise in a specific type of Code - java or financial). The *SW Developer - Project Blue* role, likewise, has more stringent required user attributes than the general *SW Developer* role (i.e., the *SDBLue* role requires assignment to *Project Blue*). Thus hierarchically, the attribute requirements are appropriate. However, the two sibling roles, *C1SD* and *C2SD* require reexamination. This is because, the required user attributes for *C2SD* are a subset of those for *C1SD*. Unless it is deemed appropriate that anyone who is in the *C1SD* role with java expertise can also gain access to the *C2SD* role, additional required user attributes must be added for *C2SD*. Let us say that a new attribute is defined to make the *C2SD* the required user attribute set unique. The new required attribute is `hasExpertiseIn:display` because *component 2* includes display functionality. The members of *C2SD* are then rechecked to see if it is appropriate to assign them this attribute and if so the attribute and value are added to the *UAB* for those users.

It should be noted that even though completion of this step provides a set of attribute-value pairs that may be used as the required user attributes, in a coalition environment, additional required user attributes might be added. This is because, there are some objects that should be protected from external access beyond what is done internally, to protect intellectual property for example. The result of this step is the final set of required user attributes for a role, denoted as  $au^r$ .

#### 4. EVALUATION OF ACCESS REQUESTS FROM EXTERNAL USERS

Once required attributes are determined as per Section 3, they can be used to decide whether to grant or disallow access to objects by external users who are members of organizations that are part of the coalition.

An access request by an external user can be specified as:  $\langle\langle requesting\_organization \rangle\langle requested\_object \rangle\langle requesting\_user \rangle\rangle$ . The *requesting\_organization* is a verifiable *eid* (per Section 2.1) of the organization to which the requester belongs. The *requested\_object* can either be an *oid* (if the requester knows it) or a set of object attributes, such as object type or concept,  $\{(oa_i : v_i), (oa_j : v_j), \dots\}$  that can be used to select objects from the *OAB*. Finally, the *requesting\_user* is represented by a user identifier,  $u_i$  and a set of user attributes and values,  $\{(ua_i : v_i), (ua_j : v_j), \dots\}$ . The user id is not used to make access decisions but can be used to log access requests. Access decisions are made purely on the basis of the submitted attribute-value pairs.

EXAMPLE 11. Suppose Lara Werner from HU is sending a request that she would like access to resources of type *UML* that are associated with *Project Blue*. Her request would be presented as follows:  $\langle\langle HU433 \rangle\langle (objectContains:UML), (project:Blue) \rangle\langle lwerner, (title:software\ engineer), (project:Blue), (expertise:UML) \rangle\rangle$ .

There are several steps in processing a request:

**STEP 1.** The first step in processing an object request in a coalition environment is to verify the requesting organization (i.e.,  $eid \in \chi_{47095}$ ) and determine if there are any restrictions on coalition associated with the requesting organization are noted.

**STEP 2.** If the request is made in terms of object attributes, a select query is made to the *OAB* with the specified attributes and values to determine the objects (if any)

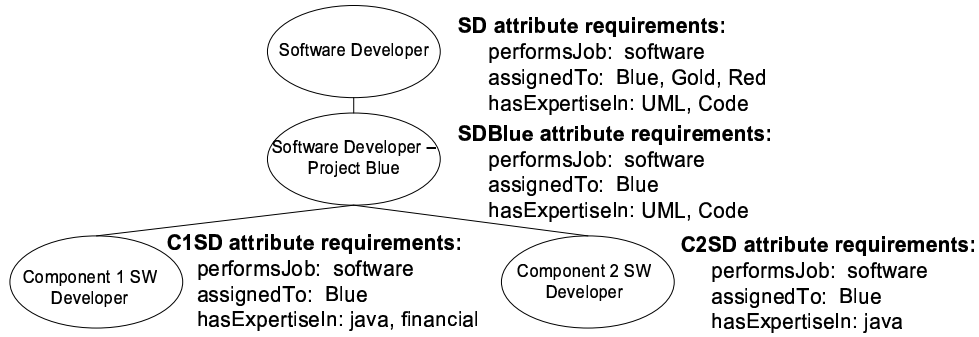


Figure 4: Role Attribute Requirement Comparisons

whose object attributes match the request. We say requested objects,  $ro$ , are the set of all objects that meet the request. An object meet the request by having equivalent attribute names to those in the request and associated attribute values which are equal, equivalent, or where the requested value has the subClass relationship with the object value. This step is skipped if a specific object is requested using  $oid$ . The roles that have permission to access the object(s) are then identified by locating the objects in  $assigned\_objects(r)$ .

EXAMPLE 12. In our example, the object is described by attributes:  $\{objectContains:UML, project:Blue\}$ . Objects with those attributes are  $o_{526}$  and  $o_{989}$ . Assuming the only operation allowed on these objects is “read”, we consult the permission to role assignment relationship,  $PA$  per Definition 1, to discover which roles have access to the object(s) that match the request. In our example, both objects  $o_{526}$  and  $o_{989}$  are assigned to the SDBlue role.

**STEP 3.** Next, the requester’s user attributes are examined and compared to the attribute requirements for the role(s),  $aur$ , that have permission to access the requested object(s).

The process for comparing the attribute requirements to the submitted attributes is as follows. The first attribute requirement is selected. If the user has not submitted an attribute with the same name, then the submitted attributes are examined to see if there is a synonymous link,  $\top(a_i, a_j)$  between the name of the required attribute  $a_i$  and the name of a submitted attribute  $a_j$ . If  $\nexists \top(a_i, a_j)$ , access is denied via that role. If there are no other roles to test, a denial message is returned to the requester with (optionally) a reason given that the request did not include sufficient credentials. If  $\exists \top(a_i, a_j)$ , then the concept hierarchy to which the values,  $v_i$  and  $v_j$  of  $a_i$  and  $a_j$  respectively are compared. The required user attribute  $a_i$  is met if  $v_i = v_j$  or if  $eq(v_i) = v_j$  or if  $subClass(v_i) = v_j$ . If the attribute requirement is not met, then access is immediately denied through that role. Otherwise, the next required user attribute is examined and the process continues until access is denied for that role or all attribute requirements for the role are successfully met. If multiple roles have access to the requested objects, then the comparisons are made until either of the required user attributes are same for any one role or required attributes are not the same for all of the roles.

EXAMPLE 13. For our example, the role SDBlue has the following attribute requirements:  $\{performsJob:software,$

$assignedTo:Blue, hasExpertiseIn:[Code,UML]\}$ . Our requester has the attributes  $\{title:software\ engineer, project:Blue, expertise:UML\}$ .

In summary, our algorithm to determine whether a request can be permitted to one or more objects is as follows: Given objects  $o_1$  to  $o_m$  that match a request (requested object list) and roles  $r_1$  to  $r_n$  (potential role list) that have access to one or more of the requested objects:

1. Compare required user attributes for  $r_i$ , a role on the potential role list, to submitted attribute-value pairs of the requester.

If submitted set of attributes is a superset of the above required user attributes for role  $r_i$ , access is given to any of the objects to which  $r_i$  has access. If no other objects are in the requested object list, respond with object ids for the objects that may be accessed. Otherwise, continue.

If the above test is not satisfied, remove  $r_i$  from the potential role list and move all objects from the requested object list to which only  $r_i$  has access to a denied objects list. If either the requested object list or the potential role list is empty, respond with the object ids for the objects that may be accessed or an access denial if there are no such objects.

2. Choose another role from the potential role list and call it  $r_i$ .

If the requester does not submit appropriate credentials covering all attribute requirements for access to the requested object, we have assumed that the access is simply denied. Alternatively, an iterative process might be introduced where a request for additional attributes can be sent, but such a process is outside the scope of this paper.

## 5. RELATED WORK

Access control research in the area of dynamic coalitions was first introduced by Philips et al. [14, 13] by providing motivating scenarios in defense and disaster recovery settings. Cohen et al. [5] proposed a model that captures the entities involved in coalition resource sharing and identifies the interrelationships among them. Two sets of researchers [4, 9] addressed the issue of automating policy negotiation and [20] addressed the issue of building trust. In our prior work [2, 18, 12], we have proposed a coalition based access

control (CBAC) model that facilitates automatic translation of coalition level policies to the implementation level policies, and vice versa. This primarily employs credential attributes in accomplishing the translation, but does not exploit semantics. Access control via attributes simplifies administration because specific users do not have to be given identities. Instead access rights are determined purely on the basis of attributes and these attributes can apply to many different users, who would present their attributes through the use of credentials. Moody and Yao in [3] proposed a similar role-based access control architecture to ours called OASIS where they mapped users via credentials issues by a third party. However, they did not address how the credential requirements are determined, which is the main focus of this paper. Wang et al. [17] presented attribute based access control (ABAC) but they also did not give a mechanism for determining which attributes should be used. Finally Li et al. [10, 11] address the problem of present a Role-Based Trust Management (RT) framework that addresses the issue of discovering credentials needed to map to a role. However, their algorithm does not address how to select attribute requirements based on existing RBAC policies.

When using the CBAC framework, one can think of two extreme cases: If a resource provider (P) requires all the credentials to be possessed by the resource requester (R) to access its resources, P's policies are ensured with the highest level of security. However, from the perspective of R it is highly restrictive (or less permissive). At the other extreme, if P does not require any credentials at all for R, then there are no guarantees on the security. However, R has the highest permissiveness to acquire the resource.

Many variations in between these extremes may exist. For example, the approach proposed in [2] computes the union of all credential attributes of the users playing a role as the required set, whereas the approach in [18] proposes a graph pruning method to reduce the required credential attribute set. The set of required attributes is reduced using frequency counts and sets of attributes held in common by users assigned to roles. The attribute significance introduced in this paper is a similar concept, but computed differently. In this paper, we further refine the required attributes using the semantic links between the associated subject and object attributes.

## 6. CONCLUSIONS

In this paper, we present an approach to facilitate automatic enforcement of access control policies among members of organizations in a coalition. Our approach exploits the semantics associated with the user and object attributes associated with a specific role and its permissions. It essentially extracts a set of required attributes by a user playing a role to access a specific object. It is an off-line process, not done at run-time so overhead issues are not critical. When an external user sends an access request, the user is given access if he/she possesses at least the required set of attributes derived through this process.

We have assumed that organizations which will be involved in coalitions will have collections of user and object attributes on which to draw. For users, attributes can be drawn from information databases that already exist on employees as well as from any credential they have received. For objects, attributes can be easily extracted particularly in

terms of keywords and concepts as well as object types. One major question and concern is the likelihood that sufficient attributes are available and certifiable that will distinguish users within a role from those not in a role. Future work plans include determining in detail what roles might exist, what objects might exist and the ontologies and credentials from which attributes might be extracted.

While our goal in this paper is to enable ad-hoc, dynamic coalition, the process of determining necessary user attributes for membership in a role could also be used to automate user assignment to roles, the process of role engineering.

## 7. REFERENCES

- [1] Owl web ontology language guide. available at <http://www.w3.org/tr/owl-guide/>.
- [2] V. Atluri and J. Warner. Automatic enforcement of access control policies among dynamic coalitions. In *International Conference on Distributed Computing and Internet Technology*, December 2004.
- [3] J. Bacon, K. Moody, and W. Yao. A model of oasis role-based access control and its support for active security. *ACM Transactions on Information and System Security*, 5(4):492–540, November 2002.
- [4] V. Bharadwaj and J. Baras. A framework for automated negotiation of access control policies. *Proceedings of DISCEX III*, 2003.
- [5] E. Cohen, W. Winsborough, R. Thomas, and D. Shands. Models for coalition-based access control (cbac). *SACMAT*, 2002.
- [6] D. Ferraiolo, R. Sandhu, S. Gavrila, D. Kuhn, and R. Chandramouli. Proposed nist standard for role-based access control. *TISSEC*, 2001.
- [7] M. P. Gallagher, A. O'Connor, and B. Kropp. The economic impact of role-based access control. *Planning report 02-1*, National Institute of Standards and Technology, March 2002.
- [8] R. Housley, W. Polk, W. Ford, and D. Solo. Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. *RFC 3280*, April 2002.
- [9] H. Khurana, S. Gavrila, R. Bobba, R. Koleva, A. Sonalker, E. Dinu, V. Gligor, and J. Baras. Integrated security services for dynamic coalitions. *Proc. of the DISCEX III*, 2003.
- [10] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust-management framework. In *IEEE Symposium on Security and Privacy*, pages 114–130, 2002.
- [11] N. Li, W. H. Winsborough, and J. C. Mitchell. Distributed credential chain discovery in trust management. *Journal of Computer Security*, 11(1):35–86, 2003.
- [12] R. Mukkamala, V. Atluri, and J. Warner. A distributed service registry for resource sharing among ad-hoc dynamic coalitions. In *Lecture Notes in Computer Science*. IFIP, December 2005.
- [13] C. Philips, E. Charles, T. Ting, and S. Demurjian. Towards information assurance in dynamic coalitions. *IEEE IAW, USMA*, February 2002.
- [14] C. Philips, T. Ting, , and S. Demurjian. Information sharing and security in dynamic coalitions. *SACMAT*, 2002.
- [15] M. Sanderson and W. B. Croft. Deriving concept hierarchies from text. In *SIGIR*, pages 206–213, 1999.
- [16] A. Schaad, J. D. Moffett, and J. Jacob. The role-based access control system of a european bank: a case study and discussion. In *SACMAT*, 2001.
- [17] L. Wang, D. Wijesekera, and S. Jajodia. A logic-based framework for attribute based access control. In *FMSE'04*, October 2004.
- [18] J. Warner, V. Atluri, and R. Mukkamala. An attribute graph based approach to map local access control policies to credential based access control policies. In *ICISS*, pages 134–147, 2005.
- [19] J. Warner, V. Atluri, and R. Mukkamala. A credential-based approach for facilitating automatic resource sharing among ad-hoc dynamic coalitions. In *IFIP*, August 2005.
- [20] T. Yu, M. Winslett, and K. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Transactions on Information and System Security*, 6(1):1–42, February 2003.