A Role-Based Administration Model for Attributes

Xin Jin
Institute for Cyber Security
Dept of Computer Science
Univ of Texas at San Antonio
San Antonio, TX, USA
xjin@cs.utsa.edu

Ram Krishnan
Institute for Cyber Security
Dept of Electrical and
Computer Engineering
Univ of Texas at San Antonio
San Antonio, TX, USA
ram.krishnan@utsa.edu

Ravi Sandhu
Institute for Cyber Security
Dept of Computer Science
Univ of Texas at San Antonio
San Antonio, TX, USA
ravi.sandhu@utsa.edu

ABSTRACT

Attribute based access control (ABAC) provides flexibility and scalability for securely managing access to resources, particularly in distributed environments. In ABAC, access requests are authorized through policies evaluated with respect to attributes of various entities such as users, subjects, objects, context, etc. Administration of user attributes is one of the major issues in ABAC. However, there has been little research in this area. This paper proposes a framework to administer user attributes using role based access control (RBAC). Our motivation is that RBAC has demonstrated advantages in ease of administration and is widely deployed in the industry. Thus, an appealing possibility is to use RBAC to manage user attributes. In this paper we propose a generalized version of the user role assignment model in the ARBAC97 administrative role based access control model. The generalized version treats role as just one possible attribute of the user. The paper explores the model's advantages and limitations and provides guidance for future development of more comprehensive user attribute administrative models.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—Access controls; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Security

Keywords

Attributes, Administration, Access Control

1. INTRODUCTION

Attribute based access control (ABAC) has been proposed to provide flexible and scalable access control in distributed systems and overcome the shortcomings of classical access control models such as discretionary access control (DAC) [10], mandatory access control (MAC) [7] and role based access control (RBAC) [9]. Access requests in such models are generally evaluated against policies which are composed of attributes of involved entities such as requesters, resources and so on. User attributes refer to properties of users in the access control context. Examples are user id, security clearance, etc. User attributes, thus, are crucial factors which restrict the permissions on resources available to the users. Considerable papers have been published towards bringing a consensus on ABAC models and systems [1] [2] [5] [11]. However, much effort has been devoted to issues regarding request evaluation and policy specification. This effort is based on the assumption that all users are associated with sets of user attributes with assigned values. Issues such as who is authorized to assign user attributes and what is the range of values he or she is allowed to assign remain unclear. The assignment and modification of user attributes, which are important aspect of ABAC, has not received sufficient attention. Currently there is no widely accepted set of informal requirements, let alone formal models, for supporting administration of user attributes. An initial informal statement of user attribute administration appears in role based trust management [4] where owners of their roles are the only entities who can manager the roles.

Our central contribution in this paper is to study administrative issues of user attribute management in ABAC. For this purpose, we use the well-known administrative role based access control model (ARBAC97) [6] [8] which to our knowledge has not been previously applied in this domain. Our motivation for choosing ARBAC97 includes its ease of administration and sizable literature. Our principle finding is that ARBAC97, with proper generalization, is suitable in large measure to address user attribute assignment administration. In particular, we generalize the user role assignment model (URA) which is part of ARBAC97 since role is just one type of user attribute, this generalization is straight forward yet efficient. We also discuss the limitations of using ARBAC97 revealed by this work and provide insights on future work in the development of a more comprehensive administrative model for ABAC.

The remainder of this paper is organized as follows. Section 2 reviews the previously published administrative role based access control model and related work. Section 3 develops formal URA-based administrative models for user attribute assignment. Section 4 discusses the limitations of the proposed approach and section 5 concludes the paper.

SRAS '12, September 19, Minneapolis, MN, USA

2. BACKGROUND

In role based access control model (RBAC) [9], permissions are associated with roles and users are made members of roles, thereby acquiring the role's permissions. RBAC simplifies administration of authorization. Administrative role based access control (ARBAC97) [6] is designed for user-role assignment, role-permission assignment and role hierarchy specification in RBAC. In this paper, we deal with user attribute assignment, and hence discuss the use of user-role assignment (URA) which is part of ARBAC97. The URA97 model is defined in two steps: granting a user membership in a role and revoking a user's membership. The goal of URA97 is to impose restrictions on which users can be added to a role by whom, as well as to clearly separate the ability to add and remove users from other operations on the role. The notion of prerequisite condition is a key part of URA97. All examples included in the following are according to figure 1 adapted from [6].

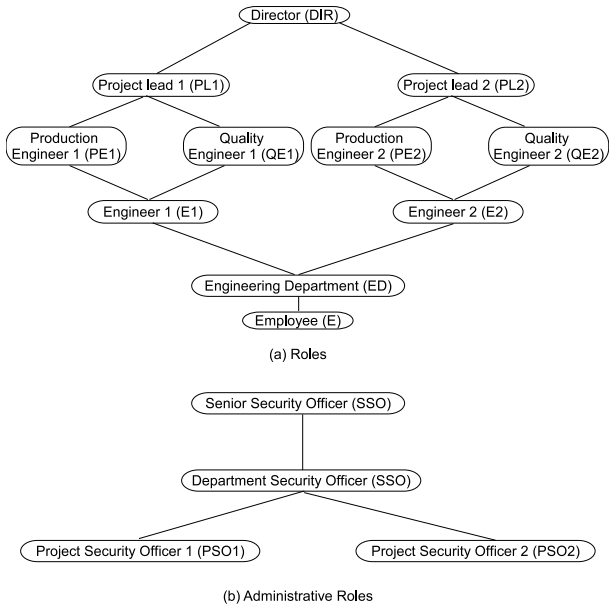

Figure 1: *Example Role and Administrative Role Hierarchies*

2.1 The URA97 Grant Model

User-role assignment is authorized by means of the following relation .

$$can\_assign \subseteq AR \times CR \times 2^R \quad (1)$$

The meaning of $can\_assign(ar, x, \{a, b, c\})$ is that a member of the administrative role x (or a member of an administrative role that is senior to x) can assign a user whose current membership, or nonmembership, in regular roles satisfies the prerequisite condition y to be a member of regular roles a, b, or c. In relation (1), AR stands for specific administrative roles such as Project Security Officer 1 (PSO1). CR is a prerequisite condition which is a boolean expression using the usual $\vee$ and $\wedge$ operators on terms of the form x and $\bar{x}$ where x is a regular role. A prerequisite condition is evaluated for a user u by interpreting x to be true if $(\exists x' \geq x)(u, x') \in$ URA and $\overline{x}$ to be true if $(\forall x' \geq x)(u, x') \notin$ URA, where URA is the user-role assignment relation of RBAC. For a given set of roles R, we let CR denote all possible prerequisite conditions that can be formed using the roles in R. 2^R represents the roles which can be assigned to the users who satisfy the prerequisite condition. In this paper, the role sets are specified using the range notation given in table 1.

Table 1: *Role Range Notation*

[x,y]={r∈R | r ≥ x ∧ y ≥ r}	[x,y)={r∈R | r ≥ x ∧ y > r}
(x,y]={r∈R | r > x ∧ y ≥ r}	(x,y)={r∈R | r > x ∧ y > r}

Table 2: *can_assign with Prerequisite Roles*

Admin. Role	Prereq. Condition	Role Range
PSO1	ED	[PL1, E1]
PSO1	ED ∧ $\overline{\text{QE1}}$	[PE1, PE1]
PSO1	ED ∧ $\overline{\text{PE1}}$	[QE1, QE1]

Examples are shown in table 2. The first tuple authorizes PSO1 role (and its seniors) to assign users with prerequisite role ED into roles {E1, QE1, PE1, PL1}. The second tuple authorizes PSO1 role to assign users with prerequisite condition ED $\wedge$ $\overline{\text{QE1}}$ to PE1. The third tuple authorizes PSO1 to assign users with prerequisite condition ED $\wedge$ $\overline{\text{PE1}}$ to QE1. The second and third together authorize PSO1 to put a user who is a member of ED into one but not both of PE1 and QE1.

2.2 The URA97 Revoke Model

In URA, the role assignment and revoke permissions are authorized separately. The URA97 model controls user-role revocation by means of the following relation.

$$can\_revoke \subseteq AR \times 2^R \quad (2)$$

In relation (2), the meaning of $can\_revoke(x, Y)$ is that a member of the administrative role x (or a member of an administrative role senior to x) can revoke membership of a user from any regular role $y \in Y$. Y is also specified using range notation. Examples are shown in table 3. The first tuple authorizes PSO1 to revoke membership from the roles {E1, PE1, QE1} (represented by the role range notation). Suppose Alice is member of PSO1 and Bob is member of PE1. Then Alice is authorized to remove Bob's membership of PE1. The second tuple authorizes PSO2 to remove membership from the roles {E2, PE2, QE2}.

2.3 User Attribute Definition

We follow the formal definition of user attributes in [2] as follows.

- UA is a finite set of user attribute functions.
- $\forall$ ua $\in$ UA, Range(ua) specifies the range of each attribute as a finite set of atomic values.
- attType : UA $\rightarrow$ {set, atomic}.
- $\forall$ ua $\in$ UA, ua : U $\rightarrow$ Range(ua) if attType(ua) = atomic, and ua : U $\rightarrow 2^{\text{Range(ua)}}$ if attType(ua) = set.

Each user is associated with a finite set of user attribute functions. Attribute ranges are represented as a finite set of atomic values. For example, range(role) = {manager,

Table 3: *Examples of can_revoke*

Aadmin. Role	Role Range
PSO1	[E1, PL1)
PSO2	[E2, PL2)

employee, prjleader}, where role is a user attribute. Each attribute is either set-valued or atomic-valued with respect to its range. Atomic-valued means that users are assigned exactly one value from the attribute's range. Set-valued attributes are assigned a subset of the attribute's range. For example, users are allowed to be assigned more than one role in a company such as role(Alice) = {manager, prjleader}. However, users are allowed to be assigned only one value for salary, e.g. salary(Alice) = 2000.

2.4 Other Related Work

Role based trust management (RT) [4] is a family of role-based trust management languages. An RT role enables its members to access specific resources assigned to that role. Each role is owned by an entity and the assignment of users to that role is under complete control of the entity. For example, ACM.Member role is administrated only by the entity ACM. Automated trust negotiation [12] reveals user information to the server through processes of negotiation when server and user gradually expose attributes as mutual trusts are established. This framework is based on the assumption that users are already associated with attributes and values are pre-assigned.

3. GENERALIZED URA MODEL (GURA)

We first give an overview of the proposed generalized URA (GURA) model and then present the formal model illustrated through examples. In addition, we discuss the advantages and limitations of the new model.

3.1 Model Overview

The main purpose of the model is to offer a means to specify the permissions for each administrative role in managing user attributes. The permissions include:

- Assign values to a specific user attribute of a specific set of users. The assignment of set-valued attributes only adds the value to the original user attributes while the assignment to atomic-valued user attributes automatically removes the old value and assigns the new value. For example, a user with seniormanager role is allowed to set the role of each employee to values such as {groupmanager, projectmanager, prj1leader}. However each employee can only be assigned one position. Thus, if role seniormanager promotes an employee from role prj1leader to role groupmanager, he must also have the permission to remove role prj1leader from that employee;

- Delete values from a specific set-valued user attribute of a specific set of users. For example, role projectmanager is authorized to add employees to any project among {proj1, proj2, proj3}. However, role projectmanager is not authorized to remove employees from those projects, only corresponding project leaders are authorized to do so.

This model is designed by generalizing role as an attribute in the URA97 model in ARBAC97. For this purpose, we first need to distinguish the difference between role and a general attribute (for simplicity, the term attributes referred to in this paper represents user attribute). As introduced in [2], attributes are functions which take a user as input and return a specific value. We summarize the difference as follows.

- Unlike roles, attributes are not assigned with permissions directly while roles are associated with permissions. For example, the name and address of users only represents certain properties and are not associated with permissions unless certain authorization policies depend on those two attributes.

- Unlike role hierarchy, an attribute's range is not required to be ordered. For example, a location attribute of each employee in a company does not reflect a hierarchy. Note that partial ordering of a user attribute value may not result in permission inheritance. In RBAC, if manager is a senior role of an employee role, then manager gains all permissions from employee and this relationship can be represented using manager $>$ employee. While in user salary attribute, there may be a relationship such as $2000 > 1500$. However, this relationship does not imply the inheritance of permissions. That is, a user with salary 2000 need not get higher privileges than the one with salary 1500.

Similar to URA97, there are relationships between the administration range of administrative roles and the semantic meanings of the roles. Its obvious that these relationships play as important constraints on the user permissions. In RBAC, human resource manager is authorized to assign a regular user an employee role and similarly in ABAC, only the administrator in computer science department is authorized to assign department attribute for each user and the value can only be assigned as ComputerScience. Thus, the model offers mechanisms to specify fine-grained permissions for each administrative role. The decision regarding which user attribute to be assigned to which administrative role depends on specific system design and should follow certain principles:

- The attribute should be related to the semantic of administrative role. For example, manager of project1 can only add or remove prj1 from employee's attribute of involvedproj and any attempt to remove prj2 from the user attribute by this role is not allowed.

- Since administrative roles may be hierarchical, senior administrative roles get the permissions on junior administrative roles. Thus, the permissions assignment should not cause duplications or conflicts.

In summary, the new model will be able to specify the following elements in permissions to be assigned to administrative roles: (1) the prerequisite conditions specified using user attributes to identify the set of users the permissions apply to; (2) the user attribute function name whose value can be modified in this permission; (3) the allowed value to be assigned.

3.2 Formal Model

3.2.1 Generalized URA Model 0 (GURA$_0$)

In this administrative system for user attributes, the generalized URA model (GURA$_0$) is composed of three relations: adding set-valued user attribute, deleting set-valued user attribute and assigning atomic-valued user attribute. Each of them can be specified by system architects through definitions 1-3 as follows.

Definition 1. Adding set-valued user attribute is controlled by means of the relation:

$$\forall sua \in \text{SUA}.can\_add_{sua} \subseteq AR \times EXPR(sua) \times 2^{\text{Range(sua)}} \quad (3)$$

The meaning of *can_add*(*ar*, *Exp*(*sua*), *AllowedRange*) is that users who are members of the role *ar* or senior roles of *ar* are authorized to add any element in *AllowedRange* to the *sua* attribute of users whose *sua* satisfies conditions specified in *Exp*(*sua*). AR represents the set of existing administrative roles. SUA is a set representing all set-valued user attributes. EXPR(sua) represents all possible logic expressions composed of the specific user attribute function *sua* and constant symbols in context. Each expression should return true or false. For example, Exp(involvedprj) could be prj1 $\in$ involvedprj(u) $\wedge$ prj2 $\notin$ involvedprj(u). The syntax for specifying these expressions is defined in this paper. We first define a common expression language (CEL) as follows:

φ ::= $\varphi \wedge \varphi$ | $\varphi \vee \varphi$ | (φ) | $\neg \varphi$ | $\exists$ x $\in$ set.φ |
$\forall$ x $\in$ set.φ | set setcompare set | atomic $\in$ set |
atomic $\notin$ set | atomic atomiccompare atomic
atomiccompare ::= $<$ | $=$ | $\leq$ | $\neq$
setcompare ::= $\subset$ | $\subseteq$ | $\nsubseteq$

This CEL is further specified for each of the languages we need to define below and set and atomic are terminals that need to be specified for each specific instance of CEL. In this relation, the EXPR(sua) is specified using an instance of CEL where set and atomic are defined as follows:

set ::= sua(u) | constantSet
atomic ::= constantAtomic

sua is the specific user attribute in the *can_assign* relation. NULL is included in all EXPR(sua) (also for the following definitions). If NULL is specified as Exp(sua) in a specific can_assign command, it means that the *can_assign* relation is applicable to all existing users. AllowedRange specifies the values which can be added to the current value of user attributes and it should be in accordance with the range of the user attribute in the context. For example, AllowedRange = {prj1, prj2}. To express the second line in table 2, the relation for attribute role is: can_add(PSO1, ED $\in$ role(u) $\wedge$ QE1 $\notin$ role(u), {PE1}), where role is one of the set-valued user attributes and the permission is assigned to role PSO1. More examples are given in table 4.

Definition 2. Deleting set-valued user attribute is controlled by means of the relation:

$$\forall sua \in \text{SUA}.can\_delete_{sua} \subseteq AR \times EXPR(sua) \times 2^{\text{Range(sua)}} \quad (4)$$

This is similar to definition 1. The meaning of *can_delete*(*ar*, *Exp*(*sua*), *AllowedRange*) is that users who are members of *ar* or senior roles of *ar* are authorized to delete values in *AllowedRange* from the attribute *sua* of users whose *sua* satisfies conditions specified in *Exp*(*sua*). To specify the second line in table 3, the relation for attribute role is: can_delete(PSO2, role(u) $\subseteq 2^{\{\text{E2,PE2,QE2}\}}$, {E2, PE2, QE2}). Note that if the administrative role attempts to delete values which do not belong to the user, this operation has no effect.

We define the following relation for atomic-valued user attribute assignment and deletion.

Definition 3. Assigning atomic-valued user attribute is controlled by means of the relation:

$$\forall aua \in \text{AUA}.can\_assign_{aua} \subseteq AR \times EXPR(aua) \times 2^{\text{Range(aua)}} \quad (5)$$

The meaning of *can_assign*(*ar*, *Exp*(*aua*), *AllowedRange*) is that users who are members of *ar* or senior roles of *ar* are authorized to assign the *aua* attribute of users which satisfies conditions specified in *Exp*(*aua*). AUA is the set of all atomic user attributes. EXPR(aua) denotes all possible expressions composed of aua using the instance of CEL where set and atomic are formally defined as follows:

set ::= constantSet
atomic ::= aua(u) | constantAtomic

aua is the specific user attribute in *can_assign* relations. Note that this relation will automatically remove the current value. Since we are dealing with atomic-valued attributes here, there is no notion of can_remove. For example, seniorManager is a role who is authorized to set the salary of users whose salary is higher than 2000 to be 6000, 7000 or 8000. It is specified as can_assign(seniorManager, salary(u) $>$ 2000, {6000, 7000, 8000}) for attribute salary. This relation will give the permission to seniorManager to assign a value for the salary attribute and thereby the permission to update the old value. If the seniorManager needs to set the salary of a user to NULL, this can be achieved by setting NULL as one of the allowed values.

Let's further understand the above definitions through a detailed example. Consider this scenario: each employee in company A is associated with attributes involvedprj, salary and group. Attribute involvedprj is a set-valued attribute returning the list of projects the user participates in. Attribute salary is an atomic-valued attribute returning the salary of the employee. Attribute group is a set-valued attribute returning the groups the user joins. Role prjmanager is a senior role of prj1leader and prj2leader. Role prj1leader is authorized to add employees to project1 only if the employee is not involved in project2 and add users to any groups. Role prj2leader is authorized to add employees to project2 only if the employee is not involved in project1 and add users to any groups. Both roles can remove employees from their projects. Role prjmanager can add any employee to prj1 or prj2 but not both. Role prjmanager can assign salary of any employee to {3000, 4000, 6000, 8000}. To satisfy the requirements, the permissions assigned to each administrative role are specified in table 4. Note that role prjmanager will inherit permissions from role prj1leader and role prj2leader.

Table 4: *Examples*

can_add relation for attribute involvedprj:
(prj1leader, prj2 $\notin$ involvedprj(u), {prj1})
(prj2leader, prj1 $\notin$ involvedprj(u), {prj2})

can_add relation for attribute group:
(prj1leader, NULL, {group1, group2, group3})
(prj2leader, NULL, {group1, group2, group3})

can_delete relation for attribute involvedprj:
(prj1leader, prj1 $\in$ involvedprj(u), {prj1})
(prj2leader, prj2 $\in$ involvedprj(u), {prj2})

can_delete relation for attribute group:
(prj1leader, NULL, {group1, group2, group3})
(prj2leader, NULL, {group1, group2, group3})

can_assign relation for attribute salary:
(prjmanager, NULL, {3000, 4000, 6000, 8000})

3.2.2 Generalized URA Model 1 (GURA$_1$)

There are limitations to the above definitions. We notice that the precondition is purely based on the user attribute which is to be modified. In the above example, prjleader may not be allowed to add employees who did not pass the technical training in the company. The precondition here is beyond the expressive power of GURA$_0$. To further generalize the relation, we can extend the precondition to be composed of all user attribute functions. Thus we define GURA$_1$ model which is composed of definitions 4-6.

Definition 4. Adding set-valued user attribute is controlled by means of relation:

$$\forall sua \in \text{SUA}.can\_add_{sua} \subseteq AR \times EXPR(UA) \times 2^{\text{Range(sua)}} \quad (6)$$

In the above set, EXPR(UA) represents all possible boolean expressions composed of all user attributes. One example of Exp(UA) is prj1 $\notin$ involvedprj(u) $\wedge$ trainingpassed(u) = true. The language used for specifying the expression is an instance of CEL where set and atomic is defined as follows:

set ::= allsetua(u) | constantSet
atomic ::= allatomicua(u) | constantAtomic
allsetua ::= {setua | setua $\in$ UA $\wedge$ attType(setua) = set}
allatomicua ::= {atomicua | atomicua $\in$ UA $\wedge$ attType(atomicua) = set}

In this language, allsetua and allatomicua means all set-valued user attribute and atomic-valued user attributes. Thus, the expression can be more fine-grained and smaller sets of eligible users can be specified for each permission. With the new definition, the range of each administrative role can be further restricted. Similarly, we give the definition of the remaining two sets for completeness.

Definition 5. Deleting Set-valued user attribute is controlled by means of the relation:

$$\forall sua \in \text{SUA}.can\_delete_{sua} \subseteq AR \times EXPR(UA) \times 2^{\text{Range(sua)}} \quad (7)$$

Table 5: *Examples*

can_add relation for attribute involvedprj:
(prj1leader, prj2 $\notin$ involvedprj(u) $\wedge$ trainingpassed(u) = true $\wedge$ clearance(u) > S $\wedge$ C $\in$ skills(u), {prj1})

(prj2leader, prj1 $\notin$ involvedprj(u) $\wedge$ trainingpassed(u) = true $\wedge$ clearance(u) > S $\wedge$ C $\in$ skills(u), {prj2})

can_add relation for attribute skill:
(secretary, NULL, {C ,C++, Java})

can_delete relation for attribute involvedprj:
(prj1leader, prj1 $\in$ involvedprj(u), {prj1})
(prj2leader, prj2 $\in$ involvedprj(u), {prj2})

can_delete relation for attribute skill:
(secretary, NULL, {C, C++, Java})

can_assign relation for attribute trainingpassed:
(trainingmanager, NULL, {true, false})

can_assign relation for attribute clearance:
(humanmanager, NULL, {TS, S, C, U})

Definition 6. Assigning atomic-valued user attribute is controlled by means of the relation:

$$\forall aua \in \text{AUA}.can\_assign_{aua} \subseteq AR \times EXPR(UA) \times 2^{\text{Range(aua)}} \quad (8)$$

EXPR(UA) used in definitions 5-6 are specified using the same language as definition 4. To illustrate the purpose of the above extension, we give the following example. Consider this scenario: UA = {involvedprj, trainingpassed, clearance, skills}. Attribute involvedprj returns the involved projects for each employee, trainingpassed represents whether the employee has passed the training. Attribute clearance represents the security level assigned to each user and attribute skills returns the programming skills of each employee. Attribute trainingpassed and clearance are atomic-valued attributes and others are set-valued. Role prj1leader is a role that can assign employees to role project1 if (1) the employee is not assigned to project2 and; (2) the employee has passed the training and; (3) the employee is assigned a higher security clearance than secret (S) and (4) the employee must be familiar with C programming. Role prj2leader is a role that can assign employee to role project2 and the conditions are the same as above except the employee should not be involved in project1. Other user attributes are managed by corresponding roles. Role trainingmanager is authorized to set trainingpassed attribute of each employee to be true or false. Role humanmanager is a role in human resource and is authorized to assign the clearance of each employee to be one of the following: top secret (TS), Secret (S), Classified (C) and Unclassified (UC). Each level represents different permissions for accessing sensitive documents in the company. Role secretary is a role that can enter employees' information into the system and thus be able to set the skills attribute of each employee by scanning their resumes. The solution is specified in table 5.

Suppose we have the following users with corresponding attributes in table 6. Combined with the policy specified in table 5, we can see that both prj1leader and prj1leader can only assign {Alice, Charlie, Dan} to their projects. However, according to the policy in table 4, both roles are authorized

Table 6: *Example User Attributes*

User	involved -prj	training -passed	clearance	skills
Alice	∅	true	TS	{C++, Java}
Bob	{prj3}	false	TS	{C++, C, Java}
Charlie	{prj3}	true	TS	{ C }
Dan	∅	true	TS	{C++, Java}
Eve	{prj1}	true	UC	{C++, Java}
Fred	{prj2}	true	UC	{C, Java}

to assign all users to their projects. We can see that $GURA_0$ provides mechanism to assign permissions to administrative roles and provide least privileges considering only single attribute while $GURA_0$ provides more expressive power.

4. DISCUSSION

We have incrementally developed a series of URA-based formal models for user attribute administration. The main approach is to define the administrative range of each administrative role and then assign those roles to users. In this work we have discovered some advantages and limitations of the role based approach as described in the subsection below.

Since we leveraged RBAC for user attribute administration, its advantages are inherited. Once the administrative permissions for each administrative role is specified, it is convenient to assign and de-assign users from administrative role without modification of role-permission assignment. The precondition can be used to accurately specify the administration range of each role. In these models, least privilege can be achieved. Each role is assigned the least permissions required. There are a number of limitations. It is cumbersome to express distributed policies such as: each user who works for more than two years can update level attribute of employees who have not passed qualification test to be qualified. In order to specify the policy, an administrative role need to be specified for each regular user and they themselves need to be assigned to the corresponding administrative role. The complexity of this operation is O(n) (suppose there are n existing users). In order to achieve fine-grained administration, large number of roles may need to be specified. For example, an administrative role projectleader need to be defined for different projects because different project leaders can assign the attribute of different sets of users. While the number of projects in large company could be hundreds of thousand.

5. CONCLUSION AND FUTURE WORK

This paper presents a GURA framework which applies RBAC to manage user-attribute assignment in attribute based access control. The main approach is to follow the URA97 model and generalize role as a general user attribute while distinguishing them from each other. Through this work, we are able to show that the generalized framework is capable of specifying a wide variety of policies and the advantages of role based access control are inherited. In future work, we plan to propose more advanced models. For instance, integrating attributes into role as administrative model to improve scalability [3]. Another interesting extension is that administration permissions are allowed to be delegated in controlled manner for distributed administration.

Acknowledgment

The authors are partially supported by grants from AFOSR MURI and the State of Texas Emerging Technology Fund.

6. REFERENCES

[1] Sushil Jajodia, P. Samarati, Maria Luisa Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Trans. Database Syst.*, 2001.

[2] Xin Jin, Ram Krishnan, and Ravi Sandhu. A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC. In *DBSec*, 2012.

[3] Xin Jin, Ravi Sandhu, and Ram Krishnan. RABAC: Role-Centric Attribute-Based Access Control. In *MMM-ACNS*, 2012.

[4] Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust management framework. In *2002 IEEE S&P*.

[5] Jaehong Park and Ravi Sandhu. The UCONabc usage control model. *ACM Trans. Inf. Syst. Secur.*, 2004.

[6] Ravi Sandhu, Venkata Bhamidipati, and Qamar Munawer. The arbac97 model for role-based administration of roles. *ACM Trans. Inf. Syst. Secur.*, 1999.

[7] Ravi S. Sandhu. Lattice-based access control models. *IEEE Computer*, 1993.

[8] Ravi S. Sandhu and Venkata Bhamidipati. The ura97 model for role-based user-role assignment. In *DBSec*, pages 262–275, 1997.

[9] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[10] Ravi S. Sandhu and P. Samarati. Access control: Principles and practice. *IEEE Com. Mag.*, 1994.

[11] Lingyu Wang, Duminda Wijesekera, and Sushil Jajodia. A logic-based framework for attribute based access control. In *In 2nd ACM Workshop on FMSE*, 2004.

[12] Ting Yu, Marianne Winslett, and Kent E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Trans. Inf. Syst. Secur.*, 2003.