The ARBAC97 Model for Role-Based Administration of Roles

RAVI SANDHU, VENKATA BHAMIDIPATI, and QAMAR MUNAWER
George Mason University

In role-based access control (RBAC), permissions are associated with roles and users are made members of roles, thereby acquiring the roles' permissions. RBAC's motivation is to simplify administration of authorizations. An appealing possibility is to use RBAC itself to manage RBAC, to further provide administrative convenience and scalability, especially in decentralizing administrative authority, responsibility, and chores. This paper describes the motivation, intuition, and formal definition of a new role-based model for RBAC administration. This model is called ARBAC97 (administrative RBAC '97) and has three components: URA97 (user-role assignment '97), PRA97 (permission-role assignment '97), and RRA97 (role-role assignment '97) dealing with different aspects of RBAC administration. URA97, PRA97, and an outline of RRA97 were defined in 1997, hence the designation given to the entire model. RRA97 was completed in 1998. ARBAC97 is described completely in this paper for the first time. We also discusses possible extensions of ARBAC97.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems; D.4.6 [**Operating Systems**]: Security and Protection—*Access controls*; D.4.7 [**Operating Systems**]: Organization and Design—*Distributed systems*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms*; H.2.0 [**Database Management**]: General—*Security, integrity, and protection*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Algorithms, Management, Security

Additional Key Words and Phrases: Authorization management, role based access control, security administration

1. INTRODUCTION

Role-based access control (RBAC) has recently received considerable attention as a promising alternative to traditional discretionary and mandatory access controls (see the companion papers in this issue: [Nyanchama and Osborn 1999; Bertino et al. 1999; Ferraiolo et al. 1999]). In RBAC, permis-

This research was supported in part by the National Science Foundation, the National Institute of Standards and Technology, and the National Security Agency at the Laboratory for Information Security Technology at George Mason University.
Authors' address: Information and Software Engineering, George Mason University, Mail Stop 4A4, Fairfax, VA 22030; email: sandhu@gmu.edu; http://www.list.gmu.edu.

sions are associated with roles and users are made members of appropriate roles, thereby acquiring the roles' permissions. This greatly simplifies management of permissions. Roles are created for the various job functions in an organization, and users are assigned roles based on their responsibilities and qualifications. Users can be easily reassigned from one role to another. Roles can be granted new permissions as new applications and systems are incorporated, and permissions can be revoked from roles as needed. Role-role relationships can be established to lay out broad policy objectives.

RBAC is policy-neutral and flexible. The policy that is enforced is a consequence of the detailed configuration of various RBAC components. RBAC's flexibility allows a wide range of policies to be implemented. As an illustration of its flexibility, RBAC can be configured to enforce classical mandatory access controls [Nyanchama and Osborn 1996; Sandhu 1996] and discretionary access controls [Sandhu and Munawer 1998].

RBAC supports three well-known security principles: *least privilege*, *separation of duties*, and *data abstraction*. Least privilege is supported because RBAC can be configured so only those permissions required for tasks conducted by members of the role are assigned to the role. Separation of duties is achieved by ensuring that mutually exclusive roles must be invoked to complete a sensitive task, such as requiring an accounting clerk and account manager to participate in issuing a check. Data abstraction is supported by means of abstract permissions such as credit and debit for an account object, rather than the read, write, execute permissions typically provided by the operating system. However, these principles must be embodied in a detailed configuration of RBAC components.

Administration of RBAC is very important, and must be carefully controlled to ensure that policy does not drift away from its original objectives. In large enterprise-wide systems, the number of roles can be in the hundreds or thousands, and users in the tens or hundreds of thousands. Managing these roles and users, and their interrelationships, is a formidable task that cannot realistically be centralized in a small team of security administrators. Decentralizing the details of RBAC administration without loosing central control over broad policy is a challenging goal for system designers and architects. There is tension between the desire for scalability through decentralization and maintenance of tight control. A complete solution to this problem requires further research and faces significant theoretical problems (see, for example, Harrison et al. [1976]; Sandhu [1992]; and Sandhu and Ganta [1995]). Our work provides a significant and practical advance towards this goal.

Since the main advantage of RBAC is to facilitate administration, it is natural to ask how RBAC itself can be used to manage RBAC. We believe using RBAC for managing RBAC is an important factor in its long-term success. The central contribution of this paper is a comprehensive model for role-based administration of RBAC.

There are many components in RBAC, making RBAC administration multifaceted. In particular, we can separate the issues of assigning users to

roles, assigning permissions to roles, and assigning roles to roles to define a role hierarchy. These activities are all required to bring users and permissions together. However, in many cases, they are best done by different administrators or administrative roles. Assigning permissions to roles is typically the province of application administrators. A banking application can be implemented so that credit and debit operations are assigned to a teller role, whereas approval of a loan is assigned to a managerial role. Assignment of actual individuals to the teller and managerial roles is a personnel management function. Assigning roles to roles has aspects of user-role assignment and role-permission assignment. More generally, role-role relationships establish broad policy.

Our administrative model is called ARBAC97 (administrative RBAC '97). It has three components: URA97 is concerned with user-role assignment; PRA97 (permission-role assignment '97) with permission-role assignment, and is a dual of URA97;[1]and RRA97 (role-role assignment '97) deals with role-role assignment. RRA97 itself has several components determined by the kind of roles that are involved, as discussed in Section 5.

The rest of the paper is organized as follows. Section 2 reviews the RBAC96 model, and ARBAC97 is developed in its context. Sections 3, 4, and 5 describe, respectively, URA97, PRA97, and RRA97. Section 6 discusses various aspects of ARBAC97, including implementations and extensions. Section 7 concludes the paper.

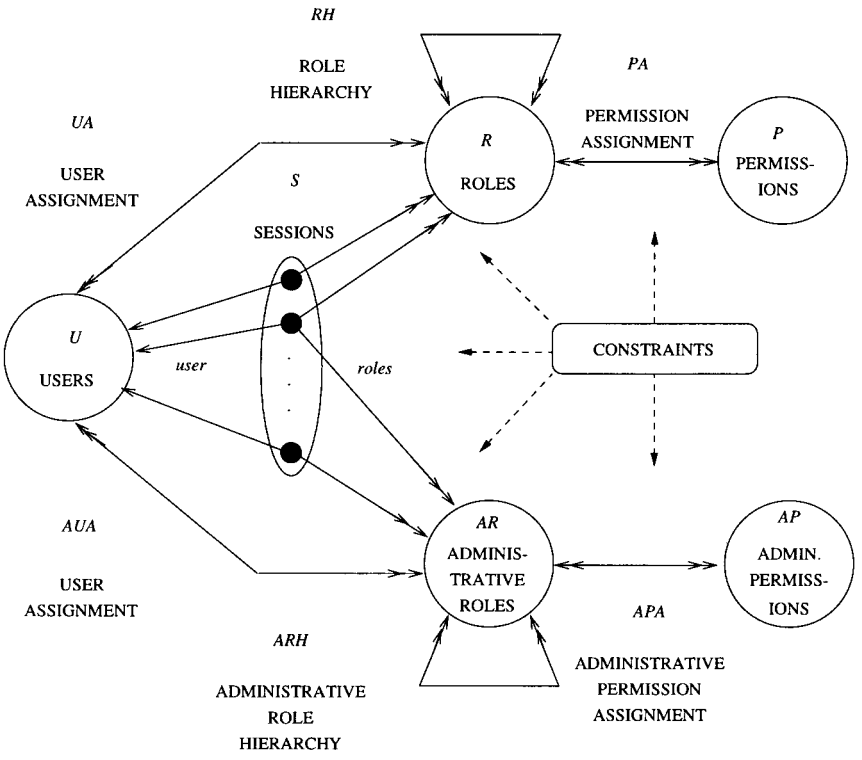
2. THE RBAC96 MODEL

A general family of RBAC models called RBAC96[2] was defined by Sandhu et al. [1996]. Figure 1 illustrates the most general model in this family. For simplicity, we use the term RBAC96 to refer to the family of models as well as to its most general member.

The top half of Figure 1 shows (regular) roles and permissions that regulate access to data and resources; the bottom half shows administrative roles and permissions. Intuitively, a user is a human being or an autonomous agent, a role is a job function or job title within the organization with some associated semantics regarding the authority and responsibility conferred on a member of the role, and a permission is an approval of a particular mode of access to one or more objects in the system or some privilege to carry out specified actions. Roles are organized in a partial order $\geq$, so that if $x \geq y$ then role x inherits the permissions of role y. Members of x are also implicitly members of y. In such cases, we say x is senior to y. Each session relates one user to possibly many roles. The idea is that a user establishes a session and activates some subset of roles that he or she is a member of (directly or indirectly by means of role hierarchy).

[1]We have often observed a duality between user-role and permission-role relationships. For example, every constraint on user-role relationships has a dual counterpart with respect to permission-role relationships, and vice versa [Sandhu et al. 1996]. Thus it is natural that URA97 and PRA97 are duals.

[2]The name RBAC96 was first used in Sandhu [1997a].



—U, a set of users; R and AR, disjoint sets of (regular) roles and administrative roles; P and AP, disjoint sets of (regular) permissions and administrative permissions; S, a set of sessions
—$UA \subseteq U \times R$, user to role assignment relation
$AUA \subseteq U \times AR$, user to administrative role assignment relation
—$PA \subseteq P \times R$, permission to role assignment relation
$APA \subseteq AP \times AR$, permission to administrative role assignment relation
—$RH \subseteq R \times R$, partially ordered role hierarchy
$ARH \subseteq AR \times AR$, partially ordered administrative role hierarchy
(both hierarchies are written as $\geq$ in infix notation)
—$user : S \rightarrow U$, maps each session to a single user (which does not change)
$roles : S \rightarrow 2^{R \cup AR}$ maps each session s_i to a set $roles(s_i) \subseteq \{r \mid (\exists r' \geq r)[(user(s_i), r') \in UA \cup AUA]\}$ (which can change with time)
session s_i has permissions $\cup_{r \in roles(s_i)}\{p \mid (\exists r'' \leq r)[(p, r'') \in PA \cup APA]\}$
—there is a collection of constraints stipulating which values of the various components enumerated above are allowed or forbidden

Fig. 1. Summary of the RBAC96 model.

Constraints that can be imposed on the other components of RBAC are an important aspect of RBAC96. It is sometimes argued that constraints are the principal motivation for RBAC. Mutually disjoint roles, such as purchasing manager and accounts payable manager, is a common example. In most organizations (except the very smallest), the same individual will not be permitted to be a member of both roles because this creates a possibility for committing fraud. This is the well-known and time-honored principle of separation of duties. Constraints are a powerful mechanism for laying out higher-level organizational policy. Once certain roles are declared to be mutually exclusive, there does not need to be so much concern

about the assignment of individual users to roles. The latter activity can then be delegated and decentralized without fear of compromising overall policy objectives. So long as the management of RBAC is entirely centralized in a single security officer, constraints are a useful convenience; but the same effect can largely be achieved by judicious care on the part of the security officer. However, if management of RBAC is decentralized (as discussed later), constraints become a mechanism by which senior security officers can restrict the ability of users who exercise administrative privileges. This enables the chief security officer to lay out the broad scope of what is acceptable and impose it as a compulsory requirement on other security officers and users.

In ARBAC97 it is assumed that constraints will be enforced while carrying out administrative chores. Constraints specifically built into ARBAC97 are in addition to other constraints that may be specified.

It is worth emphasizing that RBAC96 distinguishes roles and permissions from administrative roles and permissions, respectively, where the latter are used to manage the former. How are administrative permissions and roles managed in turn? One could consider a second level of administrative roles and permissions to manage the first level ones, and so on. We feel such a progression of administration is unnecessary. Administration of administrative roles and permissions is under control of the chief security officer, or delegated in part to administrative roles. In this paper we take the former approach.

RBAC96 has many components. Administration of RBAC involves control over each of these components, including creation and deletion of roles, creation and deletion of permissions, assignment of permissions to roles and their removal, creation and deletion of users, assignment of users to roles and their removal, definition and maintainence of the role hierarchy, definition and maintainence of constraints, and all of these in turn for administrative roles and permissions. A comprehensive administrative model is quite complex and difficult to develop in a single step. ARBAC97 takes a piecemeal approach by developing different models for user-role assignment, permission-role assignment, and role-role assignment.

3. THE URA97 MODEL FOR USER-ROLE ASSIGNMENT

In this section we develop the URA97 model for managing user-role assignment. We define URA97 in two steps: granting a user membership in a role and revoking a user's membership. In spite of its simplicity, URA97 is quite powerful and goes much beyond existing administrative models for user-role assignment. It is also applicable beyond RBAC to user-group assignment.[3]

In the simplest case, user-role assignment can be completely centralized in a single chief security officer role. This is readily implemented in

[3]The difference between roles and groups has been hotly debated [Sandhu 1997b]. For our purpose, a group is a collection of users; whereas a role is a collection of users and permissions.

existing systems. However, this simple approach does not scale to large systems. Clearly, it is desirable to decentralize user-role assignment to some degree.

In several systems it is possible to designate a role, say junior security officer (JSO), whose members have administrative control over one or more regular roles, say A, B, and C. Thus, limited administrative authority is delegated to the JSO role. Unfortunately these systems typically allow the JSO role to have complete control over roles A, B, and C. A member of JSO can not only add users to A, B. and C, but also delete users from these roles and add and delete permissions. Moreover, there is no control on which users can be added to the A, B, and C roles by JSO members. Finally, JSO members are allowed to assign A, B, and C as junior to any role in the existing hierarchy (as long as no cycle is introduced). All this is consistent with classical discretionary thinking, whereby members of JSO are effectively designated as "owners" of the A, B, and C roles, and therefore are free to do whatever they want to these roles.

In URA97, our goal 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 a prerequisite condition is a key part of URA97.

Definition 1. A *prerequisite condition* is a boolean expression using the usual $\wedge$ and $\vee$ operators on terms of the form x and $\overline{x}$ where x is a regular role (i.e., $x \in R$). A prerequisite condition is evaluated for a user u by interpreting x to be true if $(\exists x' \geq x)(u, x') \in UA$ and $\overline{x}$ to be true if $(\forall x' \geq x)(u, x') \notin ua$. For a given set of roles R, we let CR denotes all possible prerequisite conditions that can be formed using the roles in R.

In the trivial case, a prerequisite condition can be a tautology. The simplest nontrivial case of a prerequisite condition is a test for membership in a single role, where the single role is called a *prerequisite role*.

3.1 The URA97 Grant Model

User-role assignment is authorized in URA97 by the following relation.

Definition 2. The URA97 model controls user-role assignment by means of the relation

$$can\_assign \subseteq AR \times CR \times 2^R.$$

The meaning of $can\_assign(x, y, \{a, b, c\})$ 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

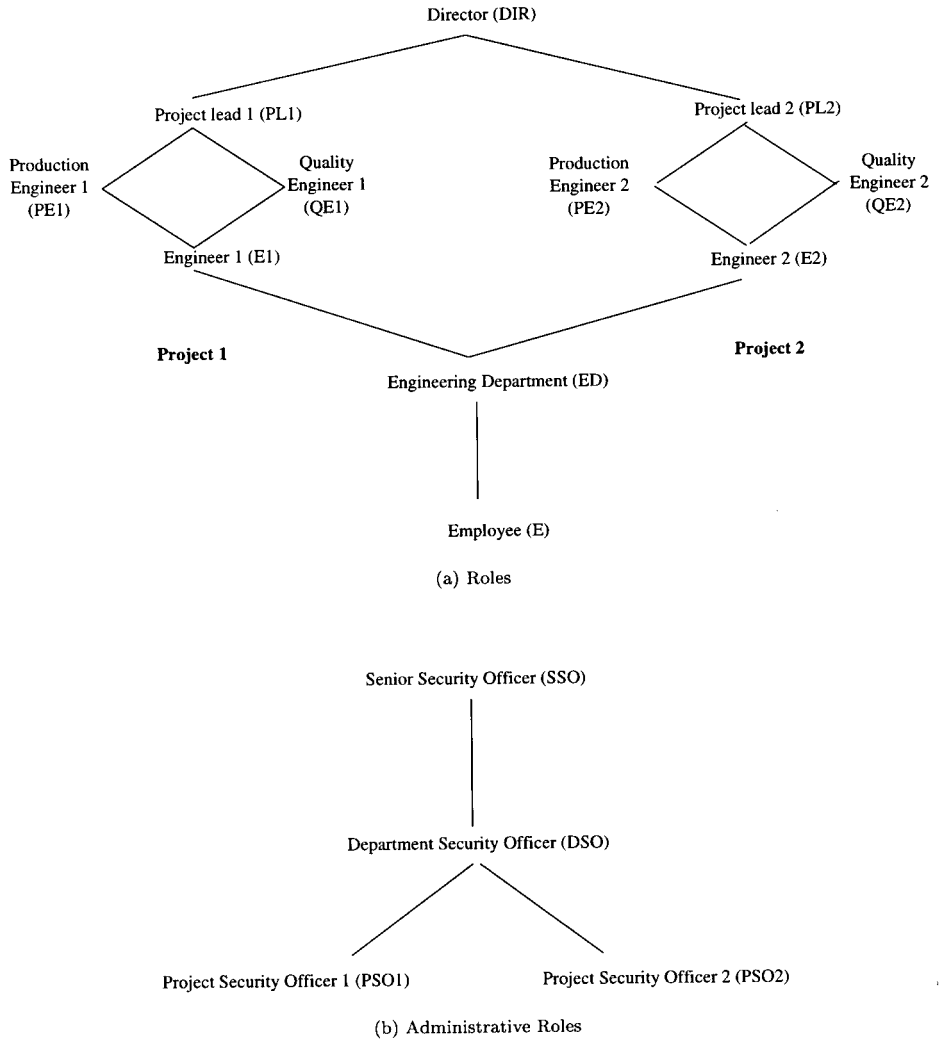


Fig. 2. Example role and administrative role hierarchies.

regular roles satisfies the prerequisite condition y to be a member of regular roles a, b, or c.[4]

To appreciate the motivation behind the *can_assign* relation, consider the role hierarchy in Figure 2(a) and the administrative role hierarchy in Figure 2(b). Figure 2(a) shows the regular roles that exist in a engineering department. There is a junior-most role E to which all employees in the organization belong. Within the engineering department there is a junior-

[4]As mentioned earlier, user-role assignment is subject to constraints, such as mutually exclusive roles or maximum membership, that may be imposed. The assignment will succeed if and only if it is authorized by *can_assign* and satisfies all relevant constraints.

most role ED and senior-most role DIR. In between there are roles for two projects within the department, project 1 on the left and project 2 on the right. Each project has a senior-most project lead role (PL1 and PL2) and a junior-most engineer role (E1 and E2). In between, each project has two incomparable roles, production engineer (PE1 and PE2) and quality engineer (QE1 and QE2).

Figure 2(a) suffices for our purpose, but this structure can, of course, be extended to dozens and even hundreds of projects within the engineering department. Moreover, each project could have a different structure for its roles. The example can also be extended to multiple departments with different structure and policies applied to each department. Moreover, there is no requirement that there be a senior-most role such as DIR in this example. Similarly, there is no requirement that there be a junior-most role.

Figure 2(b) shows the administrative role hierarchy that co-exists with Figure 2(a). The senior-most role is the senior security officer (SSO). Our main interest is in the administrative roles junior to SSO. These consist of two project security officer roles (PSO1 and PSO2) and a department security officer (DSO) role with the relationships illustrated in the figure.

For the sake of illustration, we define the *can_assign* relation shown in Table I in the role set column. Each tuple in this example has the simplest prerequisite condition for testing membership in a single role, known as the prerequisite role.

The PSO1 role has partial responsibility over project 1 roles. Let Alice be a member of the PSO1 role and Bob a member of the ED role. Alice can assign Bob to any of the E1, PE1, and QE1 roles, but not to the PL1 role. Also, if Charlie is not a member of the ED role, then Alice cannot assign him to any project 1 role. Hence, Alice has authority to enroll users in the E1, PE1, and QE1 roles, provided these users are already members of ED. Note that if Alice assigns Bob to PE1, he does not need to be explicitly assigned to E1, since E1 permissions are inherited via the role hierarchy. The PSO2 role is similar to PSO1, but with respect to project 2. The DSO role inherits the authority of PSO1 and PSO2 roles, but can further add users who are members of ED to the PL1 and PL2 roles. The SSO role can add users who are in the E role to the ED role, as well as add users who are in the ED role to the DIR role. This ensures that even the SSO must first enroll a user in the ED role before that user is enrolled in a role senior to ED. This is a reasonable specification for *can_assign*. There are, of course, lots of other equally reasonable specifications in this context. This is a matter for policy decision; our model provides the necessary flexibility.

In general, we expect that the role being assigned is senior to the role previously required of the user. That is, if we have $can\_assign(a, b, C)$ where b is a prerequisite role, then b is junior to all roles $c \in C$. We believe this is usually the case, but we do not require it in the model. This allows URA97 to be applicable to situations where there is no role hierarchy, or where such a constraint may not be appropriate.

Table I. *can_assign* with Prerequisite Roles

Admin. Role	Prereq. Role	Role Set	Role Range
PSO1	ED	{E1, PE1, QE1}	[E1, PL1)
PSO2	ED	{E2, PE2, QE2}	[E2, PL2)
DSO	ED	{PL1, PL2}	(ED, DIR)
SSO	E	{ED}	[ED, ED]
SSO	ED	{DIR}	(ED, DIR]

The notation in Table I has benefited from the administrative role hierarchy. Thus, for the DSO, we specified the role set as {PL1, PL2} and the other values are inherited from PSO1 and PSO2. Similarly for the SSO. Nevertheless, explicit enumeration of the role set is unwieldy, particularly if we scale-up to dozens or hundreds of projects in the department. Moreover, explicit enumeration is not resilient with respect to changes in the role hierarchy. Suppose a third project is introduced in the department, with roles E3, PE3, QE3, PL3, and PSO3 analogous to corresponding roles for projects 1 and 2. We can add the following row to Table I.

Admin. Role	Prereq. Role	Role Set
PSO3	ED	{E3, PE3, QE3}

This is a reasonable change when the new project and its roles are introduced into the regular and administrative role hierarchies. However, we also need to modify the row for DSO in Table I to include PL3.

Consider instead the range notation illustrated in Table I in the role range column. The role set and role range columns in Table I show the same role sets. A role range defines this set by identifying a range within the role hierarchy of Figure 1(a) by means of the familiar closed and open interval notation.

Definition 3. Role sets are specified in the URA97 model by the *range notation* given below

$$[x, y] = \{r \in R \mid x \geq r \wedge r \geq y\} \quad [x, y) = \{r \in R \mid x \geq r \wedge r > y\}$$
$$(x, y] = \{r \in R \mid x > r \wedge r \geq y\} \quad (x, y) = \{r \in R \mid x > r \wedge r > y\}$$

This notation is resilient to modifications, such as addition of a third project, in the role hierarchy, which requires addition of the following row to Table I.

Admin. Role	Prereq. Role	Role Range
PSO3	ED	[E3, PL3)

No other change is required, since the (ED, DIR) range specified for the DSO automatically picks up PL3.

Range notation is not resilient to all changes in the role hierarchy. Deletion of one of the end points of a range can leave a dangling reference and an invalid range. As we will see, RRA97 does not permit this to happen, thereby maintaining referential integrity of the range. Changes to role-role relationships could also cause a range to be drastically different from its original meaning. Nevertheless, the range notation is much more convenient than explicit enumeration. There is also no loss of generality in adopting the range notation, since every set of roles can be expressed as a union of disjoint ranges.

Strictly speaking, the role set and role range specifications of Table I are not identical. With role sets, the DSO role is explicitly authorized to enroll users in PL1 and PL2, and inherits the ability to enroll users in other project 1 and 2 roles from PSO1 and PSO2. On the other hand, with role range, the DSO role is explicitly authorized to enroll users in all project 1 and 2 roles. As it stands, the net effect is the same. However, if modifications are made to the role hierarchy or to the PSO1 or PSO2 authorizations, the effect can be different. The DSO role set authorization in Table I can be replaced by the following row, to make it more nearly identical to the specified role range.

Admin. Role	Prereq. Role	Role Set
DSO	ED	{E1, PE1, QE1, PL1, E2, PE2, QE2, PL2}

Now, even if the PSO1 and PSO2 roles in Table I are modified to the role sets {E1} and {E2}, respectively, the DSO role will still retain administrative authority over all project 1 and project 2 roles. Of course, explicit and implicit specifications will never behave exactly identically under *all* circumstances. For instance, introduction of a new project 3 will exhibit differences discussed above. Conversely, the DSO role range authorization in Table I can be replaced by the following rows to make it more nearly identical to the specified role set.

Admin. Role	Prereq. Role	Role Range
DSO	ED	[PL1, PL1]
DSO	ED	[PL2, PL2]

There is an analogous situation to the SSO role in Table I. Clearly, we must anticipate the impact of future changes when we specify the *can_assign* relation.

An example of *can_assign*, which uses prerequisite conditions rather than prerequisite roles, is shown in Table II. The authorizations for PSO1 and PSO2 were changed relative to Table I. Consider the PSO1 tuples

Table II. *can_assign* with Prerequisite Conditions

Admin. Role	Prereq. Condition	Role Range
PSO1	ED	[E1, E1]
PSO1	ED $\wedge$ $\overline{\text{QE1}}$	[PE1, PE1]
PSO1	ED $\wedge$ $\overline{\text{PE1}}$	[QE1, QE1]
PSO1	PE1 $\wedge$ QE1	[PL1, PL1]
PSO2	ED	[E2, E2]
PSO2	ED $\wedge$ $\overline{\text{QE2}}$	[PE2, PE2]
PSO2	ED $\wedge$ $\overline{\text{PE2}}$	[QE2, QE2]
PSO2	PE2 $\wedge$ QE2	[PL2, PL2]
DSO	ED	(ED, DIR)
SSO	E	[ED, ED]
SSO	ED	(ED, DIR]

(analysis for PSO2 is exactly similar). The first tuple authorizes PSO1 to assign users with prerequisite role ED into E1. The second one authorizes PSO1 to assign users with prerequisite condition ED $\wedge$ $\overline{\text{QE1}}$ to PE1. Similarly, the third tuple authorizes PSO1 to assign users with prerequisite condition ED $\wedge$ $\overline{\text{PE1}}$ to QE1. Taken together, the second and third tuples authorize PSO1 to put a user who is a member of ED into one but not both of PE1 and QE1. This illustrates how mutually exclusive roles can be enforced by URA97. PE1 and QE1 are mutually exclusive with respect to the power of PSO1. However, for the DSO and SSO, these are not mutually exclusive. Hence, the notion of mutual exclusion is relative in URA97. The fourth tuple authorizes PSO1 to put a user who is a member of both PE1 and QE1 into PL1. Of course, a user could have become a member of both PE1 and QE1 only by actions of a more powerful administrator than PSO1.

3.2 The URA97 Revoke Model

We now turn to consideration of the URA97 revoke model. The objective is to define a revoke model that is consistent with the philosophy of RBAC. This causes us to depart from classical discretionary approaches to revocation.

In the typical discretionary approach to revocation, there are at least two issues that introduce complexity and subtlety. Suppose Alice grants Bob some permission P. This is done at Alice's discretion because Alice is either the owner of the object to which P pertains or has been granted administrative authority over P by the actual owner. Alice can later revoke P from Bob. Now suppose Bob has received permission P from Alice and from Charlie. If Alice revokes her grant of P to Bob, he should still continue to retain P because of Charlie's grant. Cascading revokes is a related issue. Suppose Charlie's grant was in turn obtained from Alice, perhaps Bob's permission should end up being revoked by Alice's action. Or perhaps it should not, since Alice only revoked her direct grant to Bob but not the indirect one via Charlie, which really occurred at Charlie's discretion. A considerable literature has developed examining the subtleties that arise,

especially when hierarchical groups and negative permissions or denials are brought into play (see, for example, Bertino et al. [1997]).

The RBAC approach to authorization is quite different from the traditional discretionary one. In RBAC, users are made members of roles because of their job function or task assignment in the interests of the organization. Granting membership in a role is specifically not done at the grantor's whim. Suppose Alice makes Bob a member of a role X. In URA97, this happens because Alice is assigned suitable administrative authority over X via some administrative role Y, and Bob is eligible for membership in X due to Bob's existing role memberships (and nonmemberships), satisfying the prerequisite condition. Moreover, there are some organizational circumstances that cause Alice to grant Bob this membership. It is not merely being done at Alice's personal fancy. Now, if at some later time Alice is removed from the administrative role Y, there is clearly no reason to also remove Bob from X. A change in Alice's job function should not necessarily undo her previous grants. Presumably some other administrator, say Dorothy, will take over Alice's responsibility. Similarly, suppose Alice and Charlie both grant membership to Bob in X. At some later time, Bob is reassigned to some other project and no longer needs to be a member of role X. It is not material whether Alice, or Charlie, or both, or Dorothy revokes Bob's membership. Bob's membership in X is revoked due to a change in organizational circumstances.

We now introduce our notation for authorizing revocation.

Definition 4. The URA97 model controls user-role revocation by means of the relation

$$can\_revoke \subseteq AR \times 2^R.$$

The meaning of $can\_revoke(x, Y)$ is that a member of the administrative role x (or a member of an administrative role that is senior to x) can revoke membership of a user from any regular role $y \in Y$. Y is specified using the range notation of Definition 3. We say Y defines the *range of revocation*.

The revocation operation in URA97 is said to be *weak* because it applies only to the role that is directly revoked. Suppose Bob is a member of PE1 and E1. If Alice revokes Bob's membership from E1, he continues to be a member of the senior role PE1, and therefore can use the permissions of E1. To make the notion of weak revocation precise, we introduce the following terminology. Recall that UA is the user assignment relation.

Definition 5. Let us say a user U is an *explicit member* of role x if $(U, x) \in UA$, and that U is an *implicit member* of role x if for some $x' > x$, $(U, x') \in UA$.

RBAC96 allows a user to be an explicit and implicit member of the same role.

Weak revocation has an impact on explicit membership only. It has the straightforward meaning stated below.

Table III. Example of *can_revoke*

Admin. Role	Role Range
PSO1	[E1, PL1)
PSO2	[E2, PL2)
DSO	(ED, DIR)
SSO	[ED, DIR]

Table IV. Example of Strong Revocation

User	E1	PE1	QE1	PL1	DIR	Alice revokes user from E1
Bob	Yes	Yes	No	No	No	removed from E1, PE1
Cathy	Yes	Yes	Yes	No	No	removed from E1, PE1, QE1
Dave	Yes	Yes	Yes	Yes	No	no effect
Eve	Yes	Yes	Yes	Yes	Yes	no effect

Weak revocation. Let Alice have a session with administrative roles $A = \{a_1, a_2, \ldots, a_k\}$ and let Alice try to weakly revoke Bob from role x. If Bob is not an explicit member of x, this operation has no effect; otherwise if there exists a *can_revoke* tuple (b, Y) such that there exists $a_i \in A$, $a_i \geq b$ and $x \in Y$ revoke Bob's explicit membership in x.

Strong revocation of U's membership in x requires that U be removed not only from explicit membership in x, but also from explicit (or implicit) membership in all roles senior to x. However, strong revocation in URA97 takes effect only if all implied revocations upward in the role hierarchy are within the revocation range of the administrative roles active in that session. Strong revocation is theoretically equivalent to a series of weak revocations, but it is a useful and convenient operation for administrators.

We have used this principle of defining stronger operations in terms of weaker ones in ARBAC97 at several places. The general approach is to define simple and essential operations and give them a formal meaning. More complex operations are then defined in terms of the simple ones. In an implementation, however, the more complex operations can be directly implemented for performance reasons, rather than being implemented as sequences of the weaker operation. This allows us to keep a simple underlying semantics for the model, while allowing for the crafting of convenient bells and whistles.

Consider the example of *can_revoke*, shown in Table III, and interpret it in context of the hierarchies in Figures 2(a) and 2(b). Let Alice be a member of PSO1, and let this be the only administrative role she has. Alice is authorized to strongly revoke memberships of users from roles E1, PE1, and QE1. Table IV illustrates the effect of Alice's strong revocation of a user from role E1. Alice is not allowed to strongly revoke Dave and Eve from E1 because they are members of senior roles outside the scope of Alice's revoking authority. If Alice was assigned to the DSO role, she could strongly revoke Dave from E1, but would still not be able to strongly revoke

Eve's membership in E1. In order to strongly revoke Eve from E1, Alice needs to be in the SSO role.

The algorithm for strong revocation is stated in terms of weak revocation, as follows.

> *Strong revocation.* Let Alice have a session with administrative roles $A = \{a_1, a_2, \ldots, a_k\}$, and let Alice try to strongly revoke Bob from role x. Find all roles $y \geq x$ such that Bob is an explicit member of y. Weakly revoke Bob from all such y as if Alice did this weak revoke in this session. If any of the weak revokes fail, then Alice's strong revoke has no effect, otherwise all weak revokes succeed.

An alternate approach is to do only those weak revokes that succeed and ignore the rest. URA97 allows this as an option to the behavior identified above. In general, we can give flexible meaning to strong revocation as long as the meaning can be expressed in terms of weak revocation.

So far we have looked at the cascading of revocation upward in the role hierarchy. There is also a downward cascading effect. Consider Bob in our example who is a member of E1 and PE1. Suppose further that Bob is an explicit member of PE1, and thereby an implicit member of E1. What happens if Alice revokes Bob from PE1? If we remove (Bob, PE1) from the *UA* relation, Bob's implicit membership in E1 will also be removed. On the other hand, if Bob is an explicit member of PE1 and also an explicit member of E1, then Alice's revocation of Bob from PE1 does not remove him from E1. The revoke operations we have defined in URA97 have the following effect.

> *Property 1.* Implicit membership in a role a is dependent on explicit membership in some senior role $b > a$. Therefore, when explicit membership of a user is revoked from b, implicit membership is also automatically revoked on junior role a, unless there is some other senior role $c > a$ in which the user continues to be an explicit member.

Note that our examples *can_assign* in Table I and *can_revoke* in Table III are complementary, in that each administrative role has the same range for adding users and removing users from roles. Although this would be a common case, we do not impose it as a requirement on our model.

3.3 Discussion

To summarize, URA97 controls user-role assignment by means of the relation $can\_assign \subseteq AR \times CR \times 2^R$. Role sets are specified using the range notation of Definition 3. Assignment has a simple behavior whereby $can\_assign(a, b, C)$ authorizes a session with an administrative role $a' \geq a$ to enroll any user who satisfies the prerequisite condition b into any role $c \in C$. The prerequisite condition is a boolean expression using the usual $\wedge$ and $\vee$ operators on terms of the form x and $\overline{x}$, respectively, denoting membership and nonmembership in regular role x. Revocation is

controlled in URA97 by the relation *can_revoke* $\subseteq AR \times 2^R$. Weak revocation applies only to explicit membership in a single role. Strong revocation cascades upwards in the role hierarchy. In both cases, revocation cascades downwards as noted in Property 1.

URA97 does not deal with creation and deletion of users. URA97 treats this as a centralized task in an enterprise. Everything else thereafter can be based on role-based prerequisite conditions.

As we discussed earlier, when a user's administrative roles are revoked that user's assignments and revocations remain in effect because they were done for organizational reasons and not at the user's whim. A related issue is what happens when the prerequisite condition that authorized Alice to assign Bob to a role gets changed. Say that Alice as PSO1 assigns Bob to PE1, as per the second PSO1 tuple of Table II. Later, Bob is somehow made a member of QE1, perhaps by a user in DSO or SSO role. This assignment negates the prerequisite condition that enabled Alice to do her assignment. Bob's membership in PE1 will nevertheless continue. We feel this is the appropriate action. The prerequisite conditions of URA97 (and at other places in ARBAC97) are not invariants that hold for all time. They are simply enabling conditions at the moment that assignment is made.

As another example of the enabling, but not invariant, nature of prerequisite conditions consider the following in context of the *can_assign* relation of Table I. Suppose Alice as PSO1 enrolls Bob into PE1 due to his prerequisite membership in ED. Later, Charles as SSO revokes Bob from ED. Should Alice's assignment of Bob to PE1 be negated, since the prerequisite condition has been negated? It depends on Charles' intention, which in turn depends on the organizational reason for this revocation. If Charles really needs to clear Bob out from the engineering department, the correct course of action is a strong revocation of Bob from ED. If Charles does a weak revoke of Bob's explicit membership in ED, he leaves open the option that Bob will continue to participate in engineering department roles until such time Bob is revoked from all of them (say by project security officers). This latter option can be useful in allowing Bob to gracefully leave the engineering department without an abrupt termination. In such cases it might be useful for Charles to be able to freeze Bob's membership in engineering department roles so that Bob cannot be assigned to new roles. This can be done using prerequisite conditions. A role called EF (for engineering frozen) can be defined and nonmembership in EF required in the prerequisite condition of all *can_assign* tuples that authorize users to be assigned to engineering department roles.

There is a lack of symmetry between *can_assign* and *can_revoke*, in that *can_assign* involves prerequisite conditions but *can_revoke* does not. There are situations where prerequisite conditions could be useful in *can_revoke*. For instance, suppose there is another department called sales, similar to an engineering department with SD as its junior-most role and SDIR as senior-most. Consider the *can_revoke* tuple with prerequisite condition given below.

Admin. Role	Prereq. Condition	Role Range
DSO	ED	[SD, DIR]

This tuple says that if Bob is a member of ED, then DSO (of engineering department) can strongly revoke Bob from SD. This can be useful if Bob is in transition from the sales department to the engineering department. Bob can retain membership in sales department roles for some time, but this membership can be revoked by administrators of the engineering department (in case Bob spends too much time in sales department activity). A reciprocal rule for SDSO (department security officer of sales department) to revoke Bob from engineering department roles could result in interesting race conditions. Thus, the prerequisite condition could be further qualified to ED $\wedge$ $\overline{\text{EF}}$ where EF is the frozen engineering role discussed above. The utility of prerequisite conditions in *can_revoke* tuples is not clear without further theoretical analysis or empirical evidence, so we decided to omit this feature from ARBAC97.

Our final observation concerns range notation. In our example hierarchy, we have senior-most and junior-most roles, which means we can identify ranges conveniently. Suppose that in Figure 1 there is no DIR role, but that we would like to give DSO the authority specified in Table I. Since DIR does not exist, we can invent a symbol *top*, so we can say (ED,*top*) rather than (ED, DIR). A similar *bottom* symbol would also be useful if there were no junior-most role. Other enhancements to range notation can also be imagined, such as allowing the complement of a range to be specified. For simplicity, we omit these enhancements.

4. THE PRA97 MODEL FOR PERMISSION-ROLE ASSIGNMENT

PRA97 is concerned with role-permission assignment and revocation. From the perspective of a role, users and permissions have a similar character. They are essentially entities that are brought together by a role. Hence, we propose PRA97 to be a dual of URA97. The notion of a prerequisite condition is identical to that in URA97, except the boolean expression is now evaluated for membership and nonmembership of a permission in specified roles.

Definition 6. Permission-role assignment and revocation, respectively, are authorized in PRA97 by the following relations:

$$can\_assignp \subseteq AR \times CR \times 2^R$$

$$can\_revokep \subseteq AR \times 2^R.$$

The meaning of $can\_assignp(x, y, Z)$ is that a member of the administrative role x (or a member of an administrative role that is senior to x) can assign a permission whose current membership, or nonmembership, in regular roles satisfies the prerequisite condition y to regular roles in range

Table V. Example of *can_assignp* and *can_revokep*

Administrative Role	Prerequisite Condition	Role Range
DSO	DIR	[PL1, PL1]
DSO	DIR	[PL2, PL2]
PSO1	PL1 $\wedge$ $\overline{\text{QE1}}$	[PE1, PE1]
PSO1	PL1 $\wedge$ $\overline{\text{PE1}}$	[QE1, QE1]
PSO2	PL2 $\wedge$ $\overline{\text{QE2}}$	[PE2, PE2]
PSO2	PL2 $\wedge$ $\overline{\text{PE2}}$	[QE2, QE2]

(a) *can_assignp*

Administrative Role	Role Range
DSO	(ED, DIR)
PSO1	[QE1, QE1]
PSO1	[PE1, PE1]
PSO2	[QE2, QE2]
PSO2	[PE2, PE2]

(b) *can_revokep*

Z.[5] The meaning of *can_revokep*(x, Y) is that a member of the administrative role x (or a member of an administrative role that is senior to x) can revoke membership of a permission from any regular role $y \in Y$.

Table V shows examples of these relations. The DSO is authorized to take any permission assigned to the DIR role and make it available to PL1 or PL2. Thus a permission can be delegated downward in the hierarchy. PSO1 can assign permissions from PL1, either PE1 or QE1, but not to both. The remaining rows in Table V(a) are similarly interpreted. Table V(b) authorizes DSO to revoke permissions from any role between ED and DIR. PSO1 can revoke permissions from PE1 and QE2, and similarly for PSO2.

Revocation in PRA97 is weak, so permissions may still be inherited after revocation. Strong revocation can be defined in terms of weak revocation as in URA97. Strong revocation of a permissions cascades down the role hierarchy, in contrast to cascading up for revocation of user membership. For completeness, the formal definitions are given below.

Definition 7. Let us say a permission P is *explicitly assigned* to role x if $(P, x) \in PA$, and that P is *implicitly assigned* to role x if for some $x' < x$, $(P, x') \in PA$.

Weak revocation. Let Alice have a session with administrative role $A = \{a_1, a_2, \ldots, a_k\}$, and let Alice try to weakly revoke permission P from role x. If P is not explicitly assigned to x, this operation has no effect; otherwise if there exists a *can_revokep* tuple (b, Y) such that there exists $a_i \in A$, $a_i \geq b$ and $x \in Y$, revoke P's explicit assignment to x.

[5]Permission-role assignment may be subject to additional constraints, just as user-role assignment is.

Strong revocation. Let Alice have a session with administrative roles $A = \{a_1, a_2, \ldots, a_k\}$ and let Alice try to strongly revoke permission P from role x. Find all roles $y \leq x$ such that P is explicitly assigned to y. Weakly revoke P from all such y as if Alice did this weak revoke in this session. If any of the weak revokes fail, then Alice's strong revoke has no effect; otherwise all weak revokes succeed.

5. THE RRA97 MODEL FOR ROLE-ROLE ASSIGNMENT

In this setion we develop the RRA97 model for role-role assignment.

5.1 Abilities, Groups, and UP-Roles

For role-role assignment, we distinguish three kinds of roles, as follows:

—*Abilities* are roles that can only have permissions and other abilities as members.

—*Groups* are roles that can only have users and other groups as members.

—*UP-Roles* are roles that have no restriction on membership, i.e., their membership can include users, permissions, groups, abilities, and other UP-roles.

The term UP-roles signifies user and permission roles. We use the term role to mean all three kinds of roles or to mean UP-roles only, as determined by context. The three kinds of roles are mutually disjoint and are identified, respectively, as A, G, and UPR.

The main reason for distinguishing among the three kinds of roles is that different administrative models apply to establish relationships among them. The distinction was motivated in the first place by abilities. An ability is a collection of permissions that should be assigned as a single unit to a role. For example, the ability to open an account in a banking application will encompass many different individual permissions. It does not make sense to assign only some of these permissions to a role, since the entire set is needed to do the task properly. The idea is that application developers package permissions into collections, called abilities, which must be assigned together as a unit to a role.

The function of an ability is to collect permissions together so that administrators can treat them as a single unit. Assigning abilities to roles is therefore very much like assigning permissions to roles. For convenience, it is useful to organize abilities into a hierarchy (i.e., partial order). Hence the PRA97 model can be adapted to produce the very similar ARA97 model for ability-role assignment.

Once the notion of abilities is introduced, by analogy there should be a similar concept on the user side. A group is a collection of users who are assigned as a single unit to a role. Such a group can be viewed as a team, which is a unit even though its membership may change over time. Groups can also be organized in a hierarchy. For group-role assignment, we adapt the URA97 model to produce the GRA97 model for group-role assignment.

This leads to the following models.

Definition 8. Ability-role assignment and revocation are, respectively, authorized in ARA97 by

$$can\_assigna \subseteq AR \times CR \times 2^A$$

$$can\_revokea \subseteq AR \times 2^A.$$

Definition 9. Group-role assignment and revocation are, respectively, authorized in GRA97 by

$$can\_assignG \subseteq AR \times CR \times 2^G$$

$$can\_revokeg \subseteq AR \times 2^G.$$

For these models, CR is interpreted as the collection of prerequisite conditions formed using roles in UPR, and the prerequisite conditions are interpreted with respect to abilities and groups, respectively. Membership of an ability in a UP-role is true if the UP-role dominates the ability, and false otherwise. Conversely, membership of a group in a UP-role is true if the UP-role is dominated by the group, and false otherwise.

Assigning an ability to an UP-role is mathematically equivalent to making the UP-role an immediate senior of the ability in the role-role hierarchy. Abilities can only have UP-roles or abilities as immediate seniors and can only have abilities as immediate juniors. In a dual manner, assigning a group to an UP-role is mathematically equivalent to making the UP-role an immediate junior of the group in the role-role hierarchy. Groups can only have UP-roles or groups as immediate juniors and can only have groups as immediate seniors. With these constraints, the ARA97 and GRA97 models are essentially identical to the PRA97 and URA97 models, respectively. This leaves us with the problem of managing relationships between UP-roles and other UP-roles, between abilities and other abilities, and between groups and other groups. We use the same model for all three cases. In the rest of the paper we understand the term role to mean one of UP-role, ability, or group, and we use the symbol R to mean the set of UP-roles, abilities, or groups as appropriate. Our examples are cast in terms of UP-roles.

5.2 The Can_Modify Relation

Decentralization of administrative authority requires that members of different administrative roles have authority over different parts of the hierarchy. Authority over a part of the role hierarchy implies autonomy in modifying the internal role structure of that part. That includes the creation and deletion of roles, as well as alternation of role-role relationships by adding or deleting the edges. For example, in Figure 2 we would like the DSO to configure changes in the role hierarchy between DIR and ED. The PSO1 would manage the hierarchy between PL1 and E1, whereas

Table VI. Example of *can_modify*

Admin. Role	UP-Role Range
DSO	(ED, DIR)
PSO1	(E1, PL1)
PSO1	(E2, PL2)

PSO2 would manage the part between PL2 and E2. This leads to the following notion of a can_modify relation.

Definition 10. In RRA97 role creation, role deletion, edge insertion, and edge deletion are all authorized by the the following relation

$$can\_modify \subseteq AR \times 2^R.$$

In this context, subsets of R are identified by the range notation, but limited to open ranges that do not include the endpoints.

Table VI illustrates an example of *can_modify*, relative to the hierarchies of Figure 2. The meaning of $can\_modify(x, Y)$ is that a member of the administrative role x (or a member of an administrative role that is senior to x) can create and delete roles in the range Y and can modify relationships between roles in Y. The examples in the rest of the paper are all in context of Figure 2 and Table VI. For purposes of our example, we ignored PSO2 in this table and instead authorized PSO1 to manage the roles of both projects. This illustrates how a single administrative role can be authorized to control multiple pieces of the role hierarchy.

The semantics of the four operations—create role, delete role, insert edge, and delete edge—are described in subsequent subsections. Some of the important intuitive ideas are mentioned here in anticipation. In particular, none of these operations is allowed to introduce a cycle in the hierarchy.

Creation of a new role requires the specification of its immediate parent and child in the existing hierarchy. Thus PSO1 can create a new role with immediate parent PL1 and immediate child E1, or a new role with immediate parent PL1 and immediate child PE1, and so on. Generally, the immediate parent and immediate child must fall within the range or be one of the endpoints as specified in can_modify. Since creation of a role also introduces two edges in the hierarchy, it is not possible to use any two roles as the immediate parent and immediate child. Clearly we do not want this operation to introduce a cycle in this manner. As we will see, we also impose additional restrictions to prevent undesirable side effects of role creation.

Deletion of a role leaves relationships between the parents and children of the deleted role unchanged. So if DSO deletes E1, PE1 and QE1 continue to be senior to ED after deletion of E1, as such deletion does not pose a problem. However, deletion of E1 leaves dangling references in Table VI, since the range (E1, PL1) no longer exists. In general, some roles are

referenced in various relations in URA97, PRA97, and RRA97. If these roles are actually deleted, we will have dangling references. Our approach is to prohibit deletion that would cause a dangling reference. Roles that cannot be deleted due to this reason can be deactivated, so that they can be phased out later by adjusting the references that prevent deletion. Furthermore, when a role is deleted, we need to do something about the users and permissions that are directly assigned to this role.

Insertion of an edge is meaningful only between incomparable nodes. Thus insertion of an edge from PL1 to E1 has no meaning, whereas insertion of an edge from PE1 to QE1 does. As we will see there are edges that should not be inserted because they can lead to anomalous side effects later.

Likewise, deletion of an edge is meaningful only if that edge is not transitively implied by other edges. For example, deletion of the edge PL1 to E1 is meaningless and has no impact on the hierarchy. Deletion of the edge QE1 to E1 does change the hierarchy. Edge deletion only applies to a single edge and does not carry over to implied transitive edges. For example, deletion of the edge QE1 to E1 makes QE1 and E1 incomparable, but QE1 continues to be senior to ED.

More sophisticated forms of these operations can be constructed out of the basic ones defined here. In these basic operations, roles and edges are created and destroyed one at a time. This approach is analogous to the definition of weak revocation in URA97 and PRA97, from which various forms of strong revocation can be constructed. Similarly, in RRA97, more complex operations can be constructed in terms of these basic ones. Our discussion here is limited to the basic operations.

5.3 Restrictions on Can_Modify

The relation can_modify confers authority to administrative roles to change the role hierarchy. We would like to restrict this authority to maintain global consistency of authorization. The issue of dangling references has already been raised, and RRA97 will not allow dangling references to occur. But this is not enough.

Consider the example of Figure 3, which is identical to Figure 2(a), except for additional edges shown in dashed lines that also bring in additional roles. Now if PSO1, who has authority over the range (E1, PL1) makes PE1 junior to QE1 by introducing an edge, the effect is to indirectly introduce a relationship between X and Y roles. The role PSO1 does not have the authority to create this relationship, so this is an anomalous side effect. We should either restrict the authority of the administrative role (in our example DSO) that introduced X and Y roles in the first place, or PSO1 should be prevented from introducing relationships that make PE1 junior to QE1 (and indirectly Y junior to X). In general, administrative roles are given autonomy within a range, but only so far as the global side effects are acceptable.

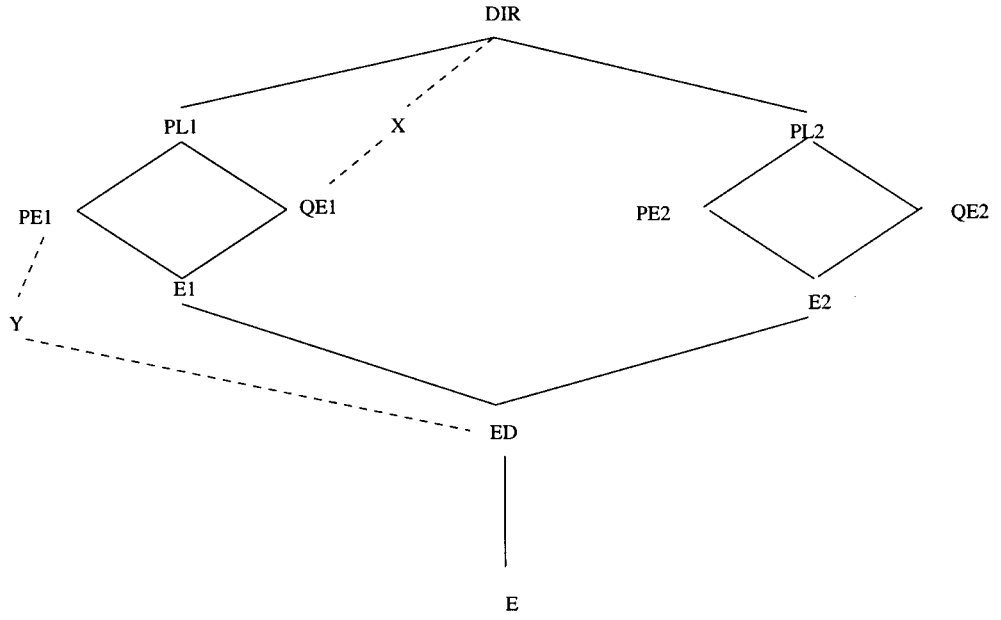


Fig. 3. Out of range impact.

To formally state these restrictions on the authority of the administrative roles, we introduce the concepts of authority range, encapsulated authority range, and create range.

5.4 Concept of Range

The concept of range is very important in RRA97. It is formally defined as follows.

Definition 11. A *range of roles* is defined by giving lower bound x and upper bound y, where y $>$ x. Formally, $(x, y) = \{z : R | x < z < y\}|$. We say x and y are the *end points* of the range.

This is similar to the notion of a range we used in URA97 and PRA97, except that in RRA97 a range does not include the end points. In URA97 and PRA97, a range may optionally exclude or include the lower or upper bound or both.

In Figure 2, (E1, PL1), (E2, PL2), and (ED, DIR) are different ranges. The range (ED, DIR) contains the roles that constitute ranges (E1, PL1) and (E2, PL2). We say ranges (E1, PL1) and (E2, PL2) are junior to range (ED, DIR).

Definition 12. For two ranges Y and Y' if $Y \subset Y'$, then Y is a *junior range* to Y', and Y' is a *senior range* to Y.

Here Y is a proper subset of Y'. This eliminates the possibility of a range being junior or senior to itself. This makes later definitions more convenient.

If two ranges Y and Y' in the role hierarchy are such that one is not junior to the other, then they are either incomparable or partially overlapping. Formal definitions of partially overlapping and incomparable ranges follow.

Definition 13. Ranges Y and Y' *partially overlap* if $Y \cap Y' \neq \phi$ and $Y \not\subset Y'$ and $Y' \not\subset Y$. Ranges Y_1 and Y_2 are said to be *incomparable* if $Y_1 \cap Y_2 = \phi$.

Note that incomparable ranges may have one common end point.

5.5 Authority Range and Encapsulated Authority Range

The members of administrative roles are authorized to modify a certain range of roles in role hierarchy. These ranges are called authority ranges.

Definition 14. Any range referenced in the can_modify relation is called an *authority range*.

To ensure that administrative authority over authority ranges does not overlap, we introduce the following restriction.

Definition 15. In RRA9, authority ranges do not partially overlap.

Note that an administrative role may have more than one authority range. Table VI shows that DSO has authority over the range (ED, DIR). In Figure 2, the authority range (ED, DIR) has two junior authority ranges, (E1, PL1) and (E2, PL2). Since these junior authority ranges are completely contained within the authority range for DSO, DSO has authority over these junior authority ranges as well. In other words, DSO has inherited the authority over the ranges (E1, PL1) and (E2, PL2).

Our model allows an administrative role to have authority over more than one incomparable authority range. Table VI shows that PSO1 has authority over two incomparable authority ranges, namely (E1, PL1) and (E2, PL2).

Consider Figure 3 again. To maintain consistency, we observed that either DSO should not be allowed to create roles X or Y in the role hierarchy, or PSO1 should not be allowed to make PE1 junior to QE1. In the latter case, the autonomy of PSO1 to manage its authority range is interfered with by DSO's actions. While this is a possibility, we pursue the former case here. Decentralization of authority and autonomy requires that all inward and outgoing edges from an authority range should only be directed to and from the end points of the authority range. The concept of encapsulation of authority range serves this purpose.

Definition 16. A range (x, y) is said to be *encapsulated* if $\forall r1 \in (x, y) \wedge \forall r2 \notin (x, y)$; we have $r2 > r1 \Leftrightarrow r2 > y$ and $r2 < r1 \Leftrightarrow r2 < x$.

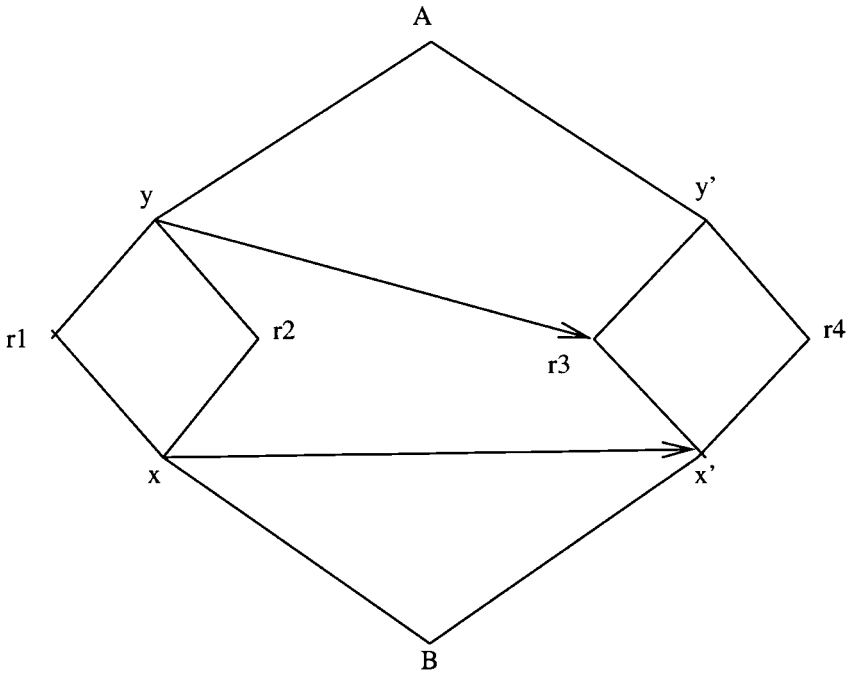


Fig. 4. Encapsulated range (x, y).

Intuitively, an encapsulated range is one in which all roles have identical relations to roles outside of the range. The intuition in RRA97 is that an encapsulated range is the correct unit for autonomous management of role-role relationships within the range. All authority ranges in RRA97 are required to be encapsulated.

Definition 17. In RRA97, authority ranges must be encapsulated.

Figures 4 and 5, respectively, show examples of an encapsulated and nonencapsulated range (x, y). The range (x, y) in Figure 4 consists of the roles {r1, r2}. All roles not in this range (such as A or B) that are senior to r1 or r2 are also senior to y. Similarly, all roles outside the range and junior to r1 or r2 are junior to x. This satisfies the above definition, so range (x, y) is encapsulated. In case of Figure 5, the range (x, y) consists of roles {r1, r2, r3}. The role y' that is senior to r3 is not senior to y. Hence (x, y) is not encapsulated.

5.6 Role Creation

As discussed earlier, creation of a role requires specification of the new role's immediate parent and child. If the immediate parent and child are the end points of an authority range, there is no difficulty. More generally, we wish to allow creation of a new role such that its immediate parent and child are within the authority range, rather than being at the end points.

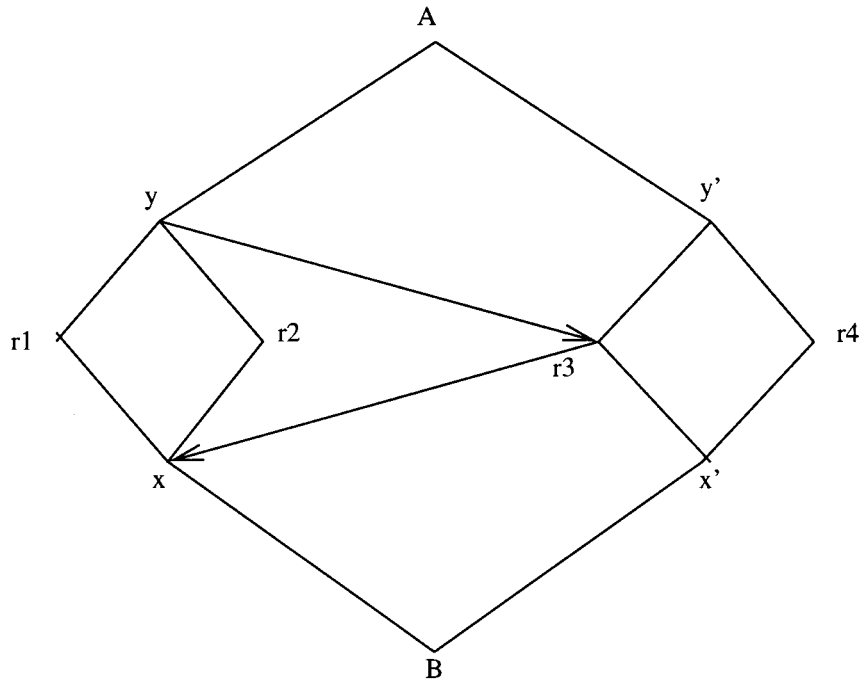


Fig. 5. Nonencapsulated range (x, y).

Thus PSO1 can create a new role with parent PL1 and child PE1. However, if DSO exercises this power, we can end up with the undesirable situation illustrated in Figure 3. To prevent this, we introduce the following notions.

Definition 18. The *immediate authority range* of role r written $AR_{immediate}(r)$ is the authority range (x, y) such that $r \in (x, y)$, and for all authority ranges (x', y') junior to (x, y), we have $r \notin (x', y')$.

Definition 19. The range (x, y) is a *create range* if $AR_{immediate}(x) = AR_{immediate}(y)$, or x is an end point of $AR_{immediate}(y)$, or y is an end point of $AR_{immediate}(x)$.

Note that only comparable roles constitute a create-range.

Consider Figure 6. Let (B, A) and (x, y) be authority ranges, whereas (x', y') is not an authority range. The ranges marked by the dotted lines, i.e., (r3, A), (x, A), and (B, y) are create ranges. However, (r1, A) or (r2, A) do not satisfy the conditions, and thereby are not create ranges.

In RRA97, we require that the immediate parent and child of a new role must be a create range in the hierarchy prior to creation of the new role.

Definition 20. In RRA97, the immediate parent and immediate child of a new role must constitute a create range prior to creation of the new role.

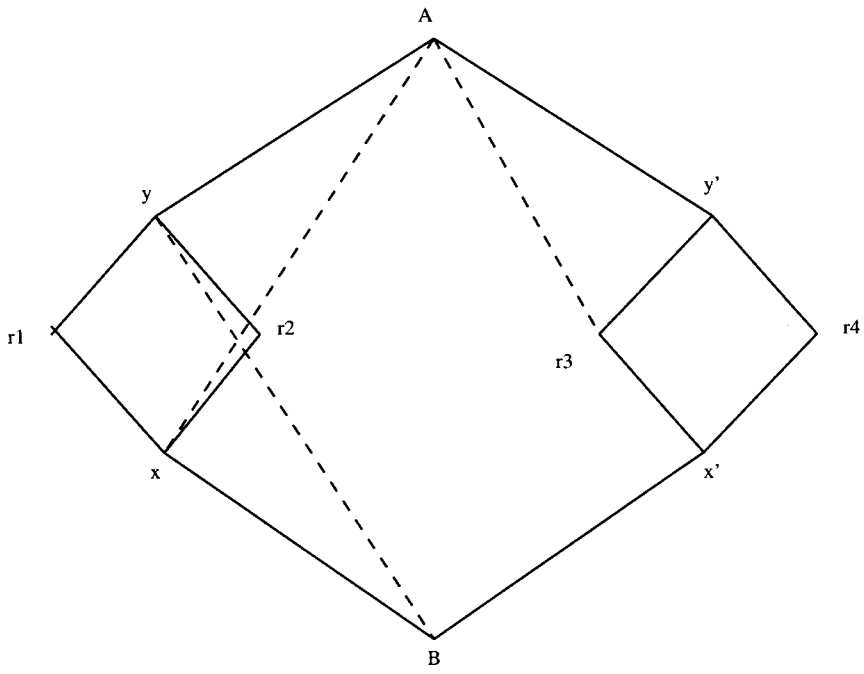


Fig. 6. Create range.

Roles can be created outside the authority ranges or without a parent or child only by the chief security officer. In general, the chief security officer can do arbitrary modifications.

5.7 Role Deletion

Deletion of roles in a hierarchy is a complicated process. Our assumption is that a role in an authority range can be deleted by the administrator of that range. It does not matter how this role got there.

ARBAC97 defines some authorization relations such as can_assign, can_revoke, and can_modify. If the roles specified as end points of the role ranges of these relationships are deleted, we will leave dangling references to nonexisting roles. The ranges with these deleted end points become meaningless. To avoid this problem, RRA97 provides two alternatives.

(1) Roles referred in can_assign, can_revoke, and can_modify relationships cannot be deleted.

(2) Roles referred in 1 above can be made inactive (explained in the next paragraph) whenever deleting them is needed. The advantage of deactivating roles is that reference to nonexisting roles is avoided, and at the same time the purpose of deletion is achieved.

A role is said to be inactive if a user associated to it cannot activate it in a session. The edges to and from the inactive role, its associated permis-

sions and assigned users remain unchanged. While a user assigned to an inactive role cannot activate it, the permissions associated with that role are still inherited by senior roles. In this way the hierarchy is not changed, but at the same time a partial effect of deletion is achieved.

RRA97 allows both of the above alternatives. Regular users cannot invoke inactive roles, but administrators can revoke users and permissions from these roles. These roles can be made empty, but cannot be deleted from the hierarchy until the references preventing deletion are suitably adjusted. Other roles in the role hierarchy can be deleted.

In case of deletion of a role, we need to preserve the permissions and users assigned to the role. RRA97 provides two alternatives to deletion of roles.

(1) Roles can be deleted only if they are empty.

(2) Delete a nonempty role, but at the same time take care of the assigned permissions and associated users, as follows: assign permissions to the immediate senior roles and assign users to immediate junior roles.

5.8 Edge Insertion

Now let us explain how the model deals with the insertion of edges in the role-to-role relationships. Insertion of transitive edges has no effect, so we only consider edges inserted between incomparable roles. When an edge is inserted, we must ensure that encapsulation of authority range is not violated. We have the following rules.

—The roles, between which the edge is inserted, must have same immediate authority range; or

—if the new edge connects a role in one authority range to a role outside the range, encapsulation of the authority range must not be violated.

For example, in Figure 5 assume edges (y, r3) and (r3, x) are initially not present and that (x, y) and (B, A) are authority ranges. Insertion of the edge (y, r3) does not pose any problem. However, in presence of this edge, insertion of edge (r3, x) violates encapsulation of authority range (x, y), hence it must not be allowed. Similarly, in the presence of (r3, x), the edge (y, r3) is not allowed. This leads to the following formal definition for insertion of an edge.

Definition 21. A new edge AB can be inserted between incomparable roles A and B

—if $AR_{immediate}(A) = AR_{immediate}(B)$, or

—if (x, y) is an authority range such that $(A = y \wedge B > x) \vee (B = x \wedge A < y)$, then insertion of AB must preserve encapsulation of (x, y).

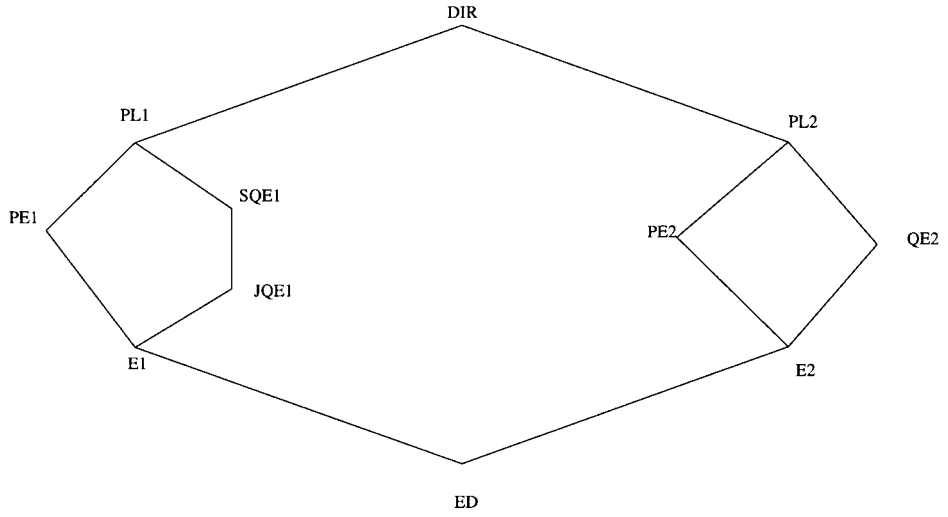


Fig. 7. Before deletion of edge from SQE1 to JQE1.

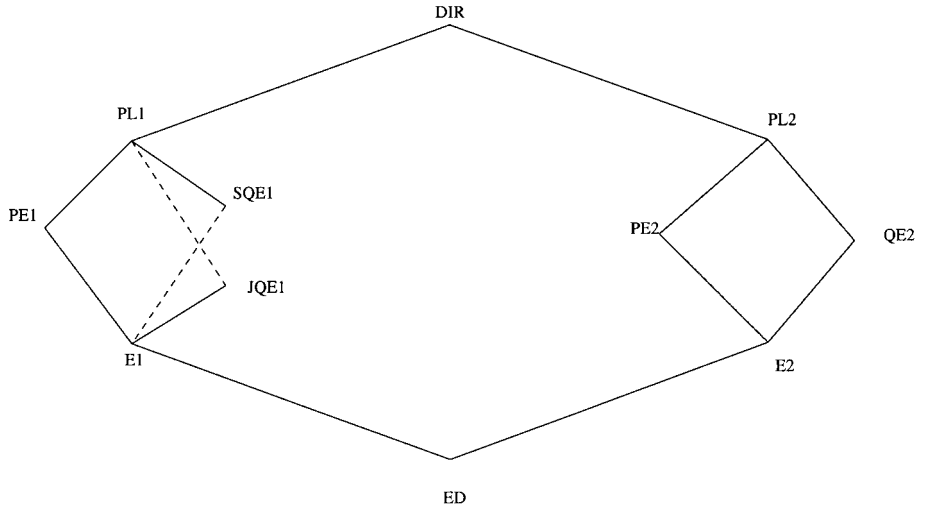


Fig. 8. After edge deletion.

5.9 Edge Deletion

Deletion of transitive edges does not change the hierarchy, so their deletion is meaningless. In RRA97, we consider only those edges for deletion that are in transitive reduction of the hierarchy. If edge AB is not in transitive reduction, then it is not a candidate for deletion.[6] For example, in Figure 7, deletion of the edge SQE1 to JQE1 will change the hierarchy. Edge deletion

[6]Other models and applications do not have this restriction. For example, Oracle allows insertion and deletion of transitive edges [Ramaswamy and Sandhu 1998].

applies to a single edge only and does not carry over to implied transitive edges. As discussed in the general rules for edge deletion, RRA97 keeps transitive edges after deletion intact. For example, deletion of the edge SQE1 to JQE1 makes SQE1 and JQE1 incomparable, but SQE1 continues to be senior to E1 and JQE1 junior to PL1 (shown in Figure 8).

There is one special case that needs to be considered. If the edge being deleted is between the end points of an authority range, deletion of the edge will disrupt the authority range and cause inconsistency in the model. Hence this operation is disallowed.

6. DISCUSSION

The ARBAC97 model was developed in a piecemeal manner. A major goal of this research is to make the model implementable using existing commercial and public-domain platforms. URA97 is described in Sandhu and Bhamidipati [1997] and implemented on several platforms. The first implementation was on Oracle [Sandhu and Bhamidipati 1999]. Similar implementations could be developed for other database management systems [Ramaswamy and Sandhu 1998]. Implementations of URA97 were also demonstrated on Unix [Sandhu and Ahn 1998a] and on Windows NT [Sandhu and Ahn 1998b]. The NIST implementation of RBAC on the Web [Ferraiolo et al. 1999] has been extended to accommodate URA97 [Sandhu and Park 1998]. PRA97 was implemented on Oracle [Sandhu and Bhamidipati 1998], and should be amenable to implementations on other database management platforms. Implementing PRA97 on Unix or Windows NT is not so straightforward, because these operating systems use access control lists to represent permissions. We need better support from the underlying operating system to effectively enforce PRA97. Similarly, we need better infrastructure on the Web to implement PRA97. RRA97 has not been implemented so far. Based on our experience with URA97, we feel that RRA97 can be implemented on the platforms that URA97 was implemented on, without any major roadblock. All this indicates that ARBAC97 is a very practical model.

Several extensions can be imagined for ARBAC97. Some are discussed in the context of URA97 at the end of Section 3. An obvious one is the incorporation of prerequisite conditions in *can_revoke* relations. In PRA97, it may be useful to distinguish permissions that can be delegated from those that cannot. For example, PL1 may have sensitive permissions that cannot be delegated by PSO1 to PE1, but others can be. This distinction can be represented by prerequisite conditions, by having a role containing the sensitive permissions and testing for nonmembership in this role in every prerequisite condition. On the other hand, a more direct solution to this requirement could be useful. Similar considerations apply on the user side to URA97. For RRA97, we could investigate more powerful operations that can be built out of the basic ones defined in this paper.

The details of how ARBAC97 would be configured are not discussed in this paper. Techniques for designing roles, administrative roles, prerequi-

site conditions, and so on, require further research. ARBAC97 has demonstrated that sophisticated administrative models can be supported by commercially available technology. However, techniques to profitably use such a model remain to be developed. We expect to invest future efforts in this direction.

7. CONCLUSION

This paper has described the motivation, intuition, and formal definition of a new role-based model, called ARBAC97, for RBAC administration. ARBAC97 has three components: URA97 for user-role assignment, PRA97 for permission-role assignment, and RRA97 for role-role assignment.

URA97 and PRA97 are exact duals of each other, and are based on the notions of prerequisite conditions and role ranges. Both models incorporate a notion of revocation that does not seek to undo actions taken by a user when that user's administrative roles are later revoked. Also, revocation is independent of who assigned the user or permission to the role. This reflects a role-oriented style, in contrast to a discretionary style.

RRA97 recognizes three kinds of roles, called abilities (roles that are assigned permissions only), groups (roles that are assigned users only), and UP-roles (roles that are assigned permissions and users). ARA97 and GRA97 deal with ability to UP-role assignment and group to UP-role assignment, respectively, and are very similar to PRA97 and URA97, respectively. The component dealing with role-role assignment proper is also (at the risk of some confusion) called RRA97. It deals with modifications of relationships between the same kinds of roles: UP-roles to UP-roles, groups to groups, and UP-roles to UP-roles. The same model applies in all three cases.

URA97 has been implemented on several platforms (Oracle, Unix, Windows NT, and the Web), while PRA97 has been implemented on Oracle. Implementation of RRA97 is quite feasible on the same platforms. Thus ARBAC97 is a practically feasible model that uses commercially available products.

We indicated by example how ARBAC97 can be used to enforce fairly sophisticated administrative policies. Nevertheless, configuring the details of ARBAC97 is a challenging task. So is analysing the consequences of a given configuration. Further research in these areas is needed to enable easy use of such administrative models.

REFERENCES

BERTINO, E., FERRARI, E., AND ATLURI, V. 1999. The specification and enforcement of authorization constraints in workflow management systems. *ACM Trans. Inf. Syst. Secur. 2*, 1 (Feb.).

BERTINO, E., SAMARATI, P., AND JAJODIA, S. 1997. An extended authorization model for relational databases. *IEEE Trans. Knowl. Data Eng. 9*, 1 (Jan./Feb.), 85–101.

FERRAIOLO, D., BARKLEY, J., AND KUHN, R. 1999. A role based access control model and reference implementation within a corporate intranet. *ACM Trans. Inf. Syst. Secur. 2*, 1 (Feb.).

HARRISON, M., RUZZO, W., AND ULLMAN, J. 1976. Protection in operating systems. *Commun. ACM 19*, 8.

NYANCHAMA, M. AND OSBORN, S. 1996. Modeling mandatory access control in role-based security systems. In *Database Security IX: Status and Prospects*. Elsevier North-Holland, Inc., New York, NY, 129–144.

NYANCHAMA, M. AND OSBORN, S. 1999. The role graph model and conflict of interest. *ACM Trans. Inf. Syst. Secur. 2*, 1 (Feb.).

RAMASWAMY, C. AND SANDHU, R. 1998. Role-based access control features in commercial database management systems. In *Proceedings of the 21st NIST-NCSC National Conference on Information Systems Security* (Arlington, VA, Oct. 5-8). 503–511.

SANDHU, R. 1997. Rationale for the RBAC96 family of access control models. In *Proceedings of the 2nd ACM Workshop on Role-Based Access Control* (Fairfax, VA, Nov. 6-7). ACM Press, New York, NY.

SANDHU, R. 1997. Roles versus groups. In *Proceedings of the 2nd ACM Workshop on Role-Based Access Control* (Fairfax, VA, Nov. 6-7). ACM Press, New York, NY.

SANDHU, R. AND AHN, G.-J. 1998. Decentralized group hieraches in unix: An experiment and lessons learned. In *Proceedings of the 21st NIST-NCSC National Conference on Information Systems Security* (Arlington, VA, Oct. 5-8).

SANDHU, R. AND AHN, G.-J. 1998. Group hierarchies with decentralized user assignment in Windows NT. In *Proceedings of the International Association of Science and Technology Development Conference on Software Engineering* (IASTED, Las Vegas, NV, Oct.).

SANDHU, R. AND BHAMIDIPATI, V. 1997. The URA97 model for role-based administration of user-role assignment. In *Database Security XI: Status and Prospect*, T. Y. Lin and X. Qian, Eds. Elsevier North-Holland, Inc., Amsterdam, The Netherlands.

SANDHU, R. AND BHAMIDIPATI, V. 1998. An oracle implementation of the pra97 model for permission-role assignment. In *Proceedings of the 3rd ACM Workshop on Role-Based Access Control* (RBAC, Fairfax, VA, Oct. 22-23). ACM Press, New York, NY, 13–21.

SANDHU, R. AND MUNAWER, Q. 1998. How to do discretionary access control using rules. In *Proceedings of the 3rd ACM Workshop on Role-Based Access Control* (RBAC, Fairfax, VA, Oct. 22-23). ACM Press, New York, NY, 47–54.

SANDHU, R. AND PARK, J. 1998. Decentralized user-role assignment for web-based intranets. In *Proceedings of the 3rd ACM Workshop on Role-Based Access Control* (RBAC, Fairfax, VA, Oct. 22-23). ACM Press, New York, NY, 1–12.

SANDHU, R. S. 1992. The typed access matrix model. In *Proceedings of the ACM Symposium on Research in Security and Privacy* (Oakland, CA, May). 122–136.

SANDHU, R. S. 1996. Role hierarchies and constraints for lattice-based access controls. In *Proceedings of the Fourth European Symposium on Research in Computer Security* (ESORICS96), E. Bertino, Ed. Springer-Verlag, New York, NY.

SANDHU, R. S. AND BHAMIDIPATI, V. 1999. Role-based administration of user-role assignment: The URA97 model and its Oracle implementation. *J. Comput. Secur. 1* (To appear).

SANDHU, R., COYNE, E. J., FEINSTEIN, H. L., AND YOUMAN, C. E. 1996. Role-based access control models. *IEEE Comput. 29*, 2 (Feb.), 38–47.

SANDHU, R. S. AND GANTA, S. 1995. On the minimality of testing for rights in transformation models. In *Proceedings of the IEEE Symposium on Research in Security and Privacy* (Oakland, CA, May). IEEE Computer Society Press, Los Alamitos, CA, 230–241.

Received: June 1998; revised: October 1998; accepted: December 1998