Proceedings of IFIP WG 11.3 Workshop on Database Security, Lake Tahoe, California, Aug 11-13, 1997

The URA97 Model for Role-Based User-Role Assignment

Ravi Sandhu and Venkata Bhamidipati

Laboratory for Information Security Technology
ISSE Department, Mail Stop 4A4
George Mason University, Fairfax, VA 22033, USA
`sandhu@isse.gmu.edu`, `www.list.gmu.edu`

Abstract In role-based access control (RBAC) permissions are associated with roles, and users are made members of appropriate roles thereby acquiring the roles' permissions. The principal motivation behind RBAC is to simplify administration. An appealing possibility is to use RBAC itself to manage RBAC, to further provide administrative convenience. In this paper we introduce a role-based administrative model, called URA97 (user-role assignment '97), for assignment of users to roles.

1 INTRODUCTION

Role-based access control (RBAC) has recently received considerable attention (see, for example, [FK92, Gui95, HDT95, NO95, SCFY96, RBAC95]). In RBAC permissions 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.

In large enterprise-wide systems the number of roles can be in the hundreds and users in the tens of thousands. Managing these roles and users, and their interrelationships is a formidable task that often is highly centralized and delegated to a small team of security administrators. It is natural to ask how RBAC itself can be used to manage RBAC.

RBAC administration encompasses 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. Thus a banking application can be implemented so 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. Role-role relationships establish broad policy. Control of these relationships is typically centralized in the hands of a few security administrators.

In this paper we focus exclusively on user-role assignment. A comprehensive administrative model for RBAC must account for all three issues mentioned above, among others. However, user-role assignment is a particularly critical administrative activity. It is likely to be the first administrative function that is decentralized and delegated to users rather than system administrators. Assigning people to tasks is a normal managerial function. Assigning users to roles should be a natural part of assigning users to tasks. A user-role assignment model can also be used for managing user-group assignment and therefore has applicability beyond RBAC.

In this paper we introduce a model for the assignment of users to roles by means of administrative roles and permissions. We call our model URA97 (user-role assignment '97). URA97 imposes strict limits on individual administrators regarding which users can be assigned to which roles. URA97 is defined in context of the family of RBAC96 family of models due to Sandhu et al [SCFY96]. However, it applies to almost any RBAC model, because user-role assignment is a basic administrative feature which will be required in any RBAC model.

The next section reviews the RBAC96 family of models followed by the definition of URA97 in section 3. URA97 has been implemented in the Oracle DBMS. Space limitations preclude description of the implementation here. Section 4 concludes the paper.

2 THE RBAC96 MODELS

A general family of RBAC models called RBAC96 was defined by Sandhu et al [SCFY96]. 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 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.

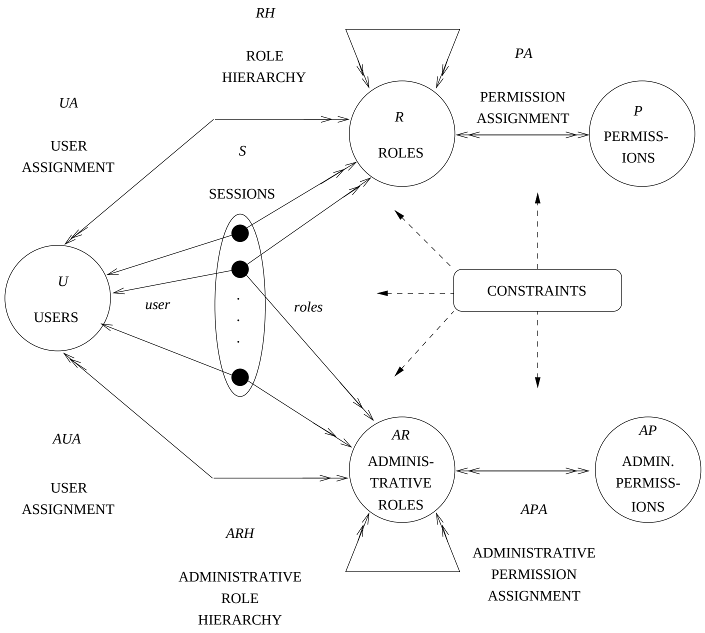
Motivation and discussion about various design decisions made in developing this family of models is given in [SCFY96, San97]. It is worth emphasizing that RBAC96 distinguishes roles and permissions from administrative roles and permissions respectively, where the latter are used to manage the former. 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.

3 THE URA97 ADMINISTRATIVE MODEL

RBAC has many components as described in the previous section. 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 would be quite complex and difficult to develop in a single step.

Fortunately administration of RBAC can be partitioned into several areas for which administrative models can be separately and independently developed to be later integrated. In particular we can separate the issues of assigning users to roles, assigning permissions to roles and defining the role hierarchy. In many cases, these activities would be best done by different administrators. Assigning permissions to roles is typically the province of application administrators. Thus a banking application can be implemented so 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. Design of the role hierarchy relates to design of the organizational structure and is the function of a chief security officer under guidance of a chief information officer.

In this paper our focus is exclusively on user-role assignment. As discussed in section 1 this is likely to be the first and most widely decentralized administrative task in RBAC. In the RBAC96 framework of figure 1 control of UA is vested in the administrative roles AR. For simplicity we limit our scope to assignment of users to regular roles. Assignment of users to administrative roles



- 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

Figure 1: Summary of the RBAC96 Model

is centralized under the chief security officer. In general the chief security officer has complete control over all aspects of RBAC96.

In the rest of this section we develop a model called URA97 in which RBAC is used to manage user-role assignment. We define URA97 in two steps dealing with granting a user membership in a role and revoking a user's membership. URA97 is deliberately designed to have a very narrow scope. For example creation of users and roles is outside its scope. 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.

In the simplest case user-role assignment can be completely centralized in a single chief security officer role. This is readily implemented in 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 (so long as this does not introduce a cycle). All this is consistent with classical discretionary thinking whereby member 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 let CR denotes all possible prerequisite conditions that can be formed using the roles in R. $\square$

In the trivial case a prerequisite condition can be a tautology. The simplest non-trivial case of a prerequisite condition is test for membership in a single role, in which situation that single role is called a prerequisite role.

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* $\subseteq AR \times CR \times 2^R$. $\square$

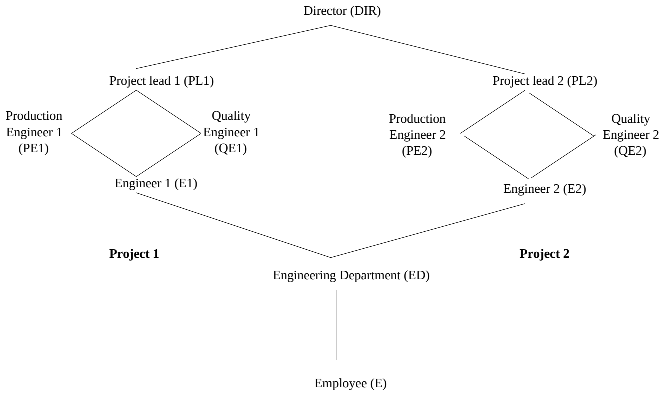


Figure 2: An Example Role Hierarchy

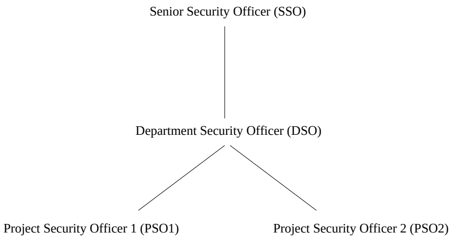


Figure 3: An Example Administrative Role Hierarchy

The meaning of $\textit{can-assign}(x, y, \{a, b, c\})$ 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 non-membership, in regular roles satisfies the prerequisite condition y to be a member of regular roles a, b or c.[1]

To appreciate the motivation behind the *can-assign* relation consider the role hierarchy of figure 2 and the administrative role hierarchy of figure 3. Figure 2 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-most role ED and senior-most role DIR. In between there are roles for two projects within the department, project

[1]User-role assignment is subject to constraints, such as mutually exclusive roles or maximum cardinality, that may be imposed. The assignment will succeed if and only if it is authorized by *can-assign* and it satisfies all relevant constraints.

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 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.

Figure 3 shows the administrative role hierarchy which co-exists with figure 2. 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 sake of illustration we define the *can-assign* relation shown in table 1 looking for the moment at the role set column. This example has the simplest prerequisite condition of 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 will be 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 of policy decision and our model provides the necessary flexibility.

In general, one would expect that the role being assigned is senior to the role previously required of the user. That is, if we have $can\text{-}assign(a, b, C)$ then b is junior to all roles $c \in C$. We believe this will usually be 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.

The notation of table 1 has benefited from the administrative role hierarchy. Thus for the DSO we have specified the role set as $\{$PL1, PL2$\}$ and the other

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]

Table 1: *can-assign* with Prerequisite Roles

values are inherited from PSO1 and PSO2. Similarly for the SSO. Nevertheless explicit enumeration of the role set is unwieldy, particularly if we were to 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 1.

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

This is a reasonable change to require 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 1 to include PL3.

Consider instead the range notation illustrated in table 1 in the role range column. The role set and role range columns of table 1 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 notation below

$$\begin{array}{lcllcl} [x,y] & = & \{r \in R \mid x \geq r \wedge r \geq y\} & [x,y) & = & \{r \in R \mid x \geq r \wedge r > y\} \\ (x,y] & = & \{r \in R \mid x > r \wedge r \geq y\} & (x,y) & = & \{r \in R \mid x > r \wedge r > y\} \end{array}$$

□

This notation is resilient to modifications in the role hierarchy such as addition of a third project which requires addition of the following row to table 1.

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

No other change is required since the [ED, DIR) range specified for the DSO will automatically pick up PL3.

The 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. Standard techniques for ensuring referential integrity would need to be applied when modifying the range hierarchy. 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 1. 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, in 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 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 of table 1 are modified respectively to the role sets {E1} and {E2}, 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 as discussed above. Conversely, the DSO role range authorization in table 1 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 with the SSO role in table 1. 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 2. The authorizations for PSO1 and PSO2 have been changed relative to table 1.

Let us consider the PSO1 tuples (analysis for PSO2 is exactly similar). The first tuple authorizes PSO1 to assign users with prerequisite role ED into E1.

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]

Table 2: *can-assign* with Prerequisite Conditions

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 a relative one 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.

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 on 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. A related issue is that of cascading revokes. 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, because 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.

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 interest of the organization. Granting of 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 non-memberships) satisfying the prerequisite condition. Moreover, there are some organizational circumstances which 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 being 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 $x' > x$, $(U, x') \in UA$. □

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

> [**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

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

Table 3: Example of *can-revoke*

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

Table 4: Example of Strong Revocation

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 that are active in a session. Strong revocation is theoretically equivalent to a series of weak revocations but it is a useful and convenient operation for administrators.

Let us consider the example of *can-revoke* shown in table 3 and interpret it in context of the hierarchies of figures 2 and 3. Let Alice be a member of PSO1, and let this be the only administrative role she has. Alice is authorized to strongly revoke membership of users from roles E1, PE1 and QE1. Table 4 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 still would 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 $y \geq x$ and Bob is a member of y. Weak revoke Bob from all such y as if Alice did this weak revoke. If any of the weak revokes fail then Alice's strong revoke has no effect otherwise all weak revokes succeed.

An alternate approach would be 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 so long as it 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 a downward cascading effect that also occurs. 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 of *can-assign* in table 1 and *can-revoke* in table 3 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.

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 non-membership 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.

4 CONCLUSION

In this paper we have developed the URA97 model for assigning users to roles and revoking users from roles. URA97 is defined in context of the RBAC96 model [SCFY96]. However, it should apply to almost any RBAC model, because user-role assignment is a basic administrative feature which will be required in any RBAC model. Authorization to assign and revoke users to and from roles is controlled by administrative roles. The model requires users must previously satisfy a designated prerequisite condition (stated in terms of membership and non-membership in roles) before they can be enrolled via URA97 into additional roles. URA97 applies only to regular roles. Control of membership in administrative roles remains entirely in hands of the chief security officer. We have identified strong and weak revocation operations in URA97 and have defined their precise meaning.

Acknowledgment This work is partially supported by the National Science Foundation, the National Security Agency and the National Institute of Standards and Technology.

References

[FK92] D. Ferraiolo and R Kuhn. Role-based access controls. In *Proc. 15th NIST-NCSC National Computer Security Conf.*, pp. 554–563, '92.

[Gui95] Luigi Guiri. A new model for role-based access control. In *Proc. of 11th Annual Computer Security Application Conference*, pages 249–255, New Orleans, LA, December 11-15 1995.

[HDT95] M.-Y. Hu, S.A. Demurjian, and T.C. Ting. User-role based security in the ADAM object-oriented design and analyses environment. In *Database Security VIII: Status and Prospects*. North-Holland, 1995.

[NO95] Matunda Nyanchama and Sylvia Osborn. Access rights administration in role-based security systems. In *Database Security VIII: Status and Prospects*. North-Holland, 1995.

[RBAC95] *Proc. of the 1st ACM Workshop on Role-Based Access Control, Nov 31-Dec. 1, 1995.* ACM, 1997.

[San97] Ravi Sandhu. Rationale for the RBAC96 family of access control models. In [RBAC95].

[SCFY96] Ravi Sandhu et al. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.