# ADMINISTRATIVE MODELS FOR ROLE-BASED ACCESS CONTROL

by

Qamar Munawer

A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
the Requirements for the Degree
of
Doctor of Philosophy
Information Technology

Committee:

# Administrative Models for Role-Based Access Control

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University.

By

Qamar Munawer
B.S., University of the Punjab, Pakistan, August 1969
M.S. (Physics), University of the Punjab, Pakistan, August 1971
M.S. (Computer Science), George Mason University, Fairfax, VA, May 1993

Director: Dr. Ravi S. Sandhu, Professor
Information and Software Engineering

Spring Semester 2000
George Mason University
Fairfax, Virginia

# DEDICATION

First of all I dedicate this dissertation to Allah for giving me knowledge and strength to accomplish this greatly desired dream of my life. Secondly, I dedicate this dissertation to my late parents Abdul Sattar Rajput and Aziz-ul-Nisa for bringing me up to the level that I fulfilled their dream but they could not see me accomplishing it. Finally, I dedicate this dissertation to my family for their support during these years with preserverance for the accomplishment of my greatly desired dream and goal of completing my PhD.

# ACKNOWLEDGMENTS

I would like to express my sincere gratitude and appreciation to my PhD advisor Professor Ravi Sandhu, for all his valuable guidance, teaching and encouragements during the realization of this research. It is Professor Ravi Sandhu's love for his students and dedication to research that made him available whenever we needed during all these years. I would like to let him know that if it was not for him I would not be getting my PhD.

I sincerely thank Professor Larry Kerschberg to be the chairman of my advisory committee and for his valuable guidelines and suggestions during research. Thanks also goes to Professor Xiaoyang (Sean) Wang for his general guidance and comments. I thank Professor Richard Carver for his teaching, motivation and guidance in my early years at George Mason.

I thank to my wife Nusrat, son Masud and daughters Rabia and Sobia for their love and affection. Thanks also goes to my friends and students both at George Mason and abroad for their friendship, valuable advice and encouragements that made my stay a memorable one.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# ABSTRACT

**THE ADMINISTRATIVE MODELS FOR ROLE-BASED ACCESS CONTROL**

**Qamar Munawer, Ph.D.**

**George Mason University, 2000**

**Dissertation Director: Dr. Ravi S. Sandhu**

Role-Based Access Control (RBAC) is a flexible and policy-neutral access control technology. For large systems, with hundreds of roles, thousands of users and millions of permissions, managing roles, users, permissions and their interrelationships is a formidable task that cannot realistically be centralized in a small team of security administrators. An appealing possibility is to use RBAC itself to facilitate decentralized administration of RBAC.

In this thesis, we first complete the Administrative RBAC model (ARBAC), that started as ARBAC97 introduced by Sandhu and Bhamidipati, by formally defining the component for role-role administration (RRA97). RRA97 is very different from user-role assignments (URA97) and permission-role assignments (PRA97) components of ARBAC97. RRA97 provides a framework for decentralized administration of role hierarchies. The desire is to give administrative roles autonomy within a range but only so far as the side effects of the resulting actions are acceptable. We introduce the concept of authority range for administrative roles so as to give them autonomous authority with respect to insertion and deletion of roles within the range and modification to their hierarchical relationships. In order to maintain overall global integrity

of the role hierarchy, we need to temper this outonomy by disallowing some operations authorized by the authority range. RRA97 provides formal definition and motivation for these restrictions.

We then formally define the notion of mobile and immobile users and permissions in context of RBAC. Mobile users are members of a role and thereby can use the permissions associated with the role. Moreover their membership in the role qualifies them to be put into other roles by appropriate administrators. Whereas immobile users can use the permissions associated with the role but this membership does not qualify them to be put into other roles. We define extensions in URA97 and PRA97 to accommodate this concept.

Finally we demonstrate that RBAC is sufficiently powerful to simulate Discretionary Access Control (DAC). Simulation of Mandatory Access Control (MAC) has already been demonstrated by Nyanchama, Osborn and Sandhu. This raises an important question as to whether or not it can accommodate the Access Control Models based on propagation of rights such as Harrison, Ruzzo and Ullman's model (HRU), Typed Access Matrix Model (TAM) or Augmented Types Access Matrix Model (ATAM). Amongest these ATAM is the most general model. We show that RBAC can easily accommodate ATAM.

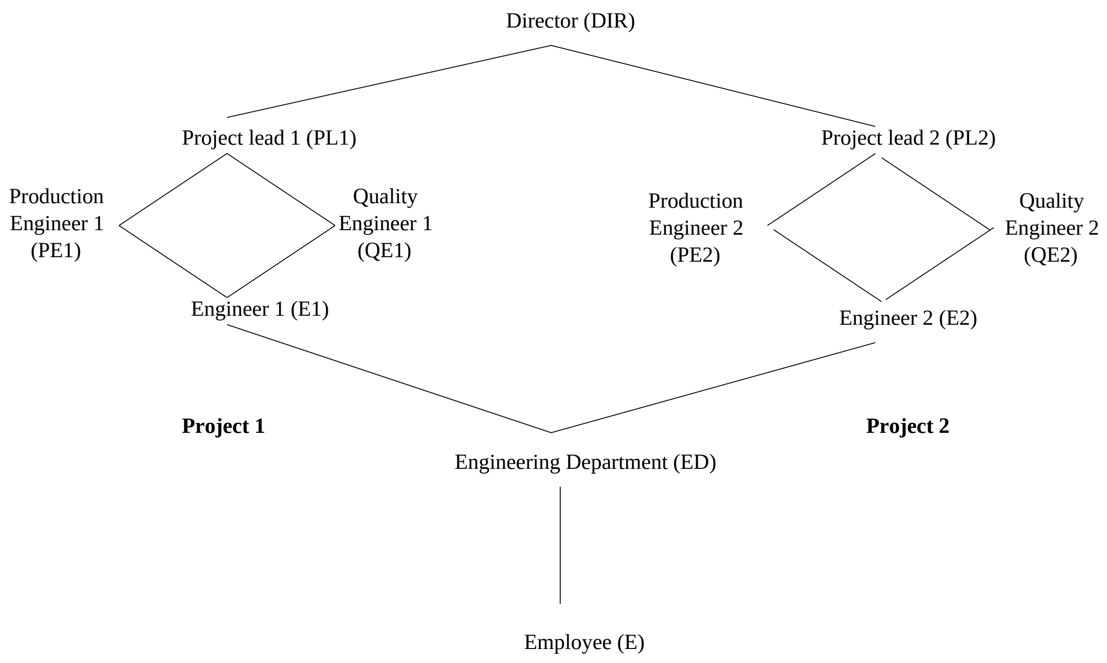
# Chapter 1

# MOTIVATION AND PROBLEM STATEMENT

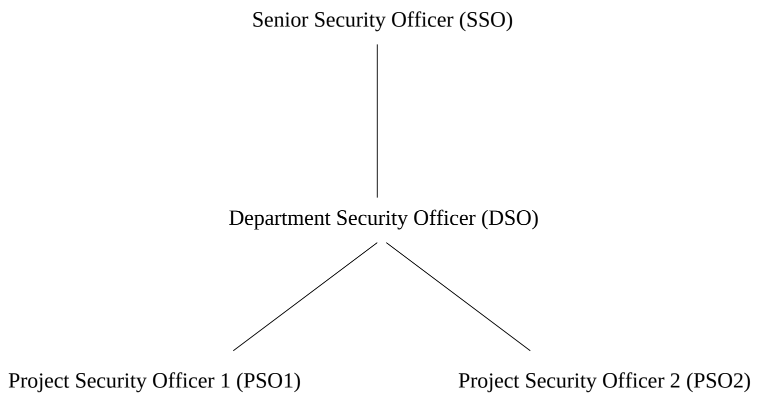
The need for controlled sharing of information and other resources among multiple users has led to the development of access control models [AELO90, Lam71, McL94, MMN90, San92b, FK92, Mur93, SS94]. Access control models provide a formalism and framework for specifying, analyzing and implementing security policies in multi-user systems. The models are desired to be flexible enough to easily accommodate diverse security policies. Numerous access control models have been published. Some are defined in terms of well known abstractions of subjects, objects and access rights (see for example [LM82, HRU76, San88, San89] and some in terms of roles, permissions and users (See for example [SCFY96, San97]). The important issues concerned with access control models are flexibility, policy neutrality, and simplicity of administration. The administrative RBAC model (ARBAC) defined by Sandhu et. al [SBC$^+$97] has these features, but it is not complete. The components for user-role assignments and permission-role assignments are defined as URA97 and PRA97. The component for role-role assignments is not developed. In role-role assignments we want to give flexibility to the administrative roles to do administration of roles (such as changing role-role relationships or addition/deletion of roles) under their authority but we do not want these actions to interfere with the authority of other administrative roles. The following example illustrates the need for this flexibility in access control models.

Consider the example of Role-Based Access Control (RBAC) where roles are created according to the job functions in the organization. Permissions are associated with roles and users are made members according to their qualifications and experience, thereby acquiring the associated permissions. Role hierarchies can be created where senior roles inherit permissions from junior roles. The user-role, permission-role and role-role assignments are controlled by administrative roles. One such example is that of an engineering department with the regular role hierarchy shown in figure 1.1(a) and administrative role hierarchy of figure 1.1(b). There is a junior most role E and every employee of the organization is a member of this role. Within the engineering department there is junior-most role ED and senior-most role DIR. Within the department there are two projects, project 1 and project 2. Each project has a senior-most project lead (PL1 and PL2) and 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). This structure can be extended to dozens or hundreds of projects within the engineering department. Moreover each project could have different structure of its roles.

The administrative hierarchy (figure 1.1(b)) coexists with the regular role hierarchy (figure 1.1(a)). There is a senior security officer role (SSO). The simplicity of administration requires the decentralization of responsibilities. Therefore our interest is in the administrative roles that are junior to SSO. There are two project security officers (PSO1 and PSO2) and a department security officer (DSO) role. The relationship of these roles is shown in figure 1.1(b). PSO1 and PSO2 have partial responsibilities over project 1 and project 2 respectively. The authority of PSO1 and PSO2 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 the alteration of role-role relationships by adding and deleting edges. For example, we might authorize



(a) Roles


Senior Security Officer (SSO)
Department Security Officer (DSO)
Project Security Officer 1 (PSO1)
Project Security Officer 2 (PSO2)

(b) Administrative Roles

Figure 1.1: Example Role and Administrative Role Hierarchies

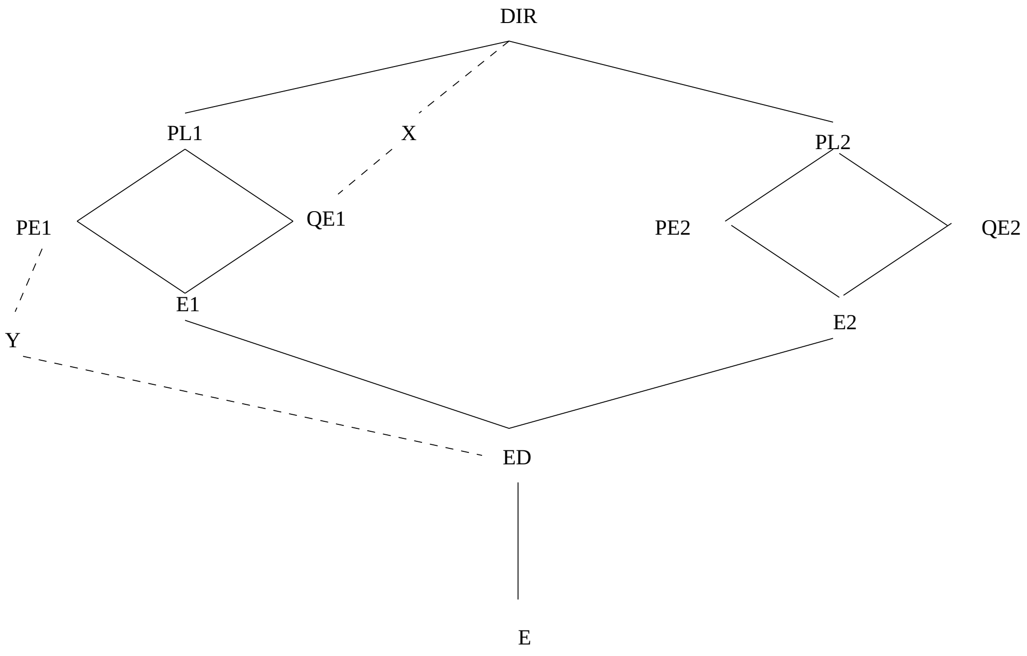


Figure 1.2: Out of Range Impact

the DSO to configure changes in the role hierarchy between DIR and ED. The PSO1 would manage the hierarchy between PL1 and E1, whereas PSO2 would manage the parts between PL2 and E2.

Consider the example of figure 1.2 that is identical to figure 1.1(a) except for additional edges shown in dashed lines which also brings in additional roles. Now if PSO1, who has authority over range between PL1 and E1, introduces an edge to make PE1 junior to QE1 the effect is to indirectly make Y junior to X. The PSO1 does not have authority to create this relationship, so this is an anomalous side effect. We should either restrict the authority of administrative role to create X and Y in the first place, or should prevent PSO1 from introducing the edge that makes PE1 junior to QE1. The administrative roles should be given autonomy within a range but only so far as the global side effects are acceptable. Access control models are needed to be flexible and policy neutral to accommodate the diverse policies of the organization but at the same time should not complicate administration due to such side effects.

In this thesis we pursue this goal by formulating new decentralized administrative models for RBAC, and by showing how existing decentralized administrative models (notably ATAM [AS92b]) can be simulated in RBAC.

## 1.1 Brief History of Access Control Models

In 1971 Lampson [Lam71] proposed the Access Matrix model based on a particular set of rules to control the propagation of access rights. The rules give the owner of the object complete discretion regarding the rights to the object. In 1972 Graham and Denning [GD72] proposed various rules by which the discretionary ability of the owner could be granted to other subjects. Although they proposed several set of rules, it was evident that additional alternatives could be proposed and it was not feasible to enumerate all policies.

These issues lead Harrison, Ruzzo and Ullman to develop the so called HRU model [HRU76]. This model does not incorporate a fixed set of rules for propagation of access rights. Instead a language is provided for a security policy designer to specify appropriate rules to the desired policy enforcement. The model could easily accommodate a rich variety of policies. It had good expressive power but extremely weak safety properties (that is, the determination of whether or not given subject can ever acquire access to an object). Safety was undecidable for most of the policies of practical interest, even for the monotonic case that does not allow deletion of access rights. Lipton and Snyder developed the take-grant model [LS77] in which they deliberately designed limited expressive power to eliminate the negative safety of HRU. This model was analyzed by many authors [LM82, Bis88]. The gap between the expressive power of take-grant and HRU was filled by Sandhu's Schematic Protection Model (SPM) [San88]. SPM has strong safety properties but has less expressive power than monotonic HRU [ALS92]. It allows a single parent creation operation. Extended Schematic Protection Model (ESPM) proposed by Amman and Sandhu [AS92a] allows multiple parents of a child. ESPM has expressive power that is equivalent to monotonic HRU and does not sacrifice the safety of SPM [San92a]. SPM and ESPM are both monotonic and therefore do not have the expressive power of HRU.

The positive safety results of SPM and ESPM and the expressive power of HRU is accommodated in Sandhu's Typed Access Matrix Model (TAM) [San92b] and Amman and Sandhu's Augmented Typed Access Matrix Model (ATAM) [AS92b]. TAM is defined by introducing strong typing into HRU. Strong typing means that each subject and object is created of a particular type that cannot be changed thereafter. ATAM is same as TAM but has the ability to check for the absence of access rights [SG93, SS92]. ATAM is considered as the current state of art with respect to formal models for generalized access control [HRU76, San88, AS92b, San92b].

The other paradigm of access control models is Role Based Access Control (RBAC) which originated with multi-user and multi-application on-line systems pioneered in 1970s. The central notion of RBAC is that permissions are associated with roles (the building block of RBAC) and users are assigned to appropriate roles thereby acquiring permissions. This greatly simplifies the management of permissions. Roles are created according to the job functions in an organization and users are assigned roles based on their qualifications, experience or responsibilities. Users can be easily reassigned from one role to another. Roles can be granted new permissions as new applications and systems are incorporated. Permissions can be revoked from roles as needed. Role hierarchies can be created according to the organizational setup and can be changed thereafter.

There are different aspects of RBAC that are dealt with in different ways in different systems. Sandhu et al [SCFY96, San98] introduced the well-known RBAC96 family which provides the general framework within which variations of RBAC can be accommodated. RBAC is policy neutral. It provides support for several important security principles like least privilege, privilege abstraction and separation of duties, but does not specify how to achieve these goals. The precise policy enforcement in RBAC is a consequence of detailed configuration of various components of RBAC, such as role hierarchies or role-role assignment (RRA), constraints, and administration of user-role assignments (URA) and permission-role assignments (PRA). RBAC can enforce traditional mandatory access control (MAC) [NO96, San96]. The formalization and implementation of RBAC is currently underway [GI96, SB97].

## 1.2 Problem Statement

RBAC has received considerable attention as a promising alternative to traditional discretionary and mandatory access control. RBAC is policy neutral and flexible. The policy that is enforced is a consequence of the detailed configuration of various components of RBAC. The flexibility of RBAC allows a wide range of policies to be implemented. The administration of RBAC is very important and must be carefully controlled to ensure the policy does not drift away from its original objectives. For large system, managing roles, users, permissions and their interrelationships is a formidable task that cannot be centralized in a small team of security officers. Decentralizing the details of RBAC administration without loosing central control over broad policy is a challenging goal for system designers. The administrative RBAC model (ARBAC97) defined by Sandhu et al. [SBC$^+$97] provides a significant advance towards this goal.

Three components are identified in ARBAC97. URA97 and PRA97 are defined whereas the component for role-role assignments (RRA97) is not developed [SBC$^+$97]. In this thesis, we would like to formulate highly decentralized administrative model for role-role relationships thereby completing the ARBAC97 model.

URA97 is concerned with user-role assignments. There are two consequences of assigning a user to a role. Firstly the user is authorized to use the permissions associated with that role and its juniors. Secondly, the user becomes eligible for assignment to other roles by appropriate administrative roles. These two aspects of role membership are tightly coupled in URA97. Consider the example of a visitor. Assignment of a visitor to a role should allow the visitor to use role's permissions but this membership should not be used by administrators to assign the visitor to other roles. (We consider visitor an immobile user.) Therefore, there is a need to decouple these two aspects. The distinction between mobile and immobile users can be very useful in practice. The concept of mobile and immobile users is needed to be get formally defined and incorporated in ARBAC. We would like to see how this concept affects user-role assignments and permission-role assignments.

RBAC is a promising alternative to traditional discretionary and mandatory access control. It has been shown that MAC can be accommodated in RBAC. Is DAC within the purview of RBAC? In this thesis we would be looking for an answer to this question. We would like to simulate DAC using roles to show the flexibility of RBAC. RBAC can accommodate both classical form of access control. This raises an important question as to whether or not it can accommodate other generalized access control models.

## 1.3 Summary of Contributions

1. The first contribution in this thesis is to complete the ARBAC97 model [SBC+97] by formally defining the RRA97 component to control role-role assignments. The effect of role-role assignments is to construct a role hierarchy in which senior roles inherit permissions from junior roles. Modifications to the role

hierarchy can have drastic impact on the effective distribution of permissions to roles. Formally introducing the concept of authority range and its encapsulation, we are effectively able to decentralize the administration of role-role assignments. For example, this makes it possible for a project security officer to rearrange roles within a project without impacting other role relationships within the department.

2. The second contribution of this thesis is that we have formally defined the concept of mobile users and permissions and accommodated it in administrative models for RBAC. The URA99 and PRA99 components are extensions to URA97 and PRA97 developed as a result of incorporation of mobile and immobile users and permissions in ARBAC.

3. Our final contribution in this thesis is that we provide a formal way of doing discretionary Access Control (DAC) in RBAC. We define several variations of DAC and demonstrate that RBAC is sufficiently powerful to encompass them. We have also shown the flexibility of RBAC to accommodate ATAM.

## 1.4 Organization of Thesis

Chapter 2 gives the brief background on Role Based Access Control Models, in particular the User-Role Assignment (URA97), Permission-Role Assignment (PRA97) Models, and Augmented Typed Access Matrix (ATAM) Model. Chapter 3 provides a formal definition of role-based administration of role hierarchies. Chapter 4 describes the evolution of URA97 and PRA97 into URA99 and PRA99 respectively. Chapter 5 covers Discretionary Access Control in RBAC. Chapter 6 theoretically compares the ATAM model and RBAC. Finally, the contributions of this thesis are summarized and discussion on the future research directions given in chapter 7.

# Chapter 2

# AN OVERVIEW OF ADMINISTRATIVE MODELS FOR ACCESS CONTROL

In this chapter we give a brief background on access control models. In particular, we review the Administrative Role Based Access Control (ARBAC97) and its components URA97 and PRA97 developed for role based administration of User-Role and Permission-Role Assignments respectively. Augmented Typed Access Matrix Model (ATAM) that is recognized as the current state of art with respect to models of generalized access control policies is reviewed in the last section of this chapter.

The chapter is organization is as follows. Section 2.1 describes the RBAC96. Section 2.2 discusses ARBAC97 with its components URA97 described in section 2.2.1 whereas URA97 grant and URA97 revoke models are defined in sections 2.2.2 and 2.2.3 respectively. Section 2.2.4 defines model for Permission-Role Assignments (PRA97). Finally section 2.3 describes the Augmented Typed Access Matrix (ATAM) model.

## 2.1 Model for Role Based Access Control (RBAC96)

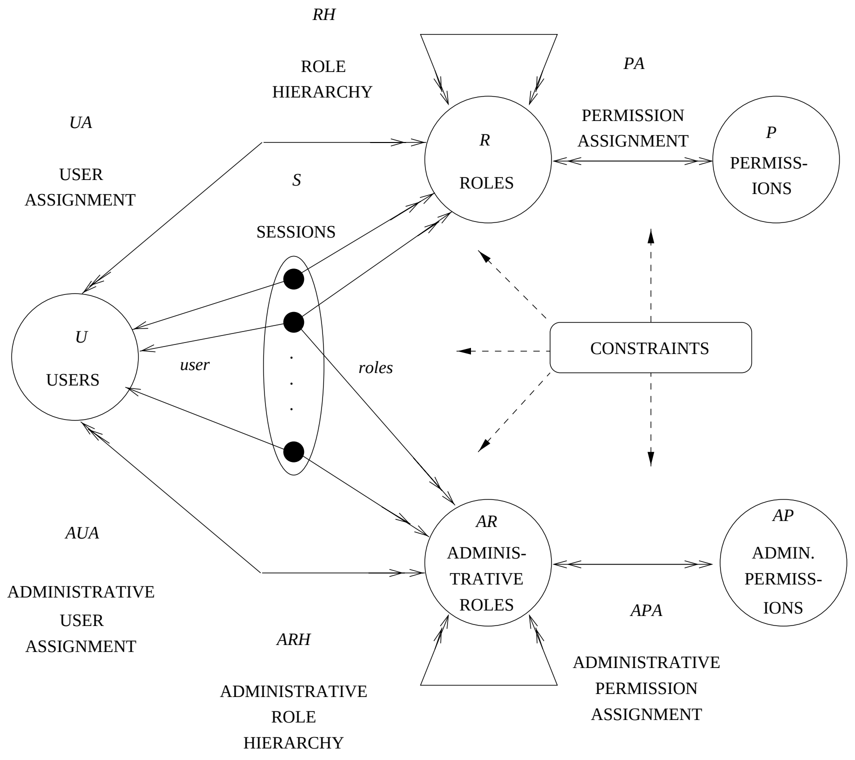
Role Based Access Control (RBAC) is considered as a promising alternative to traditional discretionary and mandatory access controls (MAC and DAC). In RBAC permissions are associated with the roles, and users are made members of appropriate roles thereby acquiring the roles' permissions. This greatly simplifies the management of permissions. Roles are created for various job functions in an organization and users are made members of the roles based on their responsibilities and qualifications. Users can easily be reassigned from one role to another. Roles can be granted new permissions as new applications or systems are incorporated. Permissions can be revoked from roles as needed. Role-role relationships can be established to layout broad policy objectives.

A general family of RBAC models called RBAC96 was defined by Sandhu et al [SCFY96, San97]. Figure 2.1 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 2.1 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, so that if $x > y$ then role $x$ inherits the permissions of role $y$, but not vice versa. In such cases, we say $x$ is senior to $y$. By obvious extension we write $x \geq y$ to mean $x > y$ or $x = y$.

Each session relates one user to possibly many roles. The idea is that a user establishes a session (e.g., by signing on to the system) and activates some subset of roles that he or she is a member of.



- $U$, 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 of roles and administrative roles $roles(s_i) \subseteq \{r \mid (\exists r' \geq r)[(user(s_i), r') \in UA \cup AUA]\}$ (which can change with time)

  session $s_i$ has the 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 2.1: Summary of the RBAC96 Model

## 2.2 Model for Administration of RBAC (ARBAC97)

In a large system the number of roles can be in hundreds or thousands, and users can be in tens or hundreds of thousands. Managing these roles and users, and their relationships is a formidable task that often is highly centralized in a small team of security administrators. An appealing possibility is to use RBAC itself to facilitate decentralized administration of RBAC. Sandhu et al [SBC$^+$97] has introduced a model for role-based administration of RBAC. This model is called Administrative RBAC97 (ARBAC97). It consists of three components. URA97 and PRA97 for the administration of user-role and permission-role assignments are described in detail [SB97, SB98]. The model is not complete because the role-role assignments (RRA97) are not formulated, Although some of the requirements and intuitive goals are described for RRA97 there are many open issues. The following sections describe URA97 and PRA97 components of ARBAC97.

### 2.2.1 URA97: A model for User-Role Assignments

In this section we define the URA97 model for the management of user-role assignments. For small systems user-role assignment can be controlled by a single system security officer role. This simple approach does not scale to large systems that require decentralization of user-role assignments to some degree.

In classical discretionary thinking an administrative role can be made owner of some regular roles and thereby given authority to add and delete users and permissions to these roles. This owner-based approach allows administrative roles to do whatever they want with the roles under their control. In URA97 we impose restrictions on which users can be added to a role by whom, as well as to clearly separate the ability to add and remove users from other operations on the role. The notion of prerequisite condition is introduced in URA97 as a key concept.

**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 $R$. □

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

User-role membership is controlled in two different ways, namely the granting, URA97 Grant Model and revoking, URA97 Revoke Model.

### 2.2.2 URA97 Grant Model

URA97 authorizes user-role assignments by the following relation.

**Definition 2** In URA97 the user-role assignments are controlled by the relation:

$$\textit{can-assign} \subseteq AR \times CR \times 2^R$$

The meaning of can-assign$(x, y, \{a, b, c\})$ is that a member of administrative role $x$ (or a member of administrative role that is senior to $x$) can assign a user whose current membership, or non membership in regular roles satisfy the prerequisite condition $y$ to be a member of regular roles $a, b$ or $c$.

Consider the role hierarchy of figure 1.1(a) and administrative role hierarchy of figure 1.1(b) showing the regular roles in engineering department and administrative role hierarchy which coexists with figure 1.1(a). The can-assign relation is defined

Table 2.1: Example of *can-assign* with Prerequisite Roles

| Administrative Role | Prerequisite Role | Role Set |
|---|---|---|
| PSO1 | ED | {E1, PE1, QE1} |
| PSO2 | ED | {E2, PE2, QE2} |
| DSO | ED | {PL1, PL2} |
| SSO | E | {ED} |
| SSO | ED | {DIR} |

(a) Subset Notation

| Administrative Role | Prerequisite Role | Role Range |
|---|---|---|
| PSO1 | ED | [E1, PL1) |
| PSO2 | ED | [E2, PL2) |
| DSO | ED | (ED, DIR) |
| SSO | E | [ED, ED] |
| SSO | ED | (ED, DIR] |

(b) Range Notation

in table 2.1(a) using role set and table 2.1(b) using range notation for the sake of illustration purposes.

Role sets are specified in URA97 and in this thesis by the following range notation.

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

Each tuple in this example has a simplest prerequisite condition of testing the membership in a single role known as the prerequisite role. The PSO1 role has partial responsibility over project 1 roles. Members of PSO1 can assign members of role ED to any of the E1, PE1, and QE1 roles, but not to PL1. Hence members of PSO1 have authority to enroll users to E1, PE1 and QE1 provided they are already members of ED. PSO2 has similar authority with respect to project 2. DSO inherits authority of PSO1 and PSO2 roles but can further add users who are members of ED to PL1 and

Table 2.2: Example of *can-assign* with Prerequisite Conditions

| Administrative Role | Prerequisite 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] |

PL2 roles. SSO can add users who are members of ED role to DIR role.

The *can-assign* relationship can use prerequisite condition as illustrated in table 2.2. The first tuple authorizes PSO1 to assign users with prerequisite role ED into E1. The second tuple authorizes PSO1 to assign users with prerequisite condition ED $\wedge$ $\overline{\text{QE1}}$ to PE1. Similar explanation is given for third tuple. Taken together the second and third tuples authorize PSO1 to put a user who is a member of ED into one not both of PE1 and QE1. This is how the mutually exclusive roles can be enforced by URA97. PE1 and QE1 are mutually exclusive with respect of the powers of PSO1. This may not be true for DSO or SSO.

### 2.2.3 URA97 Revoke Model

In this section we define a model for the revocation of users from roles. The model is consistent with RBAC philosophy but deviates from traditional discretionary access control approaches. In RBAC users are made members of the roles because of the responsibilities or task assignments in the interest of organization. The membership is not based solely on the discretion of the grantor. For example, suppose Alice makes Bob a member of role X. In URA97 it happens because Alice is assigned administrative authority over X via can-assign relation. Now if Alice is removed from the administrative role that has authorized her to grant Bob the membership of X then there is no reason to remove Bob from X. In other words it is not necessary to undo her previous grants. Take another example, suppose Bob is granted membership in X by Charles and Alice. Now if Alice revokes Bob's membership from X, Bob should still continue to have membership because of getting it from Charles. There is an issue of cascaded revokes. Suppose Charles got grant from Alice then the membership of Bob should be revoked by the action of Alice. Or perhaps should not because Alice only revoked her direct grant. URA97 does not couple revocation to the grantor in this manner. The URA97 philosophy is that grant and revocation are done for organizational reasons, and therefore should be treated as independent operations. Following approach is used to incorporate URA97 philosophy.

**Definition 3** URA97 controls user-role revocations by means of the relation:

$$\textit{can-revoke} \subseteq AR \times 2^R$$

The meaning of can-revoke($x, \{a, b, c\}$) is that a member of administrative role $x$ (or a member of administrative role that is senior to $x$) can revoke membership of user in role $a$, $b$ or $c$.

Revocation in URA97 is said to be weak because it applies only to the roles that are directly revoked.

**Definition 4** A user u is said to be an explicit member of a role x if (u, x) $\in$ UA, and an implicit member of role x if for some x' $>$ x, (u, x') $\in$ UA. □

Table 2.3: Example of *can-revoke*

| Administrative Role | Role Range |
|---|---|
| PSO1 | [E1, PL1) |
| PSO2 | [E2, PL2) |
| DSO | (ED, DIR) |
| SSO | [ED, DIR] |

RBAC96 allows a user to be an implicit as well as an explicit member of a role. Weak revocation has impact only on the explicit membership of a user. The user may continue to be an implicit member due to membership in a senior role.

Weak revocation has impact only on explicit memberships and has a downward cascading impact. For example, let us consider the role hierarchy of figure 1.1(a) and administrative role hierarchy of figure 1.1(b) where Dave is an explicit member of PE1 and thereby an implicit member of E1. If Dave's membership is explicitly revoked from PE1 then the implicit membership in E1 is automatically revoked.

Strong revocation of user's membership in role x requires that its membership should be explicitly revoked from x as well as its explicit or implicit membership must also be revoked from all roles senior to x. The effect is upward cascading revocation. In URA97 it takes effect only if all implied revocations are within the revocation range of administrative roles that are active in that session. For example, let us consider an example of *can-revoke* relation of table 2.3 and the role hierarchy of figure 1.1(a) and administrative role hierarchy of figure 1.1 (b) where Dave is an explicit member of roles E1 and PL1 and Eve is an explicit member of E1 and DIR roles. The members of role DSO can strongly revoke Dave from E1 but cannot revoke Eve from E1. In order to strongly revoke Eve from E1 the administrator must have SSO role.

In the formal model strong revocation is defined to be exactly equivalent to a series of weak revocations. Thus, formally it is a redundant operation. Actual system can provide strong revocation for convenience. Implementations can be optimized to do strong revocation directly rather than as a series of weak revocations.

### 2.2.4 PRA97 Model for Permission-Role Assignment:

PRA97 is concerned with permission-role assignment and revocation. Roles bring users and permissions together and have similar character with respect to users or permissions. Therefore PRA97 is proposed to be the dual of URA97. The notion of prerequisite condition is identical to that in URA97 except the boolean expression is now evaluated for membership and non-membership of a permission in the role.

**Definition 5** In PRA97 the permission-role assignments and revocation is controlled by the following relations,

$$\textit{can-assignp} \subseteq AR \times CR \times 2^R$$

$$\textit{can-revokep} \subseteq AR \times 2^R$$

The meaning of the can-assignp$(x, y, Z)$ is that a member of administrative role $x$ (or a member of administrative role that is senior to $x$) can assign a permission whose current membership, or non membership in regular roles satisfy the prerequisite condition $y$ to regular roles in range $Z$. The meaning of can-revokep$(x, Y)$ is that a member of administrative role $x$ (or a member of an administrative role that is senior to $x$) can revoke membership of a permissions from any regular role $y \in Y$. For example, in table 2.4(a), the role DSO is authorized to take any permission assigned to DIR and make it available for roles PL1 and PL2 whereas PSO1 can assign permissions assigned to PL1 to either PE1 or QE1 but not to both. According

Table 2.4: Example of can-assignp and can-revokep

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

(a) can-assignp

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

(b) can-revokep

to the table 2.4(b), DSO is authorized to revoke permissions from any role between ED and DIR. PSO1 can revoke permissions from PE1 and QE1. Revocation in PRA97 is weak so permissions may still be inherited after revocation. Strong revocation of the permission cascades down the role hierarchy, in contrast to cascading up the user membership. Like URA97 strong revocation is formally defined in terms of weak revocation.

## 2.3 Augmented Typed Access Model (ATAM):

In this section we briefly review the Augmented Typed Access Matrix model (ATAM) [AS92b]. The access matrix model was first formalized by Harrison, Ruzzo and Ullman and called HRU [HRU76]. The model had broad expressive power, but weak safety property (i.e., the determination of whether or not a given subject can ever acquire access to a given object). In take-grant model Lipton and Snyder [LS77] have deliberately limited expressive power to eliminate negative safety of HRU. The gap between expressive power of HRU and take-grant model was filled by Sandhu's Schematic Protection Model (SPM) [San88]. SPM has strong safety but less expressive power than monotonic HRU. Extended Schematic Protection Model (ESPM) defined by Sandhu [AS92a] has expressive power that is equivalent to HRU and does not sacrifice safety of SPM. SPM and ESPM are both monotonic and are not as expressive as HRU. Sandhu [San92b] proposed TAM to incorporate the good safety results and at the same time have the general expressive power of HRU. The principal innovation of TAM is to introduce strong typing of subjects and objects into the access matrix model of HRU. Each subject or object is created of specific type, which thereafter cannot be changed. The types and rights are specified as part of the system definition and are not predefined in the model. This adds up some flexibility in term of the implementation of the security policy of the organization. The extension of TAM proposed by Sandhu is ATAM, which allows checking for the absence of rights in the commands. TAM and ATAM are equivalent in expressive power [SG93], however from practical point of view it is beneficial to allow testing for absence of rights.

ATAM represents the distribution of rights by the access matrix. The matrix has a row and column for each subject and a row for each object. Subjects are also considered as objects. The rights a subject X possess for object Y are entered in cell [X, Y] of the access matrix. The security officer specifies the following sets as the part of definition of the system:

1. Finite set of access rights denoted by $R_{right}$.

2. The finite set of object types T. There is a set of subject types $T_S, T_S \subseteq T$

For example T = {user, so, file} specifies there are three types, user, security officer, and file, with $T_S = \{\text{user, so}\}$. The set of rights is $R_{right} = \{r, w, o\}$ where r stands for read, w for write and o for owner.

The rights in the access matrix serve two purposes. First is the authorization of the subject to perform some operation on the object or to perform some operation that changes the access matrix. For example right o in [X, Y] authorizes X to change the matrix so that subject Z can read Y. The focus of ATAM is on the second purpose of rights i.e., the authorization by which access matrix gets changed. The changes are made by means of commands of following formats:

```
**Command** $\alpha(X_1 : t_1, X_2 : t_2, X_3 : t_3, ..., X_k : t_k)$
        begin

    $if\ r_1 \in [Xs_1, Xo_1] \wedge r_2 \in [Xs_2, Xo_2] \wedge ... \wedge r_k \in [Xs_m, Xo_m]\ and$

    $R_{k+1} \notin [Xs_1, Xo_1] \wedge r_{k+2} \notin [Xs_2, Xo_2] \wedge .... \wedge r_m \notin [Xs_m, Xo_m]$
        then
            $op_1; op_2; .....; op_n$
        end
```

Or

```
**Command** $\alpha(X_1 : t_1, X_2 : t_2, X_3 : t_3, ..., X_k : t_k)$
        begin
            $Op_1; op_2; .....; op_n$
        end
```

Here $\alpha$ is name of the command $X_1, X_2, X_3, ..., X_k$ are formal parameters whose types are $t_1, t_2, t_3, ..., t_k$ whereas $r_1, r_2, r_3, ..., r_n$ are rights and $s_1, s_2, ....s_m$, and $o_1, o_2, ..., o_m$

are integers between 1 and k. The predicate following the *if* part is called the condition and sequence of operations $op_1; op_2; .....; op_n$ is called the *body* of the command. The ATAM command is invoked by substituting actual parameters of the appropriate types for the formal parameters. The usual interpretation is that the ATAM command is initiated by the first subject in the parameters list. The condition part is evaluated with respect to its actual parameters. The body of the conditional command is executed only if the condition evaluates to be true. Each op1 is one of the primitive operations from

enter r into [X, Y]

create subject X of type $t_s$

create object O of type $t_o$

delete r from [X, Y]

destroy subject X

destroy object O

Enter operation enters a right $r \in R_{right}$ into an existing cell of access matrix. If the right is already present then the contents are not changed. The delete operation removes a right from the cell. The cell is treated as a set, thus there will not be any effect if the cell does not have the right.

The create subject operation introduces an empty row and column in the access matrix. It is required that the subject being created must have a unique identity. Operation destroy subject removes the row and column corresponding to the subject. The create object operation adds an empty row in the access matrix and destroy object remove the corresponding row.

The *protection state* of the ATAM system, is defined as a set {SUB, OBJ, t, AM} where

- SUB is a set of subjects.

- OBJ is a set of objects.
- (SUB is a subset of OBJ)
- t is a type function that maps every subject to a subject type and every object to an object type.
- AM is an access matrix, with a row for every subject in SUB and column for every object in OBJ.

The *protection state* is changed by means of ATAM commands. The security officer defines a finite set of ATAM commands when the system is specified.

# Chapter 3

# RRA97 - ADMINISTRATIVE MODEL FOR ROLE-ROLE RELATIONSHIPS

As mentioned in chapter 1, access control models are desired to be simple and flexible to accommodate diverse security policies. Role-Based Access Control (RBAC has received a lot of attention due to its flexibility and simplicity in administration. Centralized management of RBAC in large systems is a tedious and costly task. An appealing possibility is to use RBAC itself to facilitate decentralized administration of RBAC. The administrative RBAC model (ARBAC97) identifies three components, URA, PRA and RRA for administration of user-role, permission-role and role-role assignments respectively. URA97 and PRA97 are already defined in literature [SB96, SBC$^+$97, SBM99].

In this chapter we formally define RRA97, a model for the decentralized administration of role-role relationships. The effect of role-role assignments is to create a role hierarchy in which senior roles inherit permissions from junior roles. Modifications to a role hierarchy can have drastic impact on the effective distribution of permissions. RRA97 formally defined in this chapter allows administrative roles to modify role-role relationships in a portion of role hierarchy without impacting other role-role relationships.

The organization of chapter is as follows. Section 3.1 discusses different kinds of roles distinguished in ARBAC97 and their assignments. In section 3.2, we formally define

RRA97, model for decentralized administration of role-role relationships, thereby completing the Administrative model for RBAC (that is, ARBAC97).

## 3.1 Kinds of Roles in RBAC

For role-role assignment we distinguish three kinds of roles, roughly speaking 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 to distinguish these three kinds of roles is that different administrative models apply to establishing relationships between 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 because 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 these 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 6** Ability-role assignment and revocation are respectively authorized in ARA97 by *can-assigna* $\subseteq AR \times CR \times 2^A$ and *can-revokea* $\subseteq AR \times 2^A$. $\square$

**Definition 7** Group-role assignment and revocation are respectively authorized in GRA97 by *can-assigng* $\subseteq AR \times CR \times 2^G$ and *can-revokeg* $\subseteq AR \times 2^G$. $\square$

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. We use the term role to mean UP-roles in the rest of the thesis.

## 3.2 RRA97 - Model for Role-Role Assignments

This section formally defines the ARBAC97 component for the administration of role hierarchies.

Decentralization of administrative authority requires that members of different administrative roles should 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 the alteration of role-role relationships by adding or deleting edges. For example in figure 1.1 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 PSO2 would manage the part between PL2 and E2. This leads to the following notion of a can-modify relation.

**Definition 8** Role creation, role deletion, edge insertion and edge deletion are all authorized by the relation, *can-modify* $\subseteq AR \times 2^{UPR}$ (with subsets of $R$ identified by the range notation but limited to open ranges that do not include the endpoints). □

Table 3.1 illustrates an example of can-modify relative to the hierarchies of figure 1.1. 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

Table 3.1: Example of can-modify

| Administrative Role | UP-Role Range |
|---|---|
| DSO | (ED, DIR) |
| PSO1 | (E1, PL1) |
| PSO1 | (E2, PL2) |

the range Y and can modify relationships between roles in a range Y. The examples in the rest of the chapter are all in context of figure 1.1 and table 3.1. For purpose of our example we have ignored PSO2 in this table and have 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. 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 will leave dangling references in table 3.1, 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 well 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 will 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 [SBC$^+$97] 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.

### 3.2.1 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 so as 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 1.2. 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 makes PE1 junior to QE1 (and indirectly Y junior to X).

## 3.3 Restriction on Authority of Administrative Roles

In general administrative roles are given autonomy within a range but only so far as global side effects are acceptable. They should be restricted so as not to have unacceptable impact on authority of other administrative roles.

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.

### 3.3.1 Concept of Range

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

**Definition 9** A range of roles is defined by giving lower bound $x$ and upper bound $y$, where $y > x$. Formally $(x, y) = \{z : \mathit{Roles} \mid x < z < y\}$. We say $x$ and $y$ are the end points of the range. □

Note that a range, as defined here, does not include the end points. In figure 1.1, (E1, PL1), (E2, PL2) and (ED, DIR) are different ranges. The range (ED, DIR) contains the roles which constitute ranges (E1, PL1) and (E2, PL2). We say ranges (E1, PL1) and (E2, PL2) are junior to range (ED, DIR).

**Definition 10** 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 to be junior or senior to itself. This makes later definitions more simple.

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 are as follows.

**Definition 11** 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.

### 3.3.2 Authority Range and Encapsulated Authority Range

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

**Definition 12** 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 13** In RRA97 authority ranges do not partially overlap. □

Note that an administrative role may have more than one authority range. Table 3.1 shows that DSO has authority over the range (ED, DIR). In figure 1.1 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 3.1 shows that PSO1 has authority over two incomparable authority ranges namely (E1, PL1) and (E2, PL2).

Let us consider figure 1.2 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 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 14** 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 $r2 < r1 \Leftrightarrow r2 < x$. □

Intuitively an encapsulated range is one in which all roles have identical relation 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. Figure 3.1 and 3.2 respectively show examples of encapsulated and non-encapsulated range $(x, y)$.

The addition/deletion of a role or an edge in hierarchy may change authority range from encapsulated to non-encapsulated one. Therefore we put restriction on creation/deletion of roles and edges.

## 3.4 Restriction on 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. 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 1.2. To prevent this from happening we introduce the following notions.

**Definition 15** 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 16** 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 3.3. 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

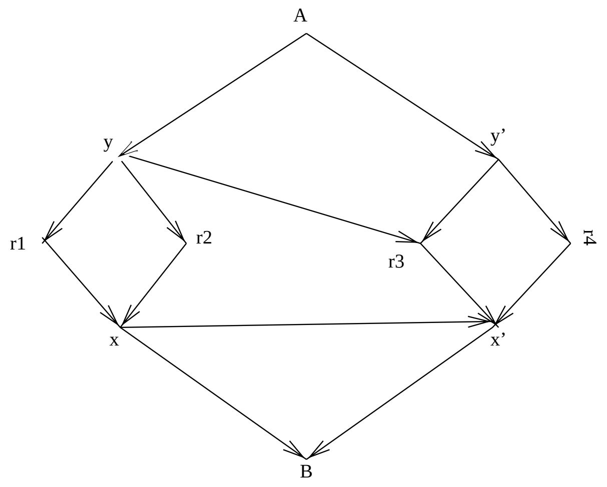


Figure 3.1: Encapsulated Range (x, y)

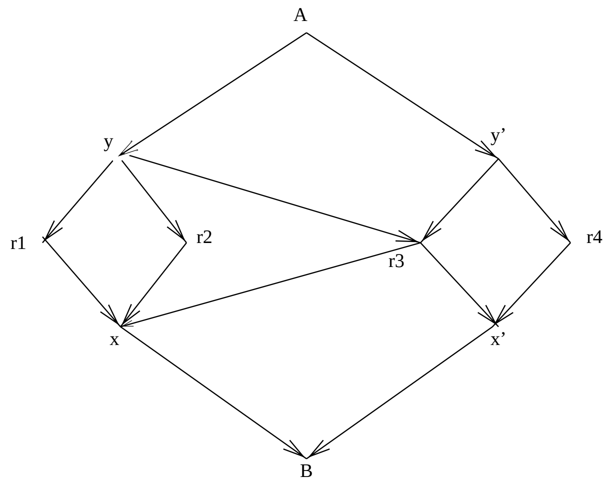


Figure 3.2: Non-Encapsulated Range (x, y)

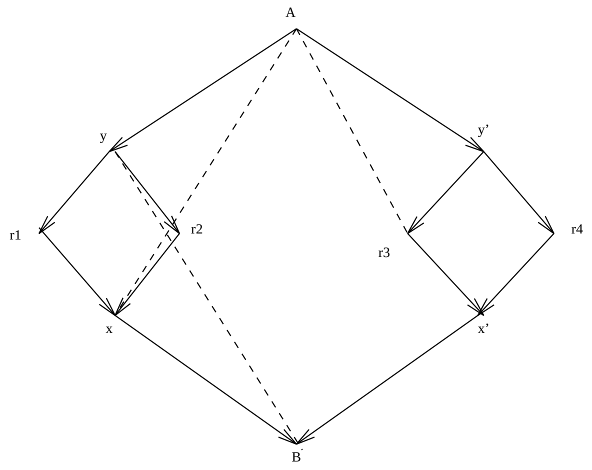


Figure 3.3: Create Range

(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.

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 still subject to constraints of RBAC96. In some cases we may not have such a powerful administrator, but it is a reasonable working assumption for our purpose.

## 3.5 Restriction on Role Deletion

Deletion of roles in a hierarchy is a complicated process. Our assumption is that role in the 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 non-existing roles. The ranges with these deleted end points will 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. Although this is a fairly restrictive constraint, it is required to keep the range referential integrity intact.
2. Roles referred in 1 above can be made inactive (explained in the next paragraph) whenever it is needed to delete them. The advantage of deactivating roles is that it avoids references to non-existing roles and at the same time achieves the purpose of deletion.

A role is said to be inactive if a user assigned to it cannot activate it in a session. The edges to and from the inactive role, its associated permissions 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 alternations. 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 for deletion of roles.

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

2. Delete 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.

## 3.6 Restriction on Edge Insertion

Now let us explain how the model deals with the insertion of edges in the role to role relationships. The 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 3.2 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) would not be allowed. This leads to the following formal definition for insertion of an edge.

**Definition 17** The new edge $AB$ between incomparable roles $A$ and $B$ can be inserted only 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 $(x, y)$. $\square$

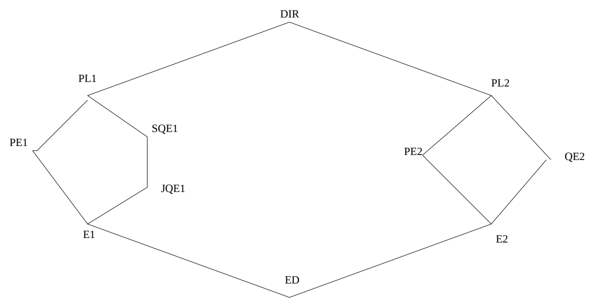


Figure 3.4: Before Deletion of edge from SQE1 to JQE1

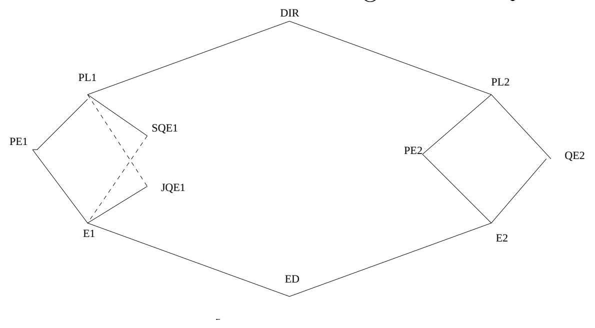


Figure 3.5: After Edge Deletion

## 3.7 Restriction on Edge Deletion

The deletion of a transitive edge 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 the transitive reduction then it is not a candidate for deletion.[1] For example in figure 3.4 deletion of the edge SQE1 to JQE1 will change the hierarchy. Edge deletion only applies to a single edge and does not carry over to implied transitive edges. As discussed in the general rules for edge deletion RRA97 keeps intact transitive edges after deletion. 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 3.5.

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.

## 3.8 Summary

In this chapter we have formally defined the model for decentralized administration of role-role relationships thus completing the definition of ARBAC97 that started in [SBC$^+$97]. In RRA97 [SM98b] the concept of authority and encapsulated range is introduced for the first time. This concept is used to restrict the administrative roles to do the role-role assignments without affecting authority of other administrative roles. Restrictions are imposed on creation and deletion of roles as well as on insertion and deletion of edges so that the encapsulated authority range remains encapsulated. Administrative roles are given autonomy within their authority range as far as the

[1]Other models do not have this restriction. For example, Oracle allows insertion and deletion of transitive edges [KL95].

global side effects are acceptable.

# Chapter 4

# ARBAC99 - ADMINISTRATIVE MODELS FOR RBAC

As mentioned in chapter 2, administrative model for RBAC (ARBAC97) defined by Sandhu et. al [SBM99] consists of three components. URA97 is concerned with user-role assignments. PRA97 is concerned with permission-role assignments and is a dual of URA97. For example, every constraint on user-role relationships has a dual counter part with respect to permission-role relationships, and vice versa. RRA97 deals with role-role relationships. RRA97 is formally defined in chapter 3.

In this chapter we introduce the concept of mobile and immobile users and permissions[1] that was lacking in ARBAC97. To incorporate mobility in URA97, we provide motivation in section 4.1 and extensions that lead to new models for user-role assignments, URA99 grant model described in section 4.2 and URA99 revoke model in section 4.3. The lack of symmetry in *can-assign* and *can-revoke* relations in URA97 model is discussed in section 4.3. Motivation for introducing mobility of permissions and extensions in PRA97 to incorporate this concept is described in section 4.4. Can URA97 accommodate mobility? The answer to this question is provided in section 4.5.

[1]The concept of mobile and immobile users and permissions is introduced for the first time in [SM98a]

## 4.1 Motivation for enhancements:

The URA97 model defined by Sandhu et. al [SB96] to decentralized the administration of user-role assignments is described in chapter 2. The assignment of users to roles is controlled by the can-assign relation. The consequence of assigning a user to a role is that the user can use the permissions associated with that role and that administrative roles can use this membership to assign this user to other roles. We want to de-couple these tightly coupled aspects of role memberships. Following examples are provided as a motivation for de-coupling these aspects of user-role membership.

1. Visitor: Assignment of a visitor to a role should allow the visitor to use the permissions of the role but this membership should not be used to assign the visitor to other roles.

2. Trainee: A user under training can be assigned to the ED role of the engineering department. Being a member of this role the user can participate in the engineering department while junior administrators can be prevented from assigning this user to any project. However after completion of training user's membership in ED role can be used to assign him to other roles.

3. Consultant. A consultant assigned to E2 role of hierarchy figure 1 is required to participate in project 2 and use the general resources of engineering department due to inherited membership in ED role. At the same time we want to prevent junior administrators to use this membership to assign the consultant to project 1.

## 4.2 URA99 Grant Model

In this section we present a formal model for User-Role Assignments keeping in view the issues described in section 4.1. URA99, a new model thus developed is an enhancement in URA97. The basic intuition of URA97, that is, the decentralization of administration of user role assignments, permission role assignments and role-role assignments is not changed. URA99 builds upon the URA97 model by introducing the concept of mobile and immobile memberships. A user's membership in a role can be mobile or immobile. Mobile membership of user u in a role x means that u can use permissions of role x and members of an administrative role can use this membership to put user u in other roles. Immobile membership of user u in a role x means that u can use permissions of role x but members of an administrative role cannot use this membership to put user u in other roles.

To formalize this distinction we consider each role x to consist of two sub-roles Mx and IMx. Membership in Mx is mobile, whereas in IMx is immobile. For compatibility with URA97 we define a set of roles R to consist of the mobile and immobile sub-roles defined as follows.

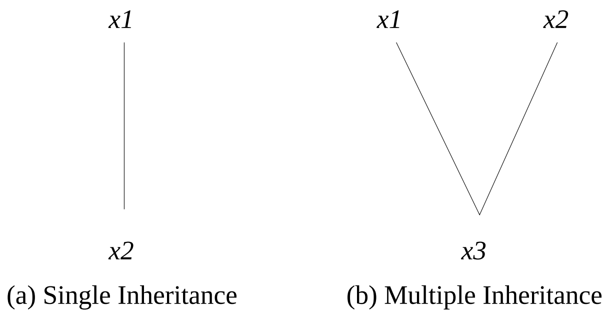
**Definition 18** For a given set of roles $R1$ we define the roles for URA99 to be

$$R = \{Mx,\ IMx \mid x \in R1\}$$

□

The definition for user assignment relation of URA97, UA $\in U \times R$ essentially remains unchanged in URA99. Assignment of user to $Mx$ signifies that the user is a mobile member of $x$. Similarly, assignment of user to $IMx$ signifies that the user is an immobile member of $x$. Combined with the previously defined notion of explicit and implicit memberships we distinguish four kind of role memberships in URA99.

**Definition 19** There are four kinds of user-role memberships in URA99 for any given role $x$.



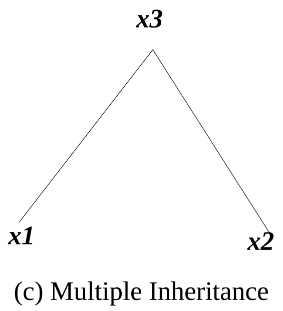


Figure 4.1: Inheritance of mobility and immobility

- *Explicit Mobile Member EMx*

  $u \in EMx \equiv (u, Mx) \in UA$

- *Explicit Immobile Member IMx*

  $u \in IMx \equiv (u, IMx) \in UA$

- *Implicit Mobile Member IMx*

  $u \in ImMx \equiv (\exists x' > x)(u, Mx') \in UA$

- *Implicit Immobile Member IMx*

  $u \in ImIMx \equiv (\exists x' > x)(u, IMx') \in UA$ $\square$

It is possible for a user to have more than one kind of membership in a role. However, we will define the semantics of URA99 so that there is strict precedence amongst these four kinds of memberships as follows:

$$EMx > IEMx > ImMx > ImIMx$$

Though a user has more than one kind of membership in a role, at any time only one of them is in effect. For example consider the hierarchy of figure 4.1(a) where role $x1$ is senior to role $x2$. A user who is an explicit mobile member of $x1$ is an implicit mobile member of $x2$. Similar inheritance implies to immobile memberships as well. Next consider hierarchy of figure 4.1(b). Let a user be an explicit mobile member of $x1$ and explicit immobile in $x2$. Now this user is implicit mobile (because of membership in $x1$) as well as implicit immobile in x3 (because of membership in $x2$). According to the precedence rule the implicit mobile is stronger than implicit immobile membership so the user will effectively have implicit mobile membership in role x3. Finally consider figure 4.1(c) and a user who is explicit mobile member of x3 and explicit immobile member of role $x2$. Now the user is implicit mobile membership (because of membership in x3) and explicit immobile member in role $x2$. According to our precedence rule the effective membership of user in role $x2$ will be immobile, whereas the user will have mobile membership in role $x1$.

The prerequisite condition for URA99 in context of mobile and immobile memberships is different from URA97. It is also different for grant and revoke models. URA99 grant model requires membership to a check for mobile membership whereas non-membership means absence of any kind of membership. The prerequisite condition for URA99 grant model is formally defined as follows.

**Definition 20** In URA99 grant model a prerequisite condition is evaluated for user u by interpreting $x$ to be true if

$$u \in EMx \vee (u \in ImMx \wedge u \notin EIMx)$$

Table 4.1: Example of *can-assign-M* with Prerequisite Conditions

| Administrative Role | Prerequisite Condition | Role Range |
|---|---|---|
| PSO1 | ED | [E1, PL1) |
| PSO2 | ED | [E2, PL2) |
| DSO | ED $\wedge$ $\overline{\text{PL2}}$ | [PL1, PL1] |
| DSO | PE1 $\wedge$ $\overline{\text{PL1}}$ | [PL2, PL2] |
| SSO | ED | (ED, DIR] |
| SSO | E | [ED, ED] |

Table 4.2: Example of *can-assign-IM* with Prerequisite Conditions

| Administrative Role | Prerequisite Condition | Role Range |
|---|---|---|
| PSO1 | ED | [E1, PL1) |
| PSO2 | ED | [E2, PL2) |
| DSO | ED $\wedge$ $\overline{\text{PL2}}$ | [PL1, PL1] |
| DSO | PE1 $\wedge$ $\overline{\text{PL1}}$ | [PL2, PL2] |
| SSO | ED | (ED, DIR] |
| SSO | E | [ED, ED] |
| DSO | E | [ED, ED] |

and $\overline{x}$ to be true if

$$u \notin Emx \wedge u \notin EIMx \wedge u \notin ImMx \wedge u \notin ImIMx$$

Note that it is not possible for $x$ and $\overline{x}$ to be simultaneously true. However they can be simultaneously false ($u \in EIMx$ and $u \notin EMx$).

The user role assignments are controlled by two relations as follows.

**Definition 21** User-role assignments as mobile members are authorized by the relation, *can-assign-M* $\subseteq AR \times CR \times 2R$ and user-role assignments as immobile members are authorized by the relation, *can-assign-IM* $\subseteq AR \times CR \times 2R$. $\square$

The meaning of *can-assign-M*$(x, y, Z)$ is that a member of administrative role $x$ (or a member of administrative role senior to $x$) can assign a user whose current membership satisfies the prerequisite condition $y$ to a regular role $z \in Z$ as a mobile member. Whereas the meaning of *can-assign-IM*$(x, y, Z)$ is that a member of administrative role $x$ (or a member of administrative role senior to $x$) can assign a user whose current membership satisfies the prerequisite condition $y$ to a regular role $z \in Z$ as a immobile member. Examples of *can-assign-M* and *can-assign-IM* are respectively shown in table 4.1 and table 4.2. The top six rows of table 4.2 are identical to table 4.1. This means mobile or immobile membership is granted at discretion of the individual administrator. URA99 requires that authorizations for granting mobile and immobile memberships be explicitly expressed in this manner. There is no implication in general that authority to grant mobile implies authority to grant immobile memberships (although this may be a common case). The last row in table 4.2 authorizes DSO to enroll any employee as an immobile member of ED. DSO does not have power to enroll employee as mobile member of ED. That power is confined to SSO. In this example DSO can enroll an employee as immobile member of ED and later the SSO can upgrade the membership to be mobile.

## 4.3 URA99 Revoke Model

In URA97 the *can-assign* relation involves prerequisite condition whereas *can-revoke* does not. URA99 revoke model fixes this lack of symmetry, which is quite independent of the issue of mobility. The revoke model also deals with the revocation of mobile and immobile memberships.

Table 4.3: Example of *can-revoke-M* with Prerequisite Conditions

| Administrative Role | Prerequisite Condition | Role Range |
|---|---|---|
| PSO1 | E | [E1, PL1) |
| PSO2 | E | [E2, PL2) |
| DSO | E | (ED, DIR) |
| SSO | E | [ED, DIR] |
| PSO1 | E1 | [E2, PL2) |
| PSO2 | E2 | [E1, PL1) |

### 4.3.1 Motivation for fixing lack of symmetry

Consider the example of engineering department where PSO1 controls the user-role assignments in project 1 roles. If user Bob is member of role E1 then PSO1 can assign him to any role of project 1. Suppose PSO1 does not want Bob to member of any role outside project 1 because that may affect Bob's performance in project 1. If Bob is assigned to any role outside project 1 then PSO1 should have authority to revoke him from there. The lack of prerequisite condition in URA97 does not provide this conditional authority to revoke a user from a role.

In URA99 revoke model we introduce two relations to authorize revocation of mobile and immobile memberships as follows.

**Definition 22** The URA99 revoke model authorizes revocation of mobile memberships by the relation *can-revoke-M*$(x, y, Z)$ and revocation of immobile membership by the relation *can-revoke-IM*$(x, y, Z)$ □

The meaning of meaning of *can-revoke-M*$(x, y, Z)$ is that a member of administrative role $x$ (or a member of administrative role senior to $x$) can revoke a user whose current membership satisfies the prerequisite condition $y$ to a regular role $z \in Z$ as a mobile member. Similarly for *can-revoke-IM* with respect to immobile memberships.

Table 4.4: Example of *can-revoke-IM* with Prerequisite Conditions

| Administrative Role | Prerequisite Condition | Role Range |
|---|---|---|
| PSO1 | E | [E1, PL1) |
| PSO2 | E | [E2, PL2) |
| DSO | E | (ED, DIR) |
| SSO | E | [ED, DIR] |
| PSO1 | E1 | [E2, PL2) |
| PSO2 | E2 | [E1, PL1) |
| DSO | E | [ED, ED] |

An example of these relations is shown in table 4.3 and table 4.4. The evaluation of prerequisite condition for URA99 revoke model is different from grant model. For revoke model we do not distinguish mobile and immobile memberships.

**Definition 23** In URA99 revoke model the prerequisite condition is evaluated for a user $u$ by interpreting $x$ to be true if

$$u \in EMx \vee u \in EIMx \vee u \in ImMx \vee u \in ImIMx$$

and $\overline{x}$ to be true if

$$u \notin EMx \wedge U \notin EIMx \wedge u \notin ImMx \wedge u \notin ImIMx$$

□

Note that unlike the grant model $x$ and $\overline{x}$ cannot be false at the same time. As in URA97 they are complements of each other.

If all membership is restricted to being mobile then URA99 is identical to URA97. This can be achieved by setting *can-assign-IM* and *can-revoke-IM* to be empty.

## 4.4 PRA99 - Model for permission-role assignments

In this section we formally define PRA99, model for permission role assignments. PRA99 model builds upon PRA97 by introducing the concept of mobile and immobile permissions.

### 4.4.1 Motivation for mobile and immobile permission

Consider figure 1.1, PRA97 allows us to give PSO1 authority to take a

permission assigned to PL1 and grant it to the roles in the range [E1, PL1}. The idea is that each project can delegate permissions of the project lead role to more junior project roles as the project security officer deems appropriate. While this may be acceptable for most project lead permissions, it is likely that some permissions are not suitable for such delegation. These permissions are to be assigned to PL1 as immobile.

### 4.4.2 Formal Definition of PRA99

The PRA99 model deals with assignments and revocation of permission from roles. In this section we define the formal model for permission role assignments. Like users, permission can also be assigned as mobile and immobile. As PRA97 relates to URA97, we have PRA99 as an exact dual of URA99. The main difference in PRA99 and URA99 is that in PRA99 implicit membership of permissions is inherited upward in the hierarchy. The definitions for PRA99 are as follows.

**Definition 24** The roles in PRA99 are the same as in URA99, that is,

$$R = \{Mx, IMx |\ x \in R1\}$$

. The permission role assignment relation is $PA \subset U \times R$. □

**Definition 25** There are four kinds of permission-role memberships in PRA99 for any given role $x$.

- *Explicit Mobile Member EMx*
  $p \in \mathit{EMx} \equiv (p, \mathit{Mx}) \in \mathit{PA}$
- *Explicit Immobile Member IMx*
  $p \in \mathit{IMx} \equiv (p, \mathit{IMx}) \in \mathit{PA}$
- *Implicit Mobile Member IMx*
  $p \in \mathit{ImMx} \equiv (\exists x' > x)(p, \mathit{Mx}') \in \mathit{PA}$
- *Implicit Immobile Member IMx*
  $p \in \mathit{ImIMx} \equiv (\exists x' > x)(p, \mathit{IMx}') \in \mathit{PA}$ $\square$

**Definition 26** In PRA99 grant model a prerequisite condition is evaluated for permission p by interpreting $x$ to be true if

$$p \in EMx \vee (p \in ImMx \wedge p \notin EIMx)$$

and $\overline{x}$ to be true if

$$p \notin EMx \wedge p \notin EIMx \wedge p \notin ImMx \wedge p \notin ImIMx$$

$\square$

The permission-role assignments are controlled by two relations as follows.

**Definition 27** Permission-role assignments as mobile members are authorized by the relation, *can-assignp-M* $\subseteq AR \times CR \times 2^R$ and User-role assignments as immobile members are authorized by the relation, *can-assignp-IM* $\subseteq AR \times CR \times 2^R$. $\square$

**Definition 28** The PRA99 revoke model authorizes revocation of mobile memberships by the relation *can-revokep-M*$(x, y, Z)$ and revocation of immobile membership by the relation *can-revokep-IM*$(x, y, Z)$. □

**Definition 29** In PRA99 revoke model the prerequisite condition is evaluated for a permission p by interpreting $x$ to be true if

$$p \in EMx \vee p \in EIMx \vee p \in ImMx \vee p \in ImIMx$$

and $\overline{x}$ to be true if

$$p \notin EMx \wedge p \notin EIMx \wedge p \notin ImMx \wedge p \notin ImIMx$$

□

## 4.5 Discussion

We have formally described the motivation and enhancements in components of ARBAC97 to accommodate mobility without loosing the basic intuition, that is, the decentralization of administration of user-role assignments, permission-role assignments and role-role hierarchies by means of administrative roles, prerequisite conditions and role ranges. The principal objective in defining modifications was simplicity of administrative models. The model with URA99, PRA99 and RRA97 components is given the name ARBAC99 [SM99]. ARBAC97 is a special case of ARBAC99. If all memberships are restricted to being mobile then URA99 is identical to URA97 and PRA99 is identical to PRA97. This can be achieved by simply setting *can-assign-IM* and *can-revoke-IM* to be empty.

## 4.6 Simulating mobility in URA97

The concept of mobility is missing from sub-models of ARBAC97. Users and permissions are mobile in URA97 and PRA97. Theoretically the possibility of immobile behavior of users and permissions is not ruled out from these sub-models. In this section we would like to investigate issues arising if mobility is introduced in ARBAC97. This will justify need for enhancements in URA97 and PRA97 that gives us the new sub-models for user-role assignments (URA99) and permission-role assignments (PRA99). ARBAC99 model consists of sub-models URA99, PRA99 and RRA97 that is left unchanged.

### 4.6.1 URA97 with fixed mobility

In URA97 all users are mobile. The simplest way of introducing the concept of immobile users is that SSO creates a special role IR, call it immobile role. The members of IR role cannot be assigned to any other role and that this role cannot be made part of any role hierarchy. Although not required but for simplicity we assume that only SSO is authorized to assign users to role IR. The members of IR cannot be assigned to other roles is a way of making them immobile users. It is imposed by modifying *can-assign* relationship. An example of *can-assign* relation is shown in table 4.5. The first tuple allows SSO to assign members of roles in range [ED, DIR] to role IR, thus making them immobile. In other tuples a check is performed as prerequisite condition before assigning a user to any other role that the user is not a member of IR.

To analysis the limitations to this method, let us consider the role hierarchy of the engineering department shown in figure 1(a). Let us say, we want user Bob to be a mobile member of the E1 role so that PSO1 can use this membership to put Bob in PE1 or QE1 roles and at the same time we want Bob to be immobile member

Table 4.5: Example of *can-assign* With mobility

| Administrative Role | Prerequisite Condition | Role Range |
|---|---|---|
| SSO | [ED, DIR] | [IR, IR] |
| PSO1 | ED $\wedge$ $\overline{\text{IR}}$ | [E1, PL1) |
| PSO2 | ED $\wedge$ $\overline{\text{IR}}$ | [E2, PL2) |
| DSO | ED $\wedge$ $\overline{\text{PL2}}$ $\wedge$ $\overline{\text{IR}}$ | [PL1, PL1] |
| DSO | PE1 $\wedge$ $\overline{\text{PL1}}$ $\wedge$ $\overline{\text{IR}}$ | [PL2, PL2] |
| SSO | ED $\wedge$ $\overline{\text{IR}}$ | (ED, DIR] |
| SSO | E $\wedge$ $\overline{\text{IR}}$ | [ED, ED] |

of ED role to make sure that he should not be put by administrative roles to other projects. The method does not enforce this flexibility with respect to memberships. Bob can have only one character in the system. He is either mobile or immobile. It is not possible to make Bob mobile in one range and immobile in the other roles. Let us consider another example, if Bob is a consultant (described in section 4.1) and an explicit member of role E2 thereby acquire implicit member ship to ED role. We cannot prevent other administrative roles from using this membership to assign him to project 1. Forcing restriction so that he should not be assigned to project 1 will enforce that he cannot be assigned to any other role of project 2. We term this mobility as fixed mobility. The impact of fixed mobility is global to the system.

### 4.6.2 URA97 with dynamic mobility

In this section we describe another approach that allows user to have mobile membership in certain roles and at the same time have immobile membership in others. URA97 does not distinguish these kinds of memberships. To introduce mobility in URA97 framework, we create two roles for each role. For example ED will be replaced with two roles MED and IMED representing mobile ED and immobile ED respectively. Members of MED are mobile whereas that of IMED are immobile. Similarly

E1 is replaced with ME1 and IME1 and so on. The administrative roles can assign membership to these roles. We do not restrict membership in these roles. A user can be explicit member of mobile role as well as immobile role at the same time but only one will be effective at a time. In case a user has both mobile as well as immobile membership it can be resolved using same precedence rules defined for URA99.

This method may be practical if there are no role hierarchies. The effective membership of a role is easy to find. However if there are role-role relationships, as in case of engineering department, then we have to maintain two separate hierarchies. One for mobile roles and other for immobile roles. Any change in one will require a similar change in the other hierarchy. For example, consider hierarchy of figure 1.1 (a), we will have two hierarchies one with MX roles and the other with IMX (where X is a regular role in figure 1.1, like ED and E1 etc.) roles with similar relationships. Now if PL1 wants to add an edge between MPE1 and MQE1, he has to add an edge between IMPE1 and IMQE1 as well. Similarly adding a role, deleting a role, inserting or deleting an edge, all these actions require maintenance of two separate hierarchies. That may become cumbersome if there are thousands of roles. Not only this, finding the membership will require tracing membership in different hierarchies before resolving any conflict.

The *can-assign* relation is modified to accommodate this concept. Some tuples of *can-assign* relation are shown in table 4.6. For simplicity, We assume that similar authority ranges in both hierarchies are controlled by same administrative roles. For example, PSO1 controls assignments in both [ME1, MPL1] and [IME1 and IMPL1] ranges. We call [ME1, MPL1] and [IME1 and IMPL1] ranges similar authority ranges in two hierarchies.

From the perspective of roles, users and permission have similar character. PRA97 is concerned with the permission-role assignments. PRA97 is a dual of URA97

Table 4.6: Example of *can-assign* for URA97 with dynamic mobility

| Administrative Role | Prerequisite Condition | Role Range |
|---|---|---|
| PSO1 | MED | [ME1, MPL1) $\vee$ [IME1, IMPL1) |
| DSO | MED $\wedge$ $\overline{\text{MPL2}}$ $\wedge$ $\overline{\text{IMPL2}}$ | [MPL1, MPL1] $\vee$ [IMPL1, IMPL1] |

with the exception that prerequisite condition is evaluated for membership and non-membership of permissions in roles and that in role hierarchies the membership is inherited upward. Dynamic mobility in PRA97 requires two roles for mobile and immobile permissions. Permissions associated with mobile roles can be assigned to other roles whereas permissions associated with immobile roles cannot be. We can modify *can-assignp* relation to accommodate mobility as we did for URA97. The major issue here is that the mobile and immobile users can use both mobile and immobile permission associated with the roles. Roles defined in URA97 to simulate mobility (such as ME1 and IME1) are not suitable for PRA97. The reason is that immobile permissions assigned to immobile role cannot be used by the mobile users because of non-membership in immobile role. For example, Bob is mobile in E1 therefore he is member of ME1 role not of IME1 role. We have to create separate roles for permission-role assignments and have to create a relationship between these roles similar to the one created for user-role assignments in URA97. For E1 regular role we will create MPE1 and IMPE1 roles for mobile and immobile permissions respectively with a constraint that no user can be explicitly assigned to these roles, that is these are abilities. Similarly we restrict that no permissions should be associated with ME1 and IME1 roles, that means these are groups. The members of groups can use permission if there is a relationship among groups and corresponding abilities. MPE1 and IMPE1 are corresponding abilities for groups ME1 and IME1. The required relationship among these roles is shown in figure 4.2. Overall conclusion is that we will

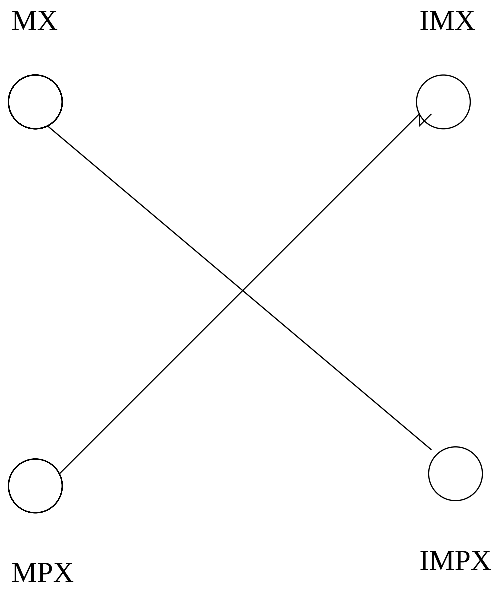


Figure 4.2: Relationship among groups and abilities for role X

have four role hierarchies, namely role hierarchy for mobile users, roles hierarchy for immobile users, role hierarchy for mobile permissions and role hierarchy for immobile permissions. We have not only to maintain same structure of these hierarchies but also to maintain relationship of corresponding roles.

Complicated *can-assign* and *can-revoke* relations and maintenance of role hierarchies with relationships between roles in different hierarchies leads to significant complication of administration and make this approach impractical.

In this chapter we have described motivation of introducing mobile and immobile character of users and permissions. The new models developed on this concept are URA99 and PRA99 that allows administrative roles to manage and change mobile and immobile behavior within their authority range. ARBAC97 is also considered to be enough flexible to incorporate mobility of users and permissions, but the components become so complicated that they loose practicality. We believe that there are benefits to have URA99 and PRA99 for administrative simplicity.

# Chapter 5

# DISCRETIONARY ACCESS CONTROL IN RBAC96

The principal contribution of this chapter is that it demonstrates that RBAC is flexible and expressive enough to simulate Discretionary Access Control (DAC). We first define the concept of an object in RBAC environment and the operations that are associated with creation and destruction of an object. MAC and DAC are two extrems of Access Control. It has already been shown in [NO96, San96] that MAC can be simulated using roles. It was assumed that RBAC is flexible enough that it can accommodate DAC as well, but there was no formal simulation. The formal simulation is described in this chapter. The results derived from this chapter has opened ways to think of accommodating the generalized DAC oriented Access Control Models like HRU and TAM in RBAC. The results of this chapter has theoretical importance.

The chapter is organized as: Section 5.1 defines DAC variations that are considered in this chapter for simulation. Section 5.2 discusses operations associated with creation of an object in RBAC and section 5.3 describes operations involved in destroying an object. Section 5.4 is gives simulation of DAC variation defined in section 5.1. Section 5.5 discusses revocation of access.

## 5.1 DAC variations

There is no well defined DAC in literature. The basic concept of DAC policy is that the originator or creator of the object retains the control over granting access to objects it owns. Owner of an object has discretionary authority over who else can access that object [SS94, SS97]. In other words DAC policy is owner-based administration of access rights. There are many variations of DAC policy, particularly concerning how the owner's discretionary power can be delegated to other users and how access is revoked. [Lam71, GD72]. In this section we define DAC variations that will be consider for simulation.

Our intention is to identify major mainstream variations of DAC and demonstrate their construction in RBAC. The constructions are such that it will be obvious how they can be extended to handle other related DAC variations. This is an intuitive, but well-founded, justification for the claim that DAC can be simulated in RBAC.

1. **Strict DAC** The first variation is a strict DAC policy. The control over the access to the object is in the hands of the owner. Suppose Tom has created an object (Tom is owner of the object) and grants read access to Bob. This DAC policy variation requires that Bob cannot propagate access to the object to any other user directly or indirectly. Tom has the right to take away any granted access from Bob.

2. **Liberal DAC**The liberal DAC policy is relaxed. We will be considering the following three variations of liberal DAC policy as:

    (a) **One Level Grant:**Owner allows access to users such that the users can grant access to the objects for which they have access to. So Tom being the owner of object O can grant access to Bob who can grant access to

Harry. But Harry cannot grant access to any other user. The users can get access either from owner or from a user to whom owner has granted right to grant access to other users.

   (b) **Two Level Grant:** Not only the owner but some of the users, on the owner's discretion, can grant access to users with a permission to further grant access. Bob can now may be allowed to grant access to Harry who can grant access to other users.

   (c) **Multilevel Grant** In two level grant variation Harry can grant read access to the users but not with a permission to grant it further. In multilevel grant variation the user Harry is permitted to grant access to users with a permission to grant it further.

   For revocation we will consider the cases where the revocation is independent of grantor and also when only the users who has granted access to an object can revoke the access from the user.

3. **DAC with Change of Ownership** The last variation we will consider is DAC policy where not only the right of granting access is propagated to other users as in (2) but the ownership can also be transferred. However we will restrict our self to objects of single ownership. Tom can allow other users, of his choice, to have access with a propagation of access right and as well as can transfer the ownership on the object it owns.

The DAC policies we consider, all share the following characteristics.

1. The creator of an object becomes its owner.

2. There is only one owner of an object.[1] In some cases ownership[ remains fixed with the original creator, whereas in other cases it can be transfered to another owner.

3. Destruction of an object can only be done by its owner.

There can be more variations of DAC policy but we are considering only the three mentioned above to make the conclusions that DAC policy can be simulated in RBAC. To specify above variations in RBAC, for simplicity, it is sufficient to consider DAC with one operation for convenience. We choose read operation on objects. Other operations like write or execute etc. can be explained on similar lines. The DAC policy only affects the $R$, $UA$, $AUA$, $PA\ and\ APA$ components of RBAC96. In the rest of the paper the values of these components before and after the execution of any of the above operations are represented as non-primed and primed values respectively. Thus R is set of roles before operation and R' after operation. Before we define a mechanism for simulation of DAC variations let us first describe the common operations which are specifically associated with the creation and destruction of object in RBAC96 system.

## 5.2 Creation of Object in RBAC

In RBAC96 the creation of an object is linked with the creation of four roles and six permissions defined as below.

**Definition 30** The creation of an object O in an RBAC96 system is associated with the creation of the following,

[1]This assumption is not critical to our constructions. It will be obvious how multiple owners can be handled. Assuming a single owner is convenient and simplifies our exposition.

1. Three administrative roles, $OWN\_O$, $PARENT\_O$ and

   $PARENTwithGRANT\_O$.

2. One regular role, $READ\_O$.

3. Eight permissions for object $O$, $addParent\_O, deleteParent\_O$,

   $destroyObjects\_O$, $addReadUser\_O$, $deleteReadUser\_O$,

   $addParentWithGrant\_O$, $deleteParentWithGrant\_O$.

   $andcanRead\_O$. □

The meaning of permission $addParent\_O$ is that if it is associated with a role then the members of that role can assign a user to role $PARENT\_O$ and the meaning of $addReadUser\_O$ is that the members of the role with which it is associates can assign users to role $READ\_O$. Similarly the meanings of other permissions can be understood from their names. We emphasis that these roles and permissions are associated with the object created and each object created will have a distinct set of such roles.

The permissions assigned to the roles at the time of creation of an object O are as follows:

1. $OWN\_O$ gets the permissions $destroyObject\_O$, $addParentWithGrant\_O$, $deleteParentWithGrant\_O$.

2. $PARENTwithGRANT\_O$ gets $addParent\_O$, $deleteParent\_O$.

3. $PARENT\_O$ is associated with $addReadUser\_O$, $deleteReadUser\_O$.

4. $READ\_O$ has associated permission $canRead\_O$.

5. The user who creates the object automatically becomes member of the roles thus created. □

In RBAC96 this behavior would be enforced by appropriate constraints. One constraint on the permissions would be that they are automatically associated with the roles at the time of their creation. Another constraint would be that the user who creates an object O, automatically becomes the member of all four roles. The result effect is that the owner have discretionary control over access to the newly created object.

**Definition 31** The creation of an object $O$ by a user $u$ results in the following changes in RBAC96 system. $\square$

1. $R' = R \cup \{READ\_O\}$
2. $AR' = AR \cup \{OWN\_O \cup PARENT\_O \cup PARENTwithGRANT\_O\}$
3. $ARH' = ARH \cup \{(OWN\_O, PARENTwithGRANT\_O) \cup$ $(PARENTwithGRANT\_O, PARENT\_O)\}$
4. $UA' = UA \ \cup \ \{(u, READ\_O)\}$
5. $AUA' = AUA \cup \{(u, OWN\_O) \cup (u, PARENT\_O)\}$
6. $PA' = PA \cup \{(canRead\_O, READ\_O)\}$
7. $APA' = APA \cup \{(addParentWithGrant\_O, OWN\_O),$
   $(deleteParentWithGrant\_O, OWN\_O),$
   $(destroyObject, OWN\_O),$
   $(addParent\_O, PARENTwithGRANT\_O),$
   $(deleteParent\_O, PARENTwithGRANT\_O),$
   $(addReadUser\_O, PARENT\_O),$
   $(deleteReadUser\_O, PARENT\_O)\}$

The members of role $OWN\_O$ manage the administrative roles $PARENT\_O$ and $PARENTwithGRANT\_O$ roles whereas the members of roles $PARENT\_O$ and $PARENTwithGRANT\_O$ roles manage the user role assignments for $READ\_O$ role. This is shown in figure 5.1. The arrow indicates which role can add/revoke membership of the role. For example, member of OWNER role can add/revoke a user from membership of roles $PARENT\_O$ and $PARENTwithGRANT\_O$ roles. The figure also shows seniority relationship between the three administrative roles, so $OWN\_O$ inherits all permissions of $PARENTwithGRANT\_O$ which in tern inherits permissions of $PARENT\_O$.

## 5.3 Destroy an Object in RBAC

Destroying an object $O$ requires deletion of four roles namely $OWN\_O$, $PARENT\_O$, $PARENTwithGRANT\_O$ and $READ\_O$ and six permissions, also removing the references to these roles in relations $UA$, $AUA$ and $APA$. This can be achieved by first finding the sub-relations of $UA$, $AUA$ and $APA$ that refer to the roles associated with the object $O$. Let us call them as $UA\_O$, $AUA\_O$ and $APA\_O$ and defined as:

**Definition 32** $UA\_O$ is defined as follows.

$$UA\_O = \{\forall(x, y) \in UA \ \mid \ y = READ\_O\}$$

$\square$

**Definition 33** $AUA\_O$ is defined as follows.

$$AUA\_O = \{\forall(x, y) \in AUA \mid y = OWN\_O \ \vee \\ y = PARENT\_O \ \vee \ y = PARENTwithGRANT\_O\} \quad \square$$

**Definition 34** $APA\_O$ is defined as follows.

$$APA\_O = \{\forall(x,y) \in APA \mid y = OWN\_O \ \vee$$
$$y = PARENT\_O \vee \ y = PARENTwithGRANT\_O\} \qquad \square$$

With these relations destroying an object O requires removal of all tuples referring to roles associated with that object. The following definition specifies the changes made when an object $O$ is destroyed.

**Definition 35** Destroying of the object $O$ owned by user $u$ means:

- $APA' = APA \Leftrightarrow APA\_O$
- $AUA' = AUA \Leftrightarrow AUA\_O$
- $UA' = UA \Leftrightarrow UA\_O$
- $AR' = AR \Leftrightarrow \{(PARENT\_O, OWN\_O, PARENTwithGRANT\_O)\}$
- $R' = R \Leftrightarrow \{READ\_O\}$

## 5.4 Simulation of Strict DAC

In strict DAC only the owner can grant/revoke read access from users. The creator is the owner of the object and being member of the roles $PARENT\_O$ and $PARENTwithGRANT\_O$ roles can change assignments of the role $READ\_O$. The constraint imposed is that the membership of administrative roles cannot change. Hence only owner can do the assignments in the $READ\_O$ role.

**Definition 36** The constraints in RBAC96 system for strict DAC are as follows.

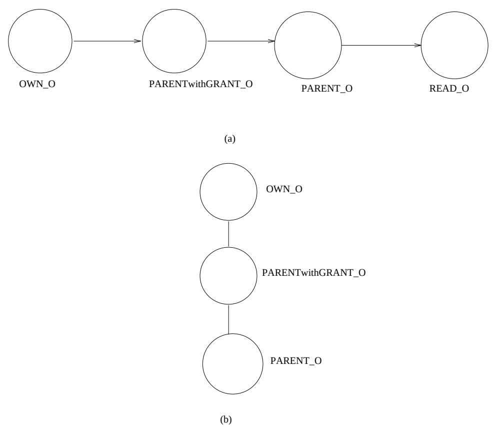


Figure 5.1: (a)Administration of roles associated with an object (b) Administrative role hierarchy

1. The cardinality of $OWN\_O,\ PARENT\_O\ and\ PARENTwithGRANT\_O$ roles is one.

2. The membership of $OWN\_O,\ PARENT\_O\ and\ PARENTwithGRANT\_O$ roles cannot change. □

**Definition 37** Granting access to user $u'$ to read object O results in following change in RBAC system:

$$UA' = UA \cup \{(u', READ\_O)\}$$

**Definition 38** Revoking access for user $u'$ to read an object $O$ results if following changes in RBAC system.

$$UA' = UA \Leftrightarrow \{(u', READ\_O)\}$$

To simulate this particular DAC policy we actually need only two roles, $OWN\_O$ and $READ\_O$. However, for consistency we give this more general construction since it applies to other forms of DAC we consider.

## 5.5 Simulation of Liberal DAC

The two variations of liberal DAC described in section 3 are considered separately for granting and revoking access to the objects. Here we describe the one level grant, two level and multilevel grant DAC. The revocation of users from roles is discussed later in this section.

### 5.5.1 One Level Grant

The one level grant DAC policy can be simulated in RBAC96 by relaxing the constraint on Strict DAC. The cardinality restriction of $PARENT\_O$ is removed. The member of $OWNER\_O$ role can assign users to $PARENT\_O$ role who can assign users to the $READ\_O$ role. The constraint in RBAC96 is modified as follows.

**Definition 39** The constraints for one level grant DAC are given below.

1. The cardinality of $OWN\_O\ and\ PARENTwithGRANT\_O$ roles is one.
2. The membership of $OWN\_O\ and\ PARENTwithGRANT\_O$ roles cannot change.

□

### 5.5.2 Two Level Grant

In two level grant DAC policy there are two types of users, those who can only grant read access and those who can grant read access with a permission to grant it further. In our simulation read access to object O is granted by the members of role $PARENT\_O$. Members of role $PARENTwithGRANT\_O$ can assign users to roles $READ\_O$ as well as to $PARENT\_O$. Assigning a user to $PARENT\_O$ authorizes that user to assign other users to have read access to object O. On the other hand assigning a user to regular role $READ\_O$ means the user can only have read access to object $O$ but cannot grant it further. Thus two level grant can be achieved by assigning users to roles $READ\_O$, $PARENT\_O\ and\ PARENTwithGRANT\_O$ as appropriate. This is specified by further relaxing the constraint defined above as follows.

**Definition 40** The constraints for two level grant DAC policy in RBAC:

1. The cardinality of $OWN\_O$ role is one.
2. The membership of $OWN\_O$ role cannot change. □

**Definition 41** Grant of read access to user $u'$ on object $O$ with a permission to grant read makes following changes.

- $AUA' = AUA \cup \{(u', PARENT\_O)\}$
- $AUA' = AUA \cup \{(u', PARENTwithGRANT\_O)\}$ □

### 5.5.3 Multilevel Grant

To grant access beyond second level we need members of role $PARENTwithGRANT\_O$ to be allowed to assign users to $PARENTwithGRANT\_O$. In two level grant only members of $OWN\_O$ have this permission. To simulate multilevel grant DAC we need to allow the members of the role $PARENTwithGRANT\_O$ to have the permission as well. The association of this permission to the role $PARENTwithGRANT\_O$ serves the purpose. Now not only the members of $OWN\_O$ but also the members of $PARENTwithGRANT\_O$ can assign users to roles $PARENTwithGRANT\_O$ and multilevel grant DAC can be achieved.

**Definition 42** Grant of read access to user $u'$ on object $O$ with a permission to grant read as well as grant it to other users makes the following changes.

- $AUA' = AUA \cup \{(u', PARENTwithGRANT\_O)\}$
- $APA' = APA \cup \{(addParentWithGrant\_O, PARENTwithGRANT\_O)\}$ $\square$

This definition authorizes members of role $PARENTwithGRANT\_O$ to assign users to role $PARENTwithGRANT\_O$

## 5.6 DAC with change of Ownership

In above variations we have restricted discussion to the case where the ownership remains with the creator of the object. This is not always true of DAC policies. There are cases where the change in ownership occurs. Change of ownership can be simulated by changing the constraints in RBAC96 as follows.

**Definition 43** The constraints for Multilevel grant DAC policy in RBAC96 is that the cardinality of $OWN\_O$ role is one but membership to it can be modified. $\square$

**Definition 44** Change of ownership from user $u$ to $u'$ means:

$$AUA' = AUA \Leftrightarrow \{(u, OWN\_O)\} \ \cup \ \{(u', OWN\_O)\}$$

## 5.7 Revocation of access

So far we have explained only the process of granting an access. This section describes the revocation of access. We are considering two cases of access revocation as follows.

### 5.7.1 Revocation is independent of granter.

This variation is easy to simulate in our constructions. It simply requires deleting a user from the membership of a role. For example if we want to revoke user $u$ from reading an object $O$ then the tuple $(u, READ\_O)$ is required to be removed from relation $UA$ of RBAC96. The following definition describes this formally.

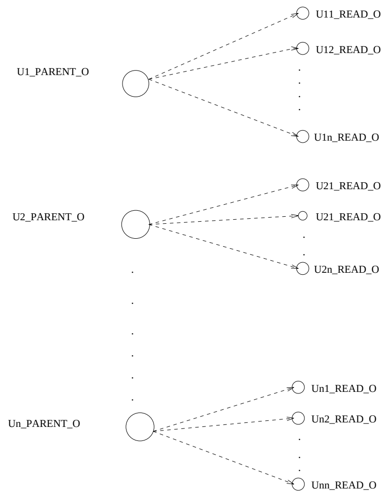


Figure 5.2: Read_O Roles associated with members of PARENT_O

**Definition 45** Revoking a user $u$ from reading an object $O$ is achieved by making following changes in RBAC:

$$UA' = UA \Leftrightarrow \{(u, READ\_O)\}$$

The members of administrative roles $PARENT\_O$ and $PARENTwithGRANT\_O$ are authorized to do these changes. Revocation does not depend upon how user $u$ has got the access to read $O$. Similarly we can define revocation from $PARENT\_O$ and $PARENTwithGRANT\_O$ roles.

### 5.7.2 Revocation only by granter of access

In this case only the user who has granted access to another user can revoke the access. The above simulation does not have this restriction. To simulate grant-dependent revocation we need to make the following changes in our construction. To explain this variation let us consider the one level grant DAC policy where members of $PARENT\_O$ role can assign users to $READ\_O$ role. We need a different administrative role $U\_PARENT\_O$ and a different regular role $U\_READ\_O$ for each user authorized to do a one level grant by the owner. These roles are automatically created when the owner authorizes user U. We also need two administrative permission created at the same time s follows.

1. $addU\_ReadUser\_O$, $deleteU\_ReadUser\_O$: respectively authorizes the operations to add users to the role $U\_RAEAD\_O$ and remove them from this role. They are assigned to role $U\_PARENT\_O$.

thereby, $U_i\_PARENT\_O$ manages the membership assignments of $U_i\_READ\_O$ role as indicated in figure 5.2 for users $U_1$, $U_2$, ..., $U_n$. The cardinality constraint of $U\_PARENT\_O$ is one. Moreover its membership cannot be changed. Thus user $U$ will be the only one granting and revoking users from $U\_READ\_O$ role.

We can allow the owner to revoke users from the $U\_READ\_O$ role by making role $U\_PARENT\_O$ junior to $OWN\_O$. Simulation of grant-dependent revocations can be similarly simulated with respect to $PARENT\_O$ and $PARENTwithGRANT\_O$ roles. This construction is cumbersome but is theoretically feasible.

In this chapter we have shown that DAC can be simulated using roles. The concept of RBAC object is introduced for the first time. The importance of this concept is that now we are able to demonstrate that administrative models built on the concept of a user object are within the purview of RBAC. Another conclusion that can be drawn from this chapter is that for each object we create four roles and eight permissions. If in a system there are hundreds of objects, we can imagine thousands of roles and permissions. The administration of such a huge number of roles and permissions is a formidable task. This is another reason why we need to decentralize administration in RBAC.

# Chapter 6

# AUGMENTED TYPED ACCESS MATRIX MODEL AND RBAC96

This chapter demonstrates the expressiveness of RBAC to accommodate other highly decentralized general administrative models. The results are theoretically important, especially in conjunction with earlier results regarding the simulation of MAC and DAC using roles. The results of this chapter show that RBAC not only subsumes both traditional forms of access control but also the highly decentralized administrative model ATAM [MS99]. Section 6.1 provides an overview of ATAM and mapping of ATAM components with RBAC. The formal simulation is described in section 6.2. In section 6.3 we give summary of ATAM simulation in RBAC.

## 6.1 Overview

As explained in chapter 2, ATAM requires that the security officer should specify the finite set of types ($T$), rights ($R_{right}$) and a set of commands as part of the system definition. The subjects and objects are created of specific types, which thereafter cannot be changed. The strong types are the principle innovation of ATAM. ATAM represents the distribution of rights by the access matrix. The access matrix has a row and column for each subject and a row for each object. Subjects are also considered as objects. Objects that have only a column in the access matrix are called pure objects. In ATAM terminology ATAM objects are denoted by symbol $OBJ$. If $SUB$ is set of subjects then set of pure Objects is $OBJ$ - $SUB$

The contents of the access matrix are changed by means of commands. The usual interpretation of ATAM command is that it is initiated by the first parameter in the parameter list of the command. For each ATAM command we create an administrative permission which performs the required checks and changes in the RBAC components. The administrative role $ADMN\_ROLE$ is created and all administrative permissions are assigned to this role. All users are made members of this administrative role. In RBAC we will have:

$AR = \{ADMN\_ROLE\}$

In our simulation we will create a role for each ATAM type. When an object $O$ is created of certain type $t$ then we create $self\_O$ role senior to role that correspond to the type $t$. At the same time we create one role for each right for $O$ along with the permissions, one for each right on the object. The permissions are assigned to the corresponding roles that cannot be changed thereafter. This means that the permission role assignments ($PA$) for these roles are initiated ONLY at the time of role creation. If the object is a subject ($S$) then a user, $user\_S$ is also created and made member of the roles $self\_S$ and $ADMN\_ROLE$. In this way if $S$ is the first parameter of the command then $user\_S$ will be able to make the checks and changes in the components of RBAC.

Deleting an object $X$ removes the roles, permissions and $user\_X$, if any, corresponding to the object $X$ from the RBAC system. The removal of the $user\_X$ will require its revocation from all roles it is explicit member of.

A change in the set $SUB$ or $OBJ$ or the change in the contents of any cell of AM changes the protection state. The change in $SUB$ and $OBJ$ changes the set of roles in the system whereas the change in the contents of the cells of AM changes the $UA$ and $PA$ components of the equivalent RBAC. This suggests that we may consider the set of components $U$, $R$, $PA$, $UA\ and\ RH$ of RBAC to represent the protection state of the ATAM system.

## 6.2 Formal description of the simulation

In this section we formally describe the simulation of ATAM system using roles. In the RBAC and ATAM mapping we use the RBAC terminology on the left hand side and ATAM on the right hand side of the equations and expressions. Therefore R appearing on left side of expression is 'ROLE' where as *R_right* on right side is the set of ATAM rights. The mapping of ATAM definition to RBAC constructions is as follows:

### 6.2.1 Administrative role ADMN_ROLE

We create an administrative role $ADMN\_ROLE$. It is represented by RBAC component as follows.

$$AR = \{ADMN\_ROLE\}$$

### 6.2.2 ATAM types are RBAC roles

For each ATAM type we will have an RBAC role. For example: if we have the set of ATAM types as $T = \{t_1, t_2, t_3, ...., t_n\}$ then in RBAC simulation we will create n roles namely, $t_1, t_2, t_3, ...., t_n$. In RBAC we will have

$$R \ \supseteq \ \{t \mid t \in T\}$$

### 6.2.3 Mapping of ATAM rights and ATAM Objects

For each ATAM object $X$ we create

- Role $self\_X$ senior to type role this object is created of.
- Roles corresponding to the object and rights (one for each right) and
- Corresponding Permissions (one for each right).
- However if the object is a subject then we also create a user, $user\_X$ and a role $session\_X$.
- $user\_X$ is assigned the membership of the $self\_X$ and $ADMN\_ROLE$.

For example, the creation of object $X$ of type $t$ in ATAM is equivalent to create a role $self\_X$ senior to role $t$ and the corresponding roles for object $X$ are $r_1\_X, r_2\_X, ....r_m\_X$ with corresponding permissions $Can\_r_1\_X, Can\_r_2\_X, ....., Can\_r_m\_X$ assuming that $R_{right} = \{r_1, r_2, r_3, ...r_m\}$. Furthermore if the object is a subject then a $user\_X$ and a $session\_X$ are created. The user $user\_X$ is made member of the role $self\_X$ and $ADMN\_ROLE$. This leads to the following mapping of ATAM to RBAC.

$U = \{user\_X \mid X \in SUB\}$

$R = \{t \mid t \in T\} \cup \{r\_X \mid r \in R_{right} \wedge X \in OBJ\} \cup \{self\_X \mid X \in OBJ\}$

$P = \{Can\_r\_X \mid r \in R_{right} \wedge X \in OBJ\}$

$Sessions = \{session\_X \mid X \in SUB\}$

$Constraint : user(session\_X) = user\_X$

$UA = \{(user\_X, self\_X) \mid X \in SUB\}$

$AUA = \{(user\_X, ADMN\_ROLE) \mid X \in SUB\}$

### 6.2.4 The ATAM commands:

Each ATAM command is equivalent to some changes in the components of RBAC. This is achieved by creating an administrative permission for each command. These permissions will perform the equivalent checks and changes in the components of RBAC. For example, for ATAM command *create_object_O* we will create an administrative permission *create_object_O* that will perform equivalent checks and changes in RBAC components. These administrative permissions are assigned to the role *ADMN_ROLE*. That is

$AP = \{\alpha \mid \alpha\ is\ an\ ATAMcommand\}$

$APA = \{(\alpha, ADMN\_ROLE) \mid \alpha\ is\ an\ ATAMcommand\}$

The ATAM command consists of three parts namely parameter list, condition and body. The translation of each of these is as follows.

- **Parameters of the command:**

  Formal parameters of ATAM commands are the checks of existing objects for the memberships or relationships of roles that map with the ATAM types.

  The Command $\alpha(X_1 : t_1, X_2 : t_2, X_3 : t_3, ..., X_k : t_k)$ is simulated as follows.

  1. Check if role $self\_X_i$ is senior to role $t_i$ for existing objects and
  2. If $X_i$ is the first parameter in the parameter list then checking the membership of $user\_X_i$ in role $self\_X_i$ and $ADMN\_ROLE$.

  Example: Command $\alpha(X_1 : t_1, X_2 : t_2)$

```
If .....
        Create object $X_2$ of type $t_2$
end
```

The above conditions for $X_1 : t_1$ will be checked whereas the checks for $X_2$ will not be performed because $X_2$ is not an existing object. On the other hand the role $self\_X_2$ senior to $t_2$ along with the roles and permissions (depending upon the set of rights) as explained in section 6.2.3 will be created as a result of 'Create object $X_2$ of type $t_2$' operation in the body of the command.

- **Condition part of the command:** The condition part of the command is the testing for the membership and non-membership in a role. For example

  a) If $r \in [Xs_1, Xo_1]$ will be translated as $user\_Xs_1 \in r\_Xo_1$ and

  b) If $r \notin [Xs1, Xo1]$ will be translated as $user\_Xs_1 \notin r\_Xo_1$

- **The body of the command:**

  The body of the command consists of the *ATAM operations* which are translated as follows.[1]

  – **Create object/subject $X$ of type $t$** is translated as creation of self_X role senior to role t, all right related roles (i.e., $r_1\_X, r_2\_X, ..., r_n\_X$) and permissions (i.e., $Can\_r_1\_X, Can\_r_2\_X, ....., Can\_r_m\_X$) with respect to object X. In RBAC this is captured by the following changes in the components:

  $R' = R \cup \{self\_X, r_1\_X, r_2\_X, ..., r_n\_X\}$

  $P' = P \cup \{Can\_r_1\_X, Can\_r_2\_X, ....., Can\_r_m\_X\}$

  $$PA' = PA \cup \{(r_1X, Can\_r_1\_X), (r_2X, Can\_r_2\_X), .....,\\ (r_mX, Can\_r_m\_X)\}$$

  $RH' = RH \cup \{(self\_X > t)\}$

[1]In this section the RBAC terminology is used on both right and left side of the equations and expressions.

If the object is a subject then we also create $user\_X$ and assign it the membership of roles $self\_X$ and $ADMN\_ROLE$. This will be equivalent to the following changes in RBAC components:

$U' = U \cup \{user\_X\}$

$UA' = UA \cup \{(user\_X, self\_X), (user\_X, ADMN\_ROLE)\}$

– **Enter r into [X, Y]** is translated as making $user\_X$ member of role $r\_Y$. The effect in RBAC is

  $UA' = UA \cup \{(user\_X, r\_Y)\}$

– **Delete r from [X, Y]** revokes the membership of $user\_X$ from role $rY$. The effect is:

  $UA' = UA \Leftrightarrow \{(user\_X, r\_Y)\}$

– **Destroy object/subject $X$:** In ATAM the effect of this operation is removal of row or/and column associated with the object/subject X from access matrix. On the RBAC side this requires the deletion of all roles and permissions associated with the object X. This is depicted by the following component changes:

  $R' = R \Leftrightarrow \{r_1X, r_2X, ..., r_nX\}$

  $P' = P \Leftrightarrow \{Can\_r_1\_X, Can\_r_2\_X, ....., Can\_r_m\_X\}$

  $PA' = PA \Leftrightarrow \{(r_1X, Can\_r_1\_X), (r_2X, Can\_r_2\_X), .....,$
  $(r_mX, Can\_r_m\_X)\}$

  $RH' = RH \Leftrightarrow \{(self\_X > t)\}$

  However if the object is a subject then we require that first the explicit membership of $user\_X$ be revoked from all other roles.

  $U' = U \Leftrightarrow \{user\_X\}$

  $(\forall r \in R) UA' = UA \Leftrightarrow (user\_X, r)$

## 6.3 Summary:

In this section we give the summary of the above translation of ATAM system into RBAC. On the left side of the expression we use RBAC terminology whereas on right is ATAM terminology. Therefore R on right represents the set of ATAM rights and on left it is set of ROLES.

### 6.3.1 Definition of RBAC components

$U = \{user\_X \mid X \in SUB\}$

$R = \{t \cup t \in T\} \cup \{r\_X \mid r \in R_{right} \wedge X \in OBJ\} \cup \{self\_X \mid X \in OBJ\}$

$AR = [ADMN\_ROLE]$

$P = \{Can\_r\_X \mid r \in R \wedge X \in OBJ\}$

$PA = \{(Can\_r\_X, r\_X) \mid r \in R \wedge X \in OBJ\}$

$AP = \{\alpha \mid \alpha \ is\ an\ ATAM\ command\}$

$APA = \{(\alpha, ADMN\_ROLE) \mid \alpha \ is\ an\ ATAM\ command\}$

$Sessions = \{session\_X \mid X \in SUB\}$

$UA = \{(user\_X, self\_X) \mid X \in SUB\}$

$AUA = \{(user\_X, ADMN\_ROLE) \cup X \in SUB\}$

$RH = \{self\_X > t \mid X \in OBJ\}$

Constraint on sessions:

- $user(session\_X) = user\_X$
- A user can have only one session that persists as long as the user remains in the system.

Constraints on PA and APA:

- The permissions are assigned to the roles only at the time of creation and cannot be changed thereafter.
- There is one administrative permission per ATAM command and they are assigned to administrative role $ADMN\_ROLE$.

## 6.4 Example

In this subsection we demonstrate the ATAM simulation in RBAC using the example of Liberal multilevel DAC policy with one level grant option of chapter 6.

### 6.4.1 Multi-level DAC policy and its simulation

In multi-level DAC policy the owner of the object can grant authority to users who in turn can use this authority to grant access to the object. So Alice being the owner of the object $O$ can grant access to Bob. Now Bob can grant access to Charles. But Bob cannot grant Charles the power to further grant access to Dorothy. The ATAM solution to this policy is as follows:

Types: $\{s,\ o\}$

Rights: $\{own,\ read,\ ReadwithGrant\}$

ATAM commands are as follows.

1. Command $Create\_Object(S : s; O : o)$

```
            create object $O$ of type $o$
            enter $own$ in $[S, O]$
            enter $read$ in $[S, O]$
        end
```

2. Command $Grant\_Read\_ObjectWithGrant(S : s; S' : s; O : o)$

```
        If $own \in [S, O]$ then
```

```
            enter $ReadwithGrant$ in $[S', O]$
        end
```

3. Command $Grant\_Read\_Object(S : s; S' : s; O : o)$

```
        If $own \in [S, O]$ or $ReadwithGrant \in [S, O]$ then
            enter $read$ in $[S', O]$
        end
```

For simplicity we do not consider the revoke commands in this example.

### 6.4.2 RBAC simulation of ATAM solution

In this section we will demonstrate the use of mappings described in section 4 to translate the ATAM solution into RBAC96 constructions.

1. Create an administrative role $ADMN\_ROLE$.
2. The types are the RBAC96 roles: Thus we create two roles $S\_ROLE$ for $s$ and $O\_ROLE$ for $o$ type.
3. The ATAM commands can be implemented by checking the memberships and relationships. Depending upon the results of these checks, roles and permissions are created/deleted and user role and permission role assignments are done. Here we show the effects of ATAM command with respect to RBAC96:
   - Command $Create\_Object(S : s; O : o)$

```
            create object $O$ of type $o$
            enter $own$ in $[S, O]$
            enter $read$ in $[S, O]$
        end
```

Is equivalent to an administrative permission Create_Object that can perform following checks and changes.

if $self\_S \ > \ S\_ROLE$ and $user\_S \in self\_S$

then

$R' = R \cup \{self\_O, OWN\_O,\ ReadWithGrant\_O,\ READ\_O\}$ and

$P' = P \cup \{CanOwn\_O,\ CanReadWithGrant\_O,\ CanRead\_O\}$

$PA' = PA \cup \{(OWN\_O,\ CanOwn\_O),\ (ReadWithGrant\_O,$
$\qquad CanReadWithGrant\_O), (READ\_O,\ CanRead\_O)\}$

$RH' = RH \cup \{self\_O > O\_ROLE\}$

$UA' = UA\{(user\_S, OWN\_O),\ (user\_S,\ READ\_O)\}$

That is, if $self\_S > S\_ROLE$ then create RBAC96 roles $self\_O$, $OWN\_O$, $READ\_O$, $ReadWithGrant\_O$ with the permissions $CanReadWithGrant\_O$, $CanOwn\_O$ and $CanRead\_O$ are created. The role $self\_O$ is made senior to role $O\_ROLE$ and we also do the permission role assignments.

- Command $Grant\_Read\_ObjectWithGrant(S : s; S' : s; O : o)$

```
        If $own \in [S, O]$ then
            enter $ReadwithGrant$ in $[S', O]$
        end
```

Is equivalent an administrative permission $Grant\_Read\_ObjectWithGrant$ that can perform following.

if $user\_S \in self\_S\ and\ self\_S > S\_ROLE$ and $self\_S' > S\_ROLE$
and $self\_O > O\_ROLE$

then

$UA' = UA \cup \{user\_S'\},\ ReadWithGrant\_O)$

That is, if $self\_S\ and\ self\_S'$ roles are senior to $S\_ROLE$ and $self\_O$ is senior to $O\_ROLE$ then $user\_S'$ are assigned the membership of

$ReadWithGrant\_O$ role.

- Command $Grant\_Read\_Object(S2 : s; S3 : s; O : o)$

```
        If $own \in [S2, O]$ or $ReadwithGrant \in [S2, O]$ then
            enter $read$ in $[S3, O]$
        end
```

  The RBAC96 administrative permission $Grant\_Read\_Object$ can do following equivalent changes.

```
If $user\_S2 > S\_ROLE$ and $user\_S \in self\_S$ and
        $user\_S3 > S\_ROLE$ and
        $self\_O > O\_ROLE$ and
        ($user\_S2 \in OWN\_O$ or $user\_S2 \in ReadWithGrant\_O$)
then
$UA' = UA \cup (user\_S3,\ READ\_O)$
```

The three administrative permissions are assigned to the role $ADMN\_ROLE$.

## 6.5 Simulation of RBAC in ATAM

In previous section we have described that RBAC can easily accommodate ATAM. Therefore RBAC is as expressive as ATAM. Is ATAM as expressive as RBAC? In this section we try to simulate RBAC using ATAM. We will agree that ATAM cannot accommodate RBAC conveniently. For simplicity we will not consider constraints in RBAC for this simulation.

For RBAC simulation using ATAM, we define following mappings.

RBAC users (U), roles (R), administrative roles (AR) and sessions (S) are ATAM subjects (SUB). RBAC permissions (P) and administrative permissions (AP) are ATAM rights ($R_{right}$).

Table 6.1: Access matrix for ATAM simulation of RBAC

| | U | R | R’ | S |
|---|---|---|---|---|
| U | | assign | assign | |
| R | | x, x1 | | |
| R’ | | | x2, x3 | |
| S | own | | | |

For simulation we will make use of following ATAM types and rights,

Types T = {U, R, AR, S} Rights $R_{right}$ = {assign, own} $\cup\{P \cup AP\}$

Creation of a RBAC user, role or session is simulated as addition of a column and a row in access matrix (AM). An example of access matrix (AM) is shown in table 6.1. Translation of the contents of AM is as follows.

User U is assigned to roles R and R’. Role R is associated with permissions x1, x2 and R’ is associated with permissions x3, x4. The AM also shows that session S is created by user U.

**User-role assignments (UA)** are controlled by *can-assign* relation. This is simulated ny ATAM commands. Prerequisite condition is the condition part and assignment is body of the ATAM command. For example, second tuple of *can-assign* relation shown in table 2.2 is simulated as follows.

```
Command PSO1_can_assign_PE1(PSO1: AR; U1: U; ED: R; QE: R)
        If $assign \in [U1, ED]$ and $assign \notin [U1, QE1]$ then
            enter assign in [U1, PE1]
        end
```

tuple 9 authorizes administrative role DSO and tuple 11 authorizes SSO to assign user to PE1. The ATAM commands that will simulate this authorization are as follows.

Command DSO_can_assign_PE1(DSO: AR; U1: U; ED: R)

```
        If assign ∈ [U1, ED] then
                enter assign in [U1, PE1]
        end
Command SSO_can_assign_PE1(SSO: AR; U1: U; ED: R)
        If assign ∈ [U1, ED] then
                enter assign in [U1, PE1]
        end
```

The concept of authority range cannot be directly silulated in ATAM. We create ATAM commands, one for each administrator that is authorized to do user-role assignment in a regular role. Therefore a change in authority range or in prerequisite condition requires a change in ATAM commands.

Similar commands can be created for revocation of users from roles using information in *can-revoke* relation.

**Permission-role assignments (PA)** in RBAC are controlled by *can-assignp* relation. PA is simulated by ATAM commands using similar constructions used for UA. The main difference in these commands is that we cannot pass permission as a parameter. The ATAM command for tuple 3 in *can-revoke* relation shown in table 2.4 is simulated as follows.

```
Command PSO1_assign_p_to_PE1(PSO1: AR; ED: R; QE: R; PE1:R)
        if $assign \in [U1, ED]$ $and$ $assign \notin [U1, QE1]$ then
                enter p in [PE1, PE1]
        end
```

Similar commands for other administrators and for other permissions can be added. It is obvious that a large number of commands will be required to simulate PA. Revocation of permissions can be simulated by adding similar commands using *can-revokep*

Table 6.2: Access matrix for ATAM simulation of role hierarchies

| | U | U' | PL1 | PE1 | QE1 | E1 |
|---|---|---|---|---|---|---|
| U | | | assign | | | |
| U' | | | | assign | | |
| PL1 | | | | | | |
| PE1 | | | assign | | | |
| QE1 | | | assign | | | |
| E1 | | | | assign | assign | |

relation.

**Role hierarchies (RH)** simulation in ATAM requires an understanding of role-role relationship concept we will use in ATAM. To simulate a role-role relationship we use the concept of assigning a role to another role. For example if we want role E1 to be junior to role PE1 (as in figure 1.1 (a)) then we assign role E1 to role PE1. This means that a right assign in a cell of access matrix (AM) shows that there is a relationship among two roles. An example of role hierarchy of range (E1, PL1) is shown in table 6.2. ATAM right assign in cell [E1,PE1] indicates that Role E1 is assigned to role PE1 thereby making E1 junior to PE1. Right assign in [E1, PE1] indicates that E1 is junior to QE1 and so on. Note the difference of having a right assign in cells [U, PL1] and [PE1, PL1]. The first value is an indication that user U is a member of role PL1, while second value indicates that the relationship among PE1 and PL1.

RRA97 model defines restrictions on the role-role assignments. Administrative roles can create role, delete role, insert edges and delete edges between roles only if these operations do not produce anomalous side effects[2]. We have found that following concepts are not easy to simulate in ATAM commands.

[2]Discussed in chapter 3

- Authority range do not partially overlap.
- Encapsulated range.
- Authority range is required to remain encapsulated when RH changes.
- The concept of create range.
- End points of an authority range cannot be deleted.
- Deletion of role requires that members of deleted role be assigned to immediate senior roles and associated permission to immediate junior roles.

**Sessions (S)** creation in RBAC is easy to simulate using following ATAM command.

```
Command create_session(U1:User; S1:S)
                    create subject S1 of type S
                    enter own in [U1, S1]
            end
```

For the simulation of use of a permission in a session we create following command.

```
Command can_use_permission_x(S:Session;, U:User; R:Role)
If assign ∈ [U, R] and own ∈ [U, S] and x ∈ [R, R] then
                    Enter x in [S, S]
                    %[Commands to use permission x]
                    delete x from [S, S]
            end
```

The philosophy behind this command is that in RBAC permissions can be revoked from a role at any time. Therefore ATAM system requires to perform checks if the permission is associated with role at the time it was used or not. On successful check it enters permission in appropriate cell to allow session to use it and after its use the permission should be deleted from there. In the interval between entering and deleting permission, the command should allow the use of permission. ATAM system does not provide this syntax. This is to be taken care programatically. Another issue here is this command allows the use of explicitly assigned permissions to a role. For implied permissions we require a mechanism to check the membership of implied permission to a role. The program for the command can make such checks from AM.

The constructions we have shown in this section are for the assignments of users and permissions to regualr roles. We can create similar commands for administrative user-role assignments (AUA) and administrative permission-role assignments (APA).

## 6.6 Summary

In this chapter we have shown that RBAC provides an open ended general framework for access control. Not only the Policies like MAC and DAC can be accommodated in RBAC framework but models such as HRU, TAM and ATAM can be reduced to RBAC. We have also shown that ATAM, on the other hand, cannot conveniently accommodate RBAC.

# Chapter 7

# CONCLUSION

This chapter lists the contributions of this thesis and presents an insight into the future directions. The contributions of this thesis are presented in 7.1 and future directions are given in section 7.2.

## 7.1 Contributions

In this thesis we have completed the Administrative models for Role-based Access Control (ARBAC) By formally defining the model for Role-Role Assignments (RRA97). This is significant as there has been no formalism to control role to role relationships. The effect of role-role assignment is to construct a role hierarchy in which senior role inherits permissions from junior roles. The model provides a decentralized administration of role hierarchies. It gives administrative roles autonomy within a range but only so far as the side effects of the resulting actions are acceptable. The model disallows some operations authorized by authority range, thereby tempering the administrative role's authority. We have formally identified these restrictions and provided their motivations.

We then address the concept of mobile and immobile users and permissions in context of administration of user-role and permission-role assignments for the first time in this arena. Introduction of mobility in URA97 and PRA97 make the constructions cumbersome, and the models difficult to use. We have formally defined enhancements to these models to give rise to URA99 and PRA99 models with out changing the basic intuition, that is the decentralization of administration of user-role, permission-role and role-role assignments by means of administrative roles, prerequisite conditions and role ranges. With the modifications we obtain the ARBAC99 model. Thai is ARBAC99 contains URA99, PRA99 and RRA97.

Role-based Access Control is a promising alternative to traditional discretionary access control (DAC and mandatory access control (MAC). Our final contribution in this thesis is to show that RBAC is sufficiently powerful to accommodate DAC. The DAC variations are defined and then simulated to show the simplicity and flexibility of RBAC to accommodate them. We have defined strict and liberal DAC and the simulation covers the issues like one level, two level, multilevel grant, DAC with change of ownership, grant-independent revocation and grant-dependent revocation. Our contribution shows that RBAC is sufficiently powerful to simulate DAC. This raises an important question as to weather it can accommodate the Access Control Models based on Discretionary Access like HRU, TAM or Augmented Types Access Matrix Model (ATAM). To answer this issue we have selected the ATAM because, firstly it can accommodate HRU and TAM and secondly it is recognized as the current state of art with respect to formal models for generalized access control, and shown that ARBAC can easily accommodate this highly decentralized administrative model. In combination with previous results on simulating MAC in RBAC [NO96, San96] our work shows RBAC accommodates bot MAC and DAC as special cases.

## 7.2 Future Research

Based on the research work in this thesis, we propose the following future research directions.

Table 7.1: *can-assign* with contradicting tuples

| Administrative Role | Prerequisite Condition | Role Range |
|---|---|---|
| PSO1 | ED | [E1, PL1) |
| PSO2 | $\overline{\text{ED}}$ | [E2, PL2) |

- ARBAC99 is a low level model which would be tedious to setup. In future there is need to look for more abstract models for administration of RBAC. This thesis provides an intuition and a good starting point in this direction.

- In ARBAC99 *can-assign*, *can-assignp*, and *can-modify* relations are used to specify restriction on user-role, permission-role and role-role assignments respectively. The relations are open ended and there are possibilities that one can add conflicting tuples in these relations. For example in *can-assign* relation one can add tuples as shown in table 7.1. This makes prerequisite condition ineffective. Similar situation may arise when we have a tuple that contradicts entry in the role range. In future, we would like to define these relations formally so that the system does not allow to add tuples in relations that contradicts any existing tuple.

- URA97 and PRA97 components of ARBAC97 are implememted [SB99]. URA97 and PRA97 models are special cases of URA99 and PRA99 models. In future we would like to see how the implementations of URA97 and PRA97 can be extended to accommodate URA99 and PRA99 models. We would also like to see the implementation of RRA97 model formally defined in this thesis for role-role assignments.

- In this thesis we have shown that ATAM cannot accommodate RBAC conveniently. In future, we need to formally define extentions in ATAM so that the issues raised in this thesis with respect to ATAM limitations can be resolved.

# BIBLIOGRAPHY

# BIBLIOGRAPHY

[AELO90] M. Abrams, K. Eggers, L. LaPadula, and I. Olson. A generalized framework for access control: An informal description. In *Proceedings of 13th NIST-NCSC National Computer Security Conference*, pages 135–143, 1990.

[ALS92] P.E. Ammann, R.J. Lipton, and Ravi S. Sandhu. The expressive power of multi-parent creation in monotonic access control models. In *Proceedings of IEEE Computer Security Foundations Workshop*, pages 148–156, Franconia, NH, June 1992.

[AS92a] P.E. Ammann and Ravi S. Sandhu. The extended schematic protection model. *The Journal Of Computer Security*, 1(3&4):335–384, 1992.

[AS92b] P.E. Ammann and Ravi S. Sandhu. Implementing transaction control expressions by checking for absence of access rights. In *Proceedings of 8th Annual Computer Security Application Conference*, pages 131–140, San Antonio, TX, December 1992.

[Bis88] M. Bishop. Theft of information in the take-grant protection model. In *Proceedings of IEEE Computer Security Foundations Workshop*, pages 194–218, Franconia, NH, June 1988.

[FK92] David Ferraiolo and Richard Kuhn. Role-based access controls. In *Proceedings of 15th NIST-NCSC National Computer Security Conference*, pages 554–563, Baltimore, MD, October 13-16 1992.

[GD72] G.S. Graham and P.J. Denning. Protection – principles and practice. In *AFIPS Spring Joint Computer Conference*, pages 40:417–429, 1972.

[GI96] Luigi Guiri and Pietro Iglio. A formal model for role-based access control with constraints. In *Proceedings of 9th IEEE Computer Security Foundations Workshop*, pages 136–145, Kenmare, Ireland, June 1996.

[HRU76] M.H. Harrison, W.L. Ruzzo, and J.D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.

[KL95] George Koch and Kevin Loney. *Oracle The Complete Reference.* Oracle Press, 1995.

[Lam71] B.W. Lampson. Protection. In *5th Princeton Symposium on Information Science and Systems*, pages 437–443, 1971. Reprinted in *ACM Operating Systems Review* 8(1):18–24, 1974.

[LM82] A. Lockman and N. Minsky. Unidirectional transport of rights and take-grant control. *IEEE Transactions on Software Engineering*, SE-8(6):597–604, 1982.

[LS77] R.J. Lipton and L. Snyder. A linear time algorithm for deciding subject security. *Journal of the ACM*, 24(3):455–464, 1977.

[McL94] J. McLean. Security models. In John Marciniak, editor, *Encyclopedia of Software Engineering.* Wiley & Sons, Inc., 1994.

[MMN90] C.J. McCollum, J.R. Messing, and L. Notargiacomo. Beyond the pale of MAC and DAC - defining new forms of access control. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 190–200, Oakland, CA, May 1990.

[MS99] Qamar Munawer and Ravi Sandhu. Simulation of augmented typed access matrix model (atam) using roles. In *1999 International Conference on Information Security, Science Hall, Shanghai, China.*, October 10-13, 1999.

[Mur93] William H. Murray. Introduction to access controls. In Zella A. Ruthberg and Hal F. Tipton, editors, *Handbook of Information Security Management*, pages 515–523. Auerbach Publishers, 1993.

[NO96] Matunda Nyanchama and Sylvia Osborn. Modeling mandatory access control in role-based security systems. In *Database Security VIII: Status and Prospects.* Chapman-Hall, 1996.

[San88] Ravi S. Sandhu. The schematic protection model: Its definition and analysis for acyclic attenuating schemes. *Journal of the ACM*, 35(2):404–432, April 1988.

[San89] Ravi S. Sandhu. The demand operation in the schematic protection model. *Information Processing Letters*, 32(4):213–219, September 1989.

[San92a] Ravi S. Sandhu. Expressive power of the schematic protection model. *The Journal Of Computer Security*, 1(1):59–98, 1992.

[San92b] Ravi S. Sandhu. The typed access matrix model. In *Proceedings of IEEE Symposium on Research in Security and Privacy*, pages 122–136, Oakland, CA, May 1992.

[San96] Ravi S. Sandhu. Role hierarchies and constraints for lattice-based access controls. In Elisa Bertino, editor, *Proc. Fourth European Symposium on Research in Computer Security*. Springer-Verlag, Rome, Italy, 1996. Published as *Lecture Notes in Computer Science, Computer Security–ESORICS96*.

[San97] Ravi Sandhu. Rationale for the RBAC96 family of access control models. In *Proceedings of the 1st ACM Workshop on Role-Based Access Control*. ACM, 1997.

[San98] Ravi Sandhu. Role-based access control. In Zelkowitz, editor, *Advances in Computers, Volume: 46*. Academic Press, 1998.

[SB96] Ravi S. Sandhu and Venkata Bhamidipati. A role-based administrative model for user-role assignment and its Oracle implementation. Technical report, Laboratory for Information Security Technology, George Mason University, 1996.

[SB97] Ravi Sandhu and Venkata Bhamidipati. The URA97 model for role-based administration of user-role assignment. In T. Y. Lin and Xiaolei Qian, editors, *Database Security XI: Status and Prospects*. North-Holland, 1997.

[SB98] Ravi Sandhu and Venkata Bhamidipati. An Oracle implementation of the PRA97 model for permission-role assignment. In *Proceedings of 3rd ACM Workshop on Role-Based Access Control*, pages 13–21, Fairfax, VA, October 22-23 1998. ACM.

[SB99] Ravi S. Sandhu and Venkata Bhamidipati. Role-based administration of user-role assignment: The URA97 model and its Oracle implementation. *The Journal Of Computer Security*, 1999. in press.

[SBC$^+$97] Ravi Sandhu, Venkata Bhamidipati, Edward Coyne, Srinivas Ganta, and Charles Youman. The ARBAC97 model for role-based administration of roles: Preliminary description and outline. In *Proceedings of 2nd ACM Workshop on Role-Based Access Control*, Fairfax, VA, November 6-7 1997. ACM.

[SBM99] Ravi Sandhu, Venkata Bhamidipati, and Qamar Munawer. The ARBAC97 model for role-based administration of roles. *ACM Transactions on Information and System Security*, 2(1), February 1999.

[SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[SG93] Ravi S. Sandhu and S. Ganta. On testing for absence of rights in access control models. In *Proceedings of IEEE Computer Security Foundations Workshop*, Franconia, NH, June 1993. 109–118.

[SM98a] Ravi Sandhu and Qamar Munawer. How to do discretionary access control using roles. In *Proceedings of 3rd ACM Workshop on Role-Based Access Control*, pages 47–54, Fairfax, VA, October 22-23 1998. ACM.

[SM98b] Ravi Sandhu and Qamar Munawer. The RRA97 model for role-based administration of role hierarchies. In *Proceedings of 14th Annual Computer Security Application Conference*, pages 39–49, Scotsdale, AZ, December 7-11 1998.

[SM99] Ravi Sandhu and Qamar Munawer. The ARBAC99 model for administration of role-based access control. In *Proceedings of 15th Annual Computer Security Application Conference*, Scotsdale, AZ, December 6-10 1999.

[SS92] Ravi S. Sandhu and G. Suri. Non-monotonic transformations of access rights. In *Proceedings of IEEE Symposium on Research in Security and Privacy*, pages 148–161, Oakland, CA, May 1992.

[SS94] Ravi Sandhu and Pierangela Samarati. Access control: Principles and practice. *IEEE Communications*, 32(9):40–48, 1994.

[SS97] Ravi S. Sandhu and Pierangela Samarati. Authentication, access control and intrusion detection. In Allen B. Tucker, editor, *The Computer Science and Engineering Handbook*, pages 1929–1948. CRC Press, 1997.

# VITA

Qamar Munawer was born on April 21, 1948, in Pakistan and is a Pakistani citizen. He received the B.S. in Physics and Mathematics from University of the Punjab, Pakistan in 1969, the M.S. in Physics from University of the Punjab, Pakistan in 1971 and M.S. in Computer Science from George Mason University, Fairfax, Virginia, USA in 1993. He was assistant professor of Physics from 1973 to 1990. During 1993-1998 he was teaching assistant with department of Information System and Software Engineering. From 1994 to 2000 he was actively involved in research in Information Security. At present he is an instructor of Sybase Inc. His current work involves class room teaching and development of technical courses that includes the implementation of Role-Based Access Control (RBAC) in Sybase products.

Permanent address: 182 D G 7/3-2 Islamabad, Pakistan.

This dissertation was typeset with LaTeX[‡] by the author.

[‡]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.