

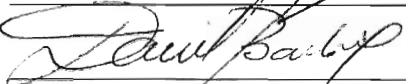
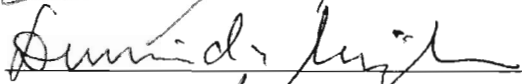





ARCHITECTURES AND MODELS FOR ADMINISTRATION OF
USER-ROLE ASSIGNMENT IN ROLE BASED ACCESS CONTROL

by

Venkata Ramana Murthy Bhamidipati
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:

	Dr. Ravi Sandhu, Dissertation Co-Director
	Dr. Daniel Menascé, Dissertation Co-Director
	Dr. Daniel Barbará, Committee Member
	Dr. Duminda Wijesekera, Committee Member
	Dr. Xinyuan Wang, Committee Member
	Dr. Daniel Menascé, Senior Associate Dean
	Dr. Lloyd J. Griffiths, Dean, The Volgenau School of Information Technology and Engineering

Date: 24th NOVEMBER 2008
Fall Semester 2008
George Mason University
Fairfax, VA

Architectures and Models for Administration of User-Role Assignment in 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

Venkata Ramana Murthy Bhamidipati
Master of Science
George Mason University, 1996
Bachelor of Engineering
Andhra University, 1994

Co-Director: Dr. Ravi Sandhu, Professor
University of Texas, San Antonio
Co-Director: Dr. Daniel Menascé, Professor
Department of Computer Science

Fall Semester 2008
George Mason University
Fairfax, VA

Copyright © 2008 by Venkata Ramana Murthy Bhamidipati
All Rights Reserved

Dedication

To Dr. Ravi Sandhu, who always provided the support and help I needed in completing this thesis work.

In loving memory of my parents Saraswathi and Krishna Mohan , who always encouraged and supported me in this endeavor.

To my wife Madhavi, who offered me unconditional love and support throughout the course of this thesis.

To my brothers, sisters and all the members of my extended family, who have given me great support for my study.

Acknowledgments

I would like to express my sincere appreciation and gratitude to my advisor and dissertation director, Professor Ravi Sandhu, who supported me and guided me throughout my doctoral studies. He always took time to help me even during the weekends and holidays. I cannot thank enough for his time and support.

Special thanks and appreciation to my dissertation co-director, Professor Daniel Menascé, who stepped in to act as my dissertation co-director and provided insight and advice on my dissertation work.

I would like to thank and extend my profound gratitude towards my dissertation committee members Professor Daniel Barbará, Professor Duminda Wijesekera and Professor Xinyuan Wang for their valuable support and help.

I would like to thank Dr. David Rine and Dr. Prasanta Bose for their support and advice during the early part of my doctoral studies.

Special thanks to Lisa Nolder for her help in taking care of administrative issues.

I would like to thank Peter Anzuini, Ram Bhandarkar and other past and present colleagues at work for their continued support and encouragement.

I would also like to thank all the staff and faculty at GMU who helped me during all these years.

Table of Contents

	Page
List of Tables	vii
List of Figures	viii
Abstract	ix
1 Introduction	1
1.1 Problem Statement	2
1.2 Contributions	3
1.3 Organization of Thesis	4
2 Background	5
2.1 RBAC96 Model	5
2.2 OM-AM framework	9
3 URA97 Model	11
3.1 URA97 Grant Model	11
3.1.1 URA97 Grant Model	11
3.1.2 Prerequisite Roles	15
3.1.3 Range Notation	16
3.1.4 Prerequisite Conditions	18
3.2 URA97 Revoke Model	20
3.2.1 Weak Revocation	21
3.2.2 Strong Revocation	22
3.3 PRA97 Model	27
3.4 Summary	29
3.5 Related Work	30
4 Push Model	34
4.1 Classification of architectures	35
4.1.1 Architectures from UA Viewpoint	36
4.1.2 Architectures from RH Viewpoint	37
4.1.3 Architectures from System Capability Viewpoint	38
4.1.4 Architectures from User and Role Occurrence Viewpoint	38
4.2 Push Model	39

4.2.1	Assigning Users to Roles	42
4.2.2	Revoking Roles from users	46
4.3	Summary	47
4.4	Related Work	48
5	ASCAA Principles	49
5.1	RBAC96 Principles	49
5.1.1	Abstraction	49
5.1.2	Separation	50
5.1.3	Least Privilege	51
5.1.4	Separation of Duty	51
5.2	ASCAA Principles	52
5.2.1	Abstraction	52
5.2.2	Separation	52
5.2.3	Containment	52
5.2.4	Automation	53
5.2.5	Accountability	56
5.3	Summary	57
6	SSRBAC08 Model	58
6.1	The SSRBAC08 Framework	59
6.2	Criteria details	63
6.2.1	Examples	67
6.2.2	Conformance to Design Principles	71
6.3	Casting URA97 in SSRBAC08	74
6.4	Example User-Role Assignment Model	76
6.4.1	User-Role Assignment	78
6.4.2	User-Role Revocation	79
6.5	Summary	82
6.6	Related Work	83
7	Conclusion	84
7.1	Contributions	84
7.2	Future Work	85
A	URA97 Implementation in Oracle	87
	Bibliography	93

List of Tables

Table	Page
3.1 Example of <i>can-assign</i> with Prerequisite Roles	16
3.2 Example of <i>can-assign</i> with Prerequisite Conditions	19
3.3 Example of <i>can-revoke</i>	23
3.4 Example of Weak Revocation	23
3.5 Example of Strong Revocation	24
3.6 Example of <i>can-assignp</i>	28
3.7 Example of <i>can-revokep</i>	28
4.1 Homogeneous versus Heterogeneous Role Hierarchy	37
4.2 User role occurrence	39
4.3 Relations for Push Model	41
4.4 Assign Algorithms	45
4.5 Revoke Algorithm	47
6.1 Example Criteria for SE1	72
6.2 Prerequisite Role in SSRBAC08	75
6.3 Prerequisite Condition in SSRBAC08	75
6.4 URA97 revoke in SSRBAC08	76
A.1 Oracle <i>can-assign</i> Relations for PSO1 from Table 3.2	88
A.2 Oracle <i>can-assign</i> Relations for Prerequisite Condition $(A \wedge D \wedge \bar{E}) \vee (B \wedge \bar{D} \wedge \bar{F})$	89
A.3 Oracle <i>can-revoke</i> Relation	89

List of Figures

Figure	Page
2.1 The RBAC96 Model	7
2.2 OM-AM Framework	9
2.3 OM-AM Framework for RBAC	10
3.1 An Example Role Hierarchy	14
3.2 An Example Administrative Role Hierarchy	14
4.1 Global Hierarchy	43
4.2 Local Hierarchies	44
6.1 SSRBAC08 components	60
6.2 An Example Role Hierarchy 2	68
6.3 Basic Self assignment Process	77
A.1 Entity-Relation Diagram for <i>can-assign</i> Relation	87

Abstract

ARCHITECTURES AND MODELS FOR ADMINISTRATION OF USER-ROLE ASSIGNMENT IN ROLE BASED ACCESS CONTROL

Venkata Ramana Murthy Bhamidipati, PhD

George Mason University, 2008

Dissertation Co-Director: Dr. Ravi Sandhu

Dissertation Co-Director: Dr. Daniel Menascé

In role based access control systems (RBAC) permissions are associated with roles, and users are made members of appropriate roles thereby acquiring the roles' permissions. This greatly simplifies management of permissions. Roles are created for the various job functions in an organization and users are assigned roles based on their responsibilities and qualifications. Users can be easily reassigned from one role to another. Roles can be granted new permissions as new applications and systems are incorporated, and permissions can be revoked from roles as needed. Role-role relationships can be established to lay out broad policy objectives. The principal motivation of RBAC is to simplify administration. In large organizations the number of roles can be in the hundreds or thousands, and users can be in the tens or hundreds of thousands, maybe even millions. To be effective, management and administration of RBAC in such systems need some form of decentralization and automation without losing central control over broad policy. An appealing possibility is to use RBAC to manage itself. Our work looks at proposing models that would allow for decentralization and automation of user-role assignment.

In this dissertation we identify architectures and models for decentralized administration of user-role assignment. Our work is performed in context of the OM-AM layered models framework. OM-AM stands for objectives, models, architectures and mechanisms. OM layer addresses security requirements and trade offs, essentially it represents “what” needs to be achieved. AM layer articulates “how” to meet the specified requirements. In this dissertation we use the terms architecture and models as they relate to OM-AM framework. Initially we focus our work on user-role assignment in a centralized system. Then we concentrate our work on user-role relationship as it pertains to distributed systems. Finally we look at how self-service and automation can be achieved in user-role assignment.

We propose a model called URA97 for user-role assignment. This model provides the semantics for granting and revoking roles from users in a centralized system. URA97 achieves assignment and revocation of users to and from roles by means of simple and intuitive relations named *can-assign* and *can-revoke*. In URA97 grant and revoke operations are performed by administrators assigned to administrative roles.

We explore some of the possible architectures in a distributed environment. These depend on how the resources, data and users are distributed and how they interact in a distributed environment. We then develop a push-based model for user-role assignment, which deals with two operations assignment of users to roles and revocation of roles from users.

URA97 was developed in context of the RBAC96 model. URA97 was developed during early stages of RBAC96 when it was still an academic discipline, since then RBAC96 has received strong support from the research and practitioner communities and today is widely practiced as preferred form of access control. It is becoming clear that relying on manual intervention in all aspects of RBAC administration is cumbersome. Concurrently access control has started adopting emerging concepts like usage control, rate limits and accountability etc. To this effect we propose five founding principles for next-generation RBAC, summarized as ASCAA for Abstraction, Separation, Containment, Automation and Accountability.

Finally we develop a framework for self service based RBAC called SSRBAC08 based on ASCAA principles. The SSRBAC08 is a modified version of RBAC96 model. The primary goal of SSRBAC08 as it pertains to our dissertation work is to show how automation, containment and accountability aspects can be achieved in user-role assignment.

Chapter 1: Introduction

Role based access control (RBAC) has gained broad acceptance from the research and practitioner communities. Several models for RBAC have been published (see, for example, [1–10]) and several commercial vendors support RBAC in their products. Adoption of the 2004 NIST/ANSI Standard RBAC Model [11] marks a maturity of concept and practice. The essential roots of this standard go back to the RBAC96¹ model. In RBAC permissions are associated with roles, and users are made members of appropriate roles thereby acquiring the roles' permissions. This greatly simplifies management of permissions. Roles are created for the various job functions in an organization and users are assigned roles based on their responsibilities and qualifications. Users can be easily reassigned from one role to another. Roles can be granted new permissions as new applications and systems are incorporated, and permissions can be revoked from roles as needed. Role-role relationships can be established to lay out broad policy objectives.

RBAC is policy-neutral and flexible. The policy that is enforced is a consequence of the detailed configuration of various RBAC components. RBAC's flexibility allows a wide range of policies to be implemented. Examples of this flexibility to support different policies can be found in [12–14].

Large RBAC systems tend to have hundreds of roles and users can be in tens of thousands. Managing and administering large RBAC system remains a challenge and a formidable task. In such large systems it is not realistic to expect administration of RBAC to be centralized within a small team of security administrators. RBAC administration needs to be decentralized to address scalability and survivability issues that arise from administration of large systems.

¹In this work we will use term RBAC96 to refer to the family of models developed by Sandhu et al [8].

In this thesis we would like to explore how to decentralize user-role assignment aspect of RBAC administration.

1.1 Problem Statement

RBAC has many components, thereby making RBAC administration multi-faceted. In particular we can separate issues of assigning users to roles, assigning permissions to roles, and assigning roles to roles to define a role hierarchy. These activities are all required to bring users and permissions together. The administrative tasks require different skills and are distinct from each other from an operational point of view. Permission-role assignment requires deep knowledge of application semantics and security needs. User-role assignment is a personnel management function, which requires greater understanding of human side. Management of overall set of roles, the role hierarchy and the policy issues related to these tasks fall into domain of business security organization.

In large enterprise-wide systems the number of roles can be in the hundreds or thousands, and users can be in the tens or hundreds of thousands, maybe even millions. Managing these roles and users, and their interrelationships is a formidable task. In order to deploy an efficient and scalable RBAC system some aspects of administration needs to be decentralized and some level of automation needs to be incorporated. We also believe that substantial advances in RBAC can further be made by reconsidering the foundational principles and expanding them to include concepts such as usage control, rate limits and automation.

In this thesis, we would like to see and formulate how user-role assignment administrative tasks could be decentralized. Distributed environments pose additional challenges to administration of roles, for example all systems may not support role hierarchies, every role may not be present in every server. We would like to see how user-role assignment can be carried out in a distributed environment. We would like to review the foundation principles of RBAC and extend them to include concepts such as usage control, rate limits and accountability. We are also interested in finding if some of the administrative tasks especially the user-role assignment can be made more self service oriented and automated

to the extent possible.

1.2 Contributions

1. Our first contribution in this thesis is that we look at user-role assignment (UA) aspect of RBAC administration. We look at how UA can be decentralized to make administration more efficient. We propose simple and intuitive relations for assignment and revocation of users from roles, named *can-assign* and *can-revoke*. We propose a user-role assignment model (URA97) to administer user-role relationship. URA97 provides semantics for granting and revoking roles from users in a centralized system. We later demonstrate that an existing popular product, namely Oracle, provides the necessary base mechanisms and extensibility to program the behavior of URA97. We then develop a model for decentralized permission-role assignment (PRA97) using similar approach.
2. Our second contribution is that we look at distributed systems to identify some of the possible architectures based on how resources, data and users are distributed. Models from user assignment perspective could include, push, pull, lookup etc. Role hierarchies could be homogeneous, heterogeneous, federated etc. and all systems may not have hierarchy capabilities. We look at these different possibilities and propose a push model to perform user-role assignment.
3. RBAC96 supports three well-known security principles: *least privilege*, *separation of duties*, and *data abstraction*. It also supports the separation of administrative functions although this principle is not explicitly articulated it is important to recognize because of its utility. RBAC96 does not actually enforce these principles, or require conformance. It is possible to completely ignore them and still technically do RBAC. We believe it is important to expand RBAC principles to include concepts like usage control, rate limits, accountability etc. This expansion of founding principles

will allow for further advancements in RBAC. Our third contribution is that we define five founding principles for next generation RBAC summarized as ASCAA for Abstraction, Separation, Containment, Automation and Accountability. Abstraction and Separation remain same from RBAC96. Containment subsumes least privilege, separation of duty from RBAC96 and includes usage and rate limits. Automation and Accountability are new principles we introduce in this work.

4. Our fourth and final contribution is that we define a framework for self service role assignment called SSRBAC08. Our design for the framework is based on ASCAA principles. We look at how RBAC96 can be modified so as to allow for self assignment of roles. The goal of SSRBAC08 is to bring some form of automation and self assignment for user-role administration tasks. We show that this can be achieved by classifying constraints as assignment, administrative, usage and revocation criteria. These criteria then can be used to formulate the assignment, usage and revocation policies for user-role administration and role usage.

1.3 Organization of Thesis

Chapter 2 gives a brief background on Role-based access control models and reviews RBAC96. Chapter 3 provides formal definition of URA97 and summarizes some of the related work done in the area. Chapter 4 covers the push model for user role assignment in a distributed system. Chapter 5 introduces the ASCAA principles for next generation RBAC. Chapter 6 provides framework for self service role assignment called SSRBAC08. Finally chapter 7 lists the contribution of this dissertation and discusses future research directions.

Chapter 2: Background

Over the years RBAC96 has remained remarkably robust and stable. Several enhancements and extensions have been proposed without changing the core concepts of the model. Administrative models for RBAC include [15–18]. Temporal considerations in RBAC were introduced in [19–21]. Separation of duty constraints in RBAC were studied in [22–24] and further explored in [25] along with other constraints. The interaction of RBAC and workflow is discussed in [26–28]. Delegation models for RBAC were first proposed in [29] and further developed in [30]. The interplay of RBAC and Trust Management has been investigated in [31, 32].

In this chapter first we will review RBAC96 model and then we review OM-AM framework which is a layered approach that helps in achieving progression from policy to mechanism.

2.1 RBAC96 Model

A general family of RBAC models called RBAC96 was defined by Sandhu et al [8]. Figure 2.1 illustrates the most general model in this family. In figure 2.1 a single headed arrow indicates a one to one relationship and a double headed arrow indicates a many to many relationship. For simplicity we overload the term RBAC96 to refer to the family of models as well as its most general member.

The top half of the figure shows roles and permissions in the system that regulate access to data and resources. The bottom half shows administrative roles and administrative permissions. RBAC96 is based on five sets of entities called users (U), roles (R), and permissions (P), and their administrative counterparts called administrative roles (AR) and

administrative permissions (AP). It is required that administrative roles and administrative permissions be respectively disjoint from the regular (i.e., non-administrative) roles and permissions. Moreover regular permissions can only be assigned to regular roles and administrative permissions can only be assigned to administrative roles.

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. Administrative permissions control operations which modify the components of RBAC, such as adding new users and roles and modifying the user assignment and permission assignment relations. Regular permissions on the other hand control operations on the data and resources and do not permit administrative operations. We loosely use the term role to include both regular and administrative roles, making this distinction precise whenever appropriate. Similarly for the term permission.

The user assignment (UA) and permission assignment (PA and APA) relations of Figure 2.1 are many-to-many, as indicated by double-headed arrow. A user can be a member of many roles, and a role can have many users. Similarly, a role can have many permissions, and the same permission can be assigned to many roles. There is a partially ordered role hierarchy RH , also written as \geq , where $x \geq y$ signifies that role x inherits the permissions assigned to role y . Equivalently $x \geq y$ signifies a user who is a member of x is also implicitly a member of y . If $x > y$ we say x is senior to y or equivalently y is junior to x . Inheritance along the role hierarchy is transitive and multiple inheritance is allowed in partial orders. There is similarly a partially ordered administrative role hierarchy ARH .

Each session in figure 2.1 relates one user to possibly many roles. Intuitively, a user establishes a session and activates some subset of roles that he or she is a member of (directly or indirectly by means of the role hierarchy). The double-headed arrows from a session to R and AR indicate that multiple roles and administrative roles can be simultaneously activated. The permissions available to the user are the union of permissions from all

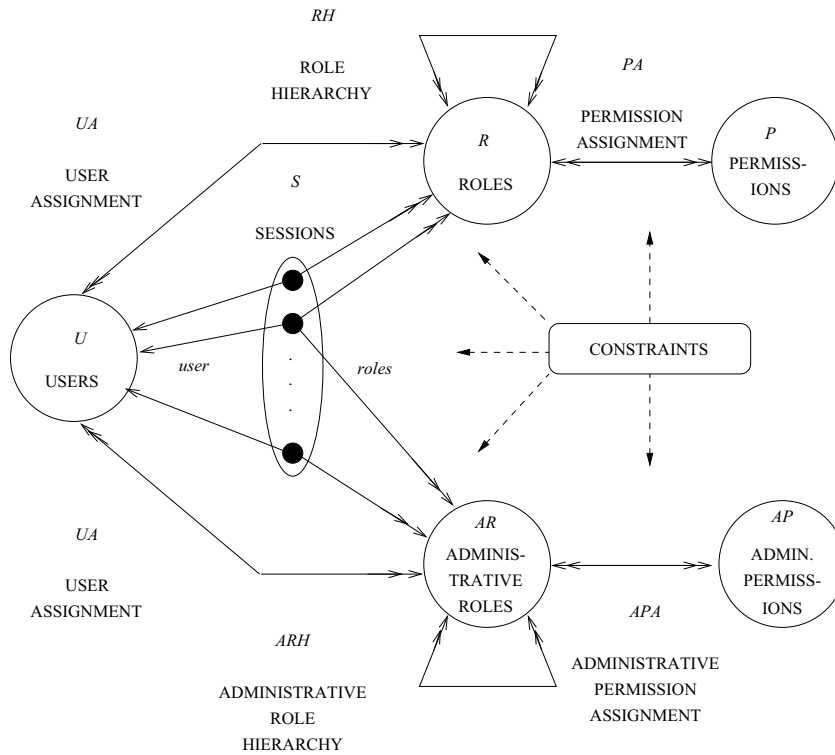


Figure 2.1: The RBAC96 Model

roles activated in that session. Each session is associated with a single user, as indicated by the single-headed arrow from the session to U . This association remains constant for the life of a session. A user may have multiple sessions open at the same time, each in a different window on the workstation screen for instance. Each session may have a different combination of active roles. The concept of a session equates to the traditional notion of a subject in access control. A subject (or session) is a unit of access control, and a user may have multiple subjects (or sessions) with different permissions active at the same time.

Finally, figure 2.1 shows a collection of constraints. Constraints can apply to any of the preceding components. An example of constraints is mutually disjoint roles, such as purchasing manager and accounts payable manager, where the same user is not permitted to be a member of both roles. Another example is a limit on the maximum number of users that can be members of some role.

The following definition formalizes the above discussion.

Definition 1. The RBAC96 model has the following components:

- U is a set of users,
- R and AR are disjoint sets of roles and administrative roles respectively,
- P and AP are disjoint sets of permissions and administrative permissions,
- $UA \subseteq U \times (R \cup AR)$, is a many-to-many user to role, and administrative role, assignment relation,
- $PA \subseteq P \times R$ and $APA \subseteq AP \times AR$, are respectively many-to-many permission to role assignment and administrative permission to administrative role assignment relations,
- $RH \subseteq R \times R$ and $ARH \subseteq AR \times AR$, are respectively partially ordered role and administrative role hierarchies (written as \geq in infix notation),
- S is a set of sessions,
- $user : S \rightarrow U$, is a function mapping each session s_i to the single user $user(s_i)$ and is constant for the session's lifetime,
- $roles : S \rightarrow 2^{R \cup AR}$ is a function mapping each session s_i to a set¹ of roles and administrative roles $roles(s_i) \subseteq \{r \mid (\exists r' \geq r)[(user(s_i), r') \in UA]\}$ (which can change within a single session) so that session s_i has the permissions $\cup_{r \in roles(s_i)} \{p \mid (\exists r'' \leq r)[(p, r'') \in PA \cup APA]\}$, and
- there is a collection of constraints stipulating which values of the various components enumerated above are allowed or forbidden. □

Motivation and discussion about various design decisions made in developing this family of models is given in [8,33]. RBAC96 distinguishes roles and permissions from administrative roles and permissions respectively, where the latter are used to manage the former.

¹Recall that 2^X is the set of all subsets of X , also called the power set of X .

The only real significant feature of RBAC standard that is missing in RBAC96 is role activation hierarchies [34]. NIST/ANSI RBAC standard allows inheritance hierarchies, activation hierarchies or some combination of them, leaving it up to vendor to specify. RBAC standard supports only separation of duty constraints, so it actually degrades the capability of RBAC96 in terms of constraints.

2.2 OM-AM framework

A layered approach called OM-AM² framework was proposed by Sandhu [36] as a way to address the security issues of authority and trust. The framework follows a layered approach analogous to OSI network layer. The four layers in OM-AM framework are objective, model, architecture and mechanism. The framework does not build one layer on top of the other rather it deals with different kinds of concepts at each layer. The mapping between each layer is many to many. Figure 2.2 depicts the OM-AM framework model. The objective

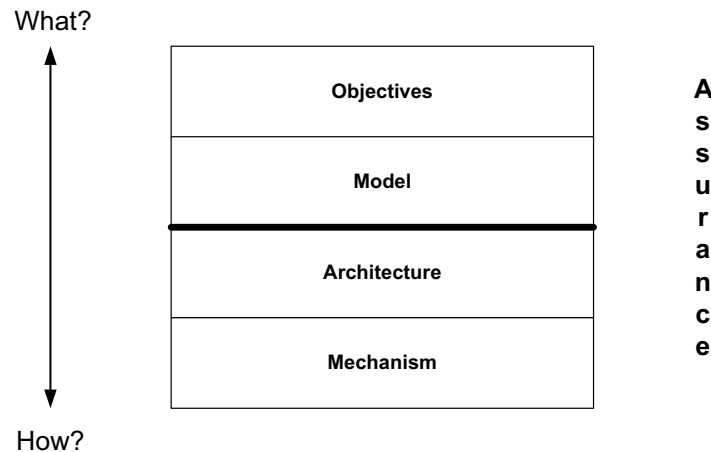


Figure 2.2: OM-AM Framework

²Subsequent to OM-AM framework Sandhu et al [35] proposed PEI models framework, which can be viewed as a generalization of OM-AM framework. The layers in PEI model are security and system goals, Policy models, enforcement models, implementation models and mechanisms.

layer represents the objectives or security goals. For example one of the stated goals of RBAC is that it is policy neutral. The model layer depicts the formal or abstract representation of the requirements. The architecture layer deals with implementation design, strategy and relationship among various components involved. Mechanism layer deals with actual implementation methods. The OM layers of the framework address “What” aspect, they together articulate what the security objectives and trade offs are. The architecture and mechanisms layers represent “How” aspect of the model and deal with realizing the objectives and requirements. Figure 2.3 represents an example OM-AM framework for RBAC systems.

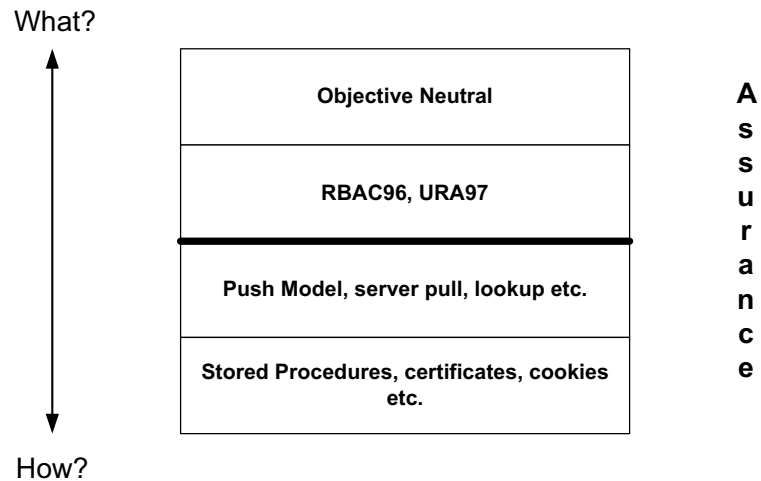


Figure 2.3: OM-AM Framework for RBAC

Our work in this dissertation is based on OM-AM framework. We use the terms architectures and models in context of the OM-AM framework.

Chapter 3: URA97 Model

As mentioned in chapter 1 large RBAC systems tend to have hundreds of roles and users can be in tens of thousands. Managing the user-role assignment is a formidable task and probably it is the first administrative function that is decentralized and delegated. In this chapter we formally define URA97. In URA97 assignments and revocations of users from roles are realized by means of administrative roles and permissions. URA97 imposes strict limits on individual administrators regarding which users can be assigned to which roles. We define URA97 in two steps, dealing with granting a user membership in a role and revoking a user's membership. Principal contribution of URA97 is to provide a concrete example of what is meant by role-based administration of user-role assignment. URA97 is defined in context of RBAC96. However, it applies to almost any RBAC model, because user-role assignment is a basic administrative feature which will be required in any RBAC model

The organization of chapter is as follows. In section 3.1 we formally define and discuss URA97 grant model followed by URA97 revoke model in 3.2. In section 3.3 we define PRA97 model for permission-role assignment which is dual of URA97. Section 3.4 summarizes URA97 model and finally in section 3.5 we discuss related work.

3.1 URA97 Grant Model

3.1.1 URA97 Grant Model

In the simplest case user-role assignment can be completely centralized in a single chief security officer role. This is readily implemented in existing systems such as Oracle. However, this simple approach does not scale to large systems. Clearly it is desirable to decentralize user-role assignment to some degree.

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

In URA97 our goal is to impose restrictions on which users can be added to a role by whom, as well as to clearly separate the ability to add and remove users from other operations on the role. The notion of a prerequisite condition is a key part of URA97.

Definition 2. A prerequisite condition is a boolean expression using the usual \wedge and \vee operators on terms of the form x and \bar{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 \bar{x} to be true if $(\forall x' \geq x)(u, x') \notin UA$. For a given set of roles R let CR denote all possible prerequisite conditions that can be formed using the roles in R . \square

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

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

Definition 3. The URA97 model controls user-role assignment by means of the relation $can\text{-}assign \subseteq AR \times CR \times 2^R$. \square

The meaning of $can\text{-}assign(x, y, \{a, b, c\})$ is that a member of the administrative role x (or a member of an administrative role that is senior to x) can assign a user whose current

membership, or non-membership, in regular roles satisfies the prerequisite condition y to be a member of regular roles a , b or c .¹

To appreciate the motivation behind the *can-assign* relation consider the role hierarchy of figure 3.1 and the administrative role hierarchy of figure 3.2. Figure 3.1 shows the regular roles that exist in a engineering department. There is a junior-most role E to which all employees in the organization belong. Within the engineering department there is a junior-most role ED and senior-most role DIR. In between there are roles for two projects within the department, project 1 on the left and project 2 on the right. Each project has a senior-most project lead role (PL1 and PL2) and a junior-most engineer role (E1 and E2). In between each project has two incomparable roles, production engineer (PE1 and PE2) and quality engineer (QE1 and QE2).

Figure 3.1 suffices for our purpose but this structure can, of course, be extended to dozens and even hundreds of projects within the engineering department. Moreover, each project could have a different structure for its roles. The example can also be extended to multiple departments with different structure and policies applied to each department.

Figure 3.2 shows the administrative role hierarchy which co-exists with figure 3.1. The senior-most role is the senior security officer (SSO). Our main interest is in the administrative roles junior to SSO. These consist of two project security officer roles (PSO1 and PSO2) and a department security officer (DSO) role with the relationships illustrated in the figure.

The role structure shown in Figure 3.1 becomes a project oriented if the users are assigned to roles in a single project. If the users are assigned to roles from multiple projects then the structure is of matrix-form and If the users are assigned to same functional role in different projects then the structure is of functional oriented.

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

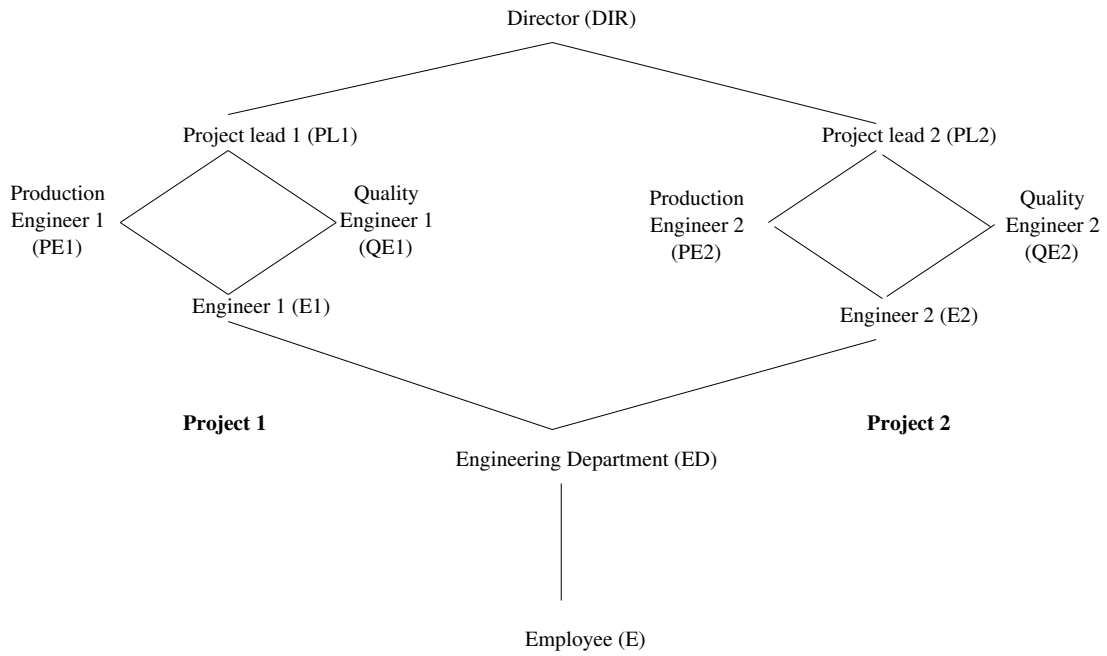


Figure 3.1: An Example Role Hierarchy

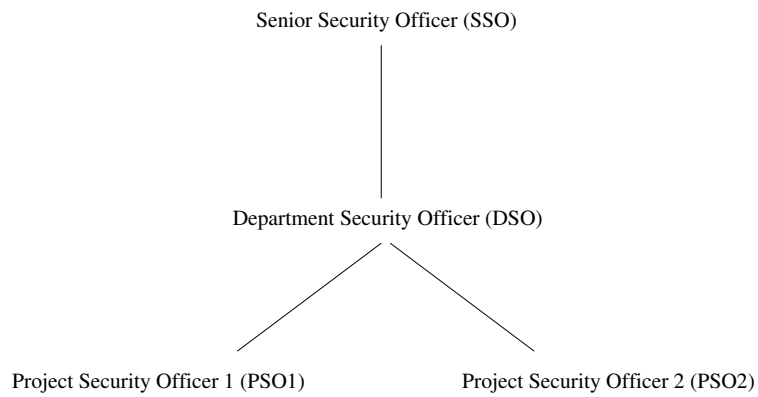


Figure 3.2: An Example Administrative Role Hierarchy

3.1.2 Prerequisite Roles

For sake of illustration we define the *can-assign* relation shown in table 3.1(a). This example has the simplest prerequisite condition of testing membership in a single role known as the prerequisite role.

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

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

The notation of table 3.1(a) has benefited from the administrative role hierarchy. Thus for the DSO we have specified the role set as $\{PL1, PL2\}$ and the other values are inherited from PSO1 and PSO2. Similarly for the SSO. Nevertheless explicit enumeration of the role set is unwieldy, particularly if we were to scale up to dozens or hundreds of projects in the department. Moreover, explicit enumeration is not resilient with respect to changes in the

Table 3.1: 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

role hierarchy. Suppose a third project is introduced in the department, with roles E3, PE3, QE3, PL3 and PSO3 analogous to corresponding roles for projects 1 and 2. We can add the following row to table 3.1(a).

Administrative Role	Prerequisite Role	Role Set
PSO3	ED	{E3, PE3, QE3}

This is a reasonable change to require when the new project and its roles are introduced into the regular and administrative role hierarchies. However, we also need to modify the row for DSO in table 3.1(b) to include PL3.

3.1.3 Range Notation

Consider instead the range notation illustrated in table 3.1(b). Table 3.1(b) shows the same role sets as table 3.1(a) but defines these sets by identifying a range within the role hierarchy of figure 2.1(a) by means of the familiar closed and open interval notation.

Definition 4. Role sets are specified in the URA97 model by the notation below

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

□

This notation is resilient to modifications in the role hierarchy such as addition of a third project which requires addition of the following row to table 3.1(b).

Administrative Role	Prerequisite Role	Role Range
PSO3	ED	[E3, PL3)

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

The range notation is, of course, not resilient to all changes in the role hierarchy. Deletion of one of the end points of a range can leave a dangling reference and an invalid range. Standard techniques for ensuring referential integrity would need to be applied when modifying the range hierarchy. Changes to role-role relationships could also cause a range to be drastically different from its original meaning. Nevertheless the range notation is much more convenient than explicit enumeration. There is also no loss of generality in adopting the range notation since every set of roles can be expressed as a union of disjoint ranges.

Strictly speaking the two specifications of table 3.1(a) and 3.1(b) are not precisely identical. In table 3.1(a) the DSO role is explicitly authorized to enroll users in PL1 and PL2, and the inherits the ability to enroll users in other project 1 and 2 roles from PSO1 and PSO2. On the other hand, in table 3.1(b) the DSO role is explicitly authorized to enroll users in all project 1 and 2 roles. As it stands the net effect is the same. However, if modifications are made to the role hierarchy or to the PSO1 or PSO2 authorizations the effect

can be different. The DSO authorization in table 3.1(a) can be replaced by the following row to make table 3.1(a) more nearly identical to table 3.1(b).

Administrative Role	Prerequisite Role	Role Set
DSO	ED	{E1, PE1, QE1, PL1, E2, PE2, QE2, PL2}

Now even if the PSO1 and PSO2 roles of table 3.1(a) are modified respectively to the role sets {E1} and {E2}, the DSO role will still retain administrative authority over all project 1 and project 2 roles. Of course, explicit and implicit specifications will never behave exactly identically under *all* circumstances. For instance, introduction of a new project 3 will exhibit differences as discussed above. Conversely, the DSO authorization in table 3.1(b) can be replaced by the following rows to make table 3.1(b) nearly identical to table 3.1(a).

Administrative Role	Prerequisite Role	Role Range
DSO	ED	[PL1, PL1]
DSO	ED	[PL2, PL2]

There is an analogous situation with the SSO role in tables 3.1(a) and 3.1(b). Clearly, we must anticipate the impact of future changes when we specify the *can-assign* relation.

3.1.4 Prerequisite Conditions

An example of *can-assign* which uses prerequisite conditions rather than prerequisite roles is shown in table 3.2. The authorizations for PSO1 and PSO2 have been changed relative to table 3.1.

Let us consider the PSO1 tuples (analysis for PSO2 is exactly similar). The first tuple authorizes PSO1 to assign users with prerequisite role ED into E1. The second one authorizes PSO1 to assign users with prerequisite condition $ED \wedge \overline{QE1}$ to PE1. Similarly, the third tuple authorizes PSO1 to assign users with prerequisite condition $ED \wedge \overline{PE1}$ to QE1. Taken together the second and third tuples authorize PSO1 to put a user who is a member

Table 3.2: Example of *can-assign* with Prerequisite Conditions

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

of ED into one but not both of PE1 and QE1. This illustrates how mutually exclusive roles can be enforced by URA97. PE1 and QE1 are mutually exclusive with respect to the power of PSO1. However, for the DSO and SSO these are not mutually exclusive. Hence, the notion of mutual exclusion is a relative one in URA97. The fourth tuple authorizes PSO1 to put a user who is a member of both PE1 and QE1 into PL1. Of course, a user could have become a member of both PE1 and QE1 only by actions of a more powerful administrator than PSO1.

In RBAC users are made members of roles because of their job function or task assignment in the interest of the organization. Prerequisite conditions allow us to specify the requirements that need to be met before granting a role to a user. Generally an employee who is an engineer is not given access to HR or Pay roll information, he has access to only his personal information. A person belonging to Pay Roll department has access to information of all the employees or employees of the division to which he has been designated. In real life many large organizations have these kind of specifications and policies in place. Prerequisite conditions provide capabilities to enforce these policies.

3.2 URA97 Revoke Model

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

In the classical discretionary approach to revocation there are at least two issues that introduce complexity and subtlety [37,38]. Suppose Alice grants Bob some permission P. This is done at Alice's discretion because Alice is either the owner of the object to which P pertains or has been granted administrative authority on P by the actual owner. Alice can later revoke P from Bob. Now suppose Bob has received permission P from Alice and from Charlie. If Alice revokes her grant of P to Bob he should still continue to retain P because of Charlie's grant. A related issue is that of cascading revokes. Suppose Charlie's grant was in turn obtained from Alice, perhaps Bob's permission should end up being revoked by Alice's action. Or perhaps it should not, because Alice only revoked her direct grant to Bob but not the indirect one via Charlie which really occurred at Charlie's discretion. A considerable literature has developed examining the subtleties that arise, especially when hierarchical groups and negative permissions or denials are brought into play (see, for example, [39–43]).

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

Similarly, suppose Alice and Charlie both grant membership to Bob in X. At some later time Bob is reassigned to some other project and no longer needs to be a member of role X. It is not material whether Alice or Charlie or both or Dorothy revokes Bob’s membership. Bob’s membership in X is being revoked due to a change in organizational circumstances.

To summarize, in classical discretionary access control the source (direct or indirect) of a permission and the identity of the revoker is typically taken into account in interpreting the revoke operation.² These issues do not arise in the same way for revocation of user-role assignment in RBAC. However, there are related subtleties that arise in RBAC concerning the interaction between granting and revocation of user-role membership and the role hierarchy. We will illustrate these in a moment.

We now introduce our notation for authorizing revocation.

Definition 5. The URA97 model controls user-role revocation by means of the relation $can-revoke \subseteq AR \times 2^R$. □

The meaning of $can-revoke(x, Y)$ is that a member of the administrative role x (or a member of an administrative role that is senior to x) can revoke membership of a user from any regular role $y \in Y$. Y is specified using the range notation of definition 4. We say Y defines the *range of revocation*. The precise semantics of revocation in URA97 needs to be carefully defined to explain its interaction with the role hierarchy.

3.2.1 Weak Revocation

In URA97 we define two notions of revocation called *weak* and *strong*. Recall that UA is the user assignment relation.

Definition 6. Let us say a user U is an *explicit member* of role x if $(U, x) \in UA$, and that U is an *implicit member* of role x if for some $x' > x$, $(U, x') \in UA$. □

Note that a user can simultaneously be an explicit and implicit member of a role.

²This is true more in theory than practice, because many commercial products opt for a simpler semantics than implied by a strict owner-based discretionary viewpoint.

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

Definition 7. [Weak Revocation Algorithm]

1. Let u have a session with administrative roles $A = \{a_1, a_2, \dots, a_k\}$, and let u try to weakly revoke v from role x .
2. If v is not an explicit member of x this operation has no effect, otherwise there are two cases.
 - (a) There exists a *can-revoke* tuple (b, Y) such that there exists $a_i \in A, a_i \geq b$ and $x \in Y$.

In this case v 's explicit membership in x is revoked.

- (b) There does not exist a *can-revoke* tuple as identified above.

In this case the weak revoke operation has no effect.

□

Let us consider the example of *can-revoke* shown in table 3.3 and interpret it in context of the hierarchies of figures 3.1 and 3.2. Let Alice be a member of PSO1, and let this be the only administrative role she has. Alice is authorized to weakly revoke membership of users from roles E1. Table 3.4(a) illustrates whether or not Alice can weakly revoke membership of a user from role E1. The effect of Alice's weak revocation of each of these users from E1 is shown in table 3.4(b). There is no effect of weak revocation on Cathy and Eve because they are not explicit members of E1 role. On the other hand Bob and Dave are removed from E1 role. Dave however still holds the E1 permissions because of his membership in senior roles.

3.2.2 Strong Revocation

Strong revocation in URA97 requires revocation of both explicit and implicit membership. Strong revocation of U's membership in x requires that U be removed not only from explicit

Table 3.3: Example of *can-revoke*

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

Table 3.4: Example of Weak Revocation

User	E1	PE1	QE1	PL1	DIR	Alice can revoke user from E1
Bob	Yes	No	No	No	No	Yes
Cathy	No	Yes	Yes	No	No	Yes
Dave	Yes	Yes	Yes	Yes	No	Yes
Eve	No	No	No	Yes	Yes	Yes

(a) Prior to Weak revocation

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

(b) After Weak revocation

membership in x , but also from explicit (or implicit) membership in all roles senior to x . Strong revocation therefore has a cascading effect upwards in the role hierarchy. However, strong revocation in URA97 takes effect only if all implied revocations upward in the role hierarchy are within the revocation range of the administrative roles that are active in a session.

Let us consider the example of *can-revoke* shown in table 3.3 and interpret it in context of the hierarchies of figures 3.1 and 3.2. Let Alice be a member of PSO1, and let this be the only administrative role she has. Alice is authorized to strongly revoke membership of users from roles E1, PE1 and QE1. Table 3.5(a) illustrates whether or not Alice can strongly revoke membership of a user from role E1. The effect of Alice's strong revocation

Table 3.5: Example of Strong Revocation

User	E1	PE1	QE1	PL1	DIR	Alice can revoke user from E1
Bob	Yes	Yes	No	No	No	Yes
Cathy	Yes	Yes	Yes	No	No	Yes
Dave	Yes	Yes	Yes	Yes	No	No
Eve	Yes	Yes	Yes	Yes	Yes	No

(a) Prior to strong revocation

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

(b) After strong revocation

of each of these users from E1 is shown in table 3.5(b). Alice is not allowed to strongly revoke Dave and Eve from E1 because they are members of senior roles outside the scope of Alice’s revoking authority. If Alice was assigned to the DSO role she could strongly revoke Dave from E1 but still would not be able to strongly revoke Eve’s membership in E1. In order to strongly revoke Eve from E1, Alice needs to be in the SSO role.

The general rule is that strong revocation takes effect within the revocation range authorized for an administrative role. The precise statement of the strong revocation algorithm becomes complicated because of the administrative role hierarchy and the possible existence of several tuples in *can-revoke* which determine the outcome. In the example above Alice is allowed to strongly revoke Cathy from E1 because of $can-revoke(PSO1, [E1,PL1])$. We should have the same result if instead of this single *can-revoke* range for PSO1 we have two ranges $can-revoke(PSO1, [E1,PE1])$ and $can-revoke(PSO1, [E1,QE1])$. Finally, because of the session concept in RBAC96 we must also pay attention to which roles Alice has turned on in the particular session. These considerations lead to the following algorithm for strong revocation.

Definition 8. Strong Revocation Algorithm

1. Let u have a session with administrative roles $A = \{a_1, a_2, \dots, a_k\}$, and let u try to strongly revoke v from role x .
2. Find all *can-revoke* tuples $(b_1, X_1), (b_2, X_2), \dots, (b_p, X_p)$ such that there exists $a_i \in A, a_i \geq b_j$ and $x \in X_j$ for $j = 1 \dots p$.
3. Let $\hat{X} = X_1 \cup X_2 \cup \dots \cup X_p$ where the union is over the actual roles identified by the ranges X_1, X_2, \dots, X_p .
4. There are two cases.

- (a) There exists $y \notin \hat{X}$ such that v is a member of y and $y > x$.

In this case u 's strong revocation has no effect.

- (b) There does not exist a role y as identified above (therefore all senior roles to x to which v belongs are in \hat{X}).

In this case v 's membership is revoked from role x and all roles senior to x .

□

In context of our example this algorithm will treat *can-revoke*(PSO1, [E1,PL1]) as equivalent to *can-revoke*(PSO1, [E1,PE1]) and *can-revoke*(PSO1, [E1,QE1]). It is similarly equivalent to *can-revoke*(PSO1, [E1,E1]), *can-revoke*(PSO1, [PE1,PE1]) and *can-revoke*(PSO1, [QE1,QE1]).

The strong revocation algorithm can also be expressed in terms of weak revoke by the following all-or-nothing transaction.

1. Let u have a session with administrative roles $A = \{a_1, a_2, \dots, a_k\}$, and let u try to strongly revoke v from role x .
2. Find all roles $y \geq x$ and v is a member of y .
3. Weak revoke v from all such y as if u did this weak revoke.

4. If any of the weak revokes fail then u 's strong revoke has no effect otherwise all weak revokes succeed.

An alternate approach would be to do only those weak revokes that succeed and ignore the rest. We decided to go with a cleaner all-or-nothing semantics in URA97.

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

Property 1. Implicit membership in a role a is dependent on explicit membership in some senior role $b > a$. Therefore when explicit membership of a user is revoked from b , implicit membership is also automatically revoked on junior role a unless there is some other senior role $c > a$ in which the user continues to be an explicit member. (This will require $b \not\prec c$.)

As we have discussed earlier, when a user's administrative roles are revoked that user's assignments and revocations remain in effect because these were done for organizational reasons and not at the user's whim. A related issue is what happens when the prerequisite condition which authorized Alice to assign Bob to a role gets changed. Say that Alice as PSO1 assigns Bob to PE1, as per the second PSO1 tuple of table 3.2. Later somehow Bob is made a member of QE1, perhaps by a user in DSO or SSO role. This assignment negates the prerequisite condition which enabled Alice to do her assignment. Bob's membership in PE1 will nevertheless continue. We feel this is the appropriate action. The prerequisite conditions of URA97 (and at other places in ARBAC97) are not invariants that hold for all time. They are simply enabling conditions at the moment that assignment is made.

As another example of the enabling but not invariant nature of prerequisite conditions consider the following in context of the *can-assign* relation of table 3.1. Suppose Alice as PSO1 enrolls Bob into PE1 due to his prerequisite membership in ED. Later Charles as SSO revokes Bob from ED. Should Alice’s assignment of Bob to PE1 be negated since the prerequisite condition has been negated? It depends on Charles’ intention, which in turn depends on the organizational reason for this revocation. If Charles really needs to clear out Bob from the engineering department the correct course of action is a strong revocation of Bob from ED. If Charles does a weak revoke of Bob’s explicit membership in ED he is leaving open the option that Bob will continue to participate in engineering department roles till such time as Bob is revoked from all of them (say by project security officers). This latter option can be useful in allowing Bob to gracefully leave the engineering department without an abrupt termination. In such cases it might be useful for Charles to be able to freeze Bob’s membership in engineering department roles so that Bob cannot be assigned to new roles. This can be done using prerequisite conditions. A role called EF (for engineering frozen) can be defined and non-membership in EF required in the prerequisite condition of all *can-assign* tuples that authorize users to be assigned to engineering department roles.

Note that our examples of *can-assign* in table 3.1(b) and *can-revoke* in table 3.3 are complementary in that each administrative role has the same range for adding users and removing users from roles. Although this would be a common case we do not impose it as a requirement on our model.

We have defined URA97 so that the same revocation range applies for both strong and weak revocation. In principle we could define different ranges for these two operations. We do not feel this added complexity would be justified.

3.3 PRA97 Model

PRA97 is concerned with role-permission assignment and revocation. From the perspective of a role, users and permissions have a similar character. They are essentially entities that are brought together by a role. Hence, we propose PRA97 to be a dual of URA97. The

Table 3.6: Example of *can-assignp*

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]

Table 3.7: Example of *can-revokep*

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

notion of a prerequisite condition is identical to that in URA97 except the boolean expression is now evaluated for membership and non-membership of a permission in specified roles.

Definition 9. *Permission-role assignment and revocation are respectively authorized in PRA97 by the following relations,*

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

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

The meaning of *can-assignp*(x, y, Z) is that a member of the administrative role x (or a member of an administrative role that is senior to x) can assign a permission whose current membership, or non-membership, in regular roles satisfies the prerequisite condition y to regular roles in range Z .³ The meaning of *can-revokep*(x, Y) is that a member of the administrative role x (or a member of an administrative role that is senior to x) can revoke membership of a permission from any regular role $y \in Y$.

³Permission-role assignment may be subject to additional constraints, just as user-role assignment is.

Tables 3.6, 3.7 show examples of these relations. The DSO is authorized to take any permission assigned to the DIR role and make it available to PL1 or PL2. Thus a permission can be delegated downward in the hierarchy. PSO1 can assign permissions from PL1 either PE1 or QE1, but not to both. The remaining rows in table 3.6 are similarly interpreted. Table 3.7 authorizes DSO to revoke permissions from any role between ED and DIR. PSO1 can revoke permissions from PE1 and QE2, and similarly for PSO2.

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

Definition 10. *Let us say a permission P is explicitly assigned to role x if $(P, x) \in PA$, and that P is implicitly assigned to role x if for some $x' < x$, $(P, x') \in PA$.*

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

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

3.4 Summary

In this chapter we developed a user-role assignment model called URA97 to perform assignment of user to roles and revocation of roles from users. URA97 controls user-role

assignment by means of the relation $can\text{-}assign \subseteq AR \times CR \times 2^R$. Role sets are specified using the range notation of definition 4. Assignment has a simple behavior whereby $can\text{-}assign(a, b, C)$ authorizes a session with an administrative role $a' \geq a$ to enroll any user who satisfies the prerequisite condition b into any role $c \in C$. The prerequisite condition is a boolean expression using the usual \wedge and \vee operators on terms of the form x and \bar{x} respectively denoting membership and non-membership regular role x .

Revocation is controlled in URA97 by the relation $can\text{-}revoke \subseteq AR \times 2^R$. Weak revocation applies only to explicit membership in a single role as per the algorithm of definition 7. Strong revocation cascades upwards in the role hierarchy as per the algorithm of definition 8. In both cases revocation cascades downwards as noted in property 1.

We also developed a model for permission-role assignment called PRA97 to perform assignment of permissions to roles and revocation of permissions from roles. PRA97 uses relations similar to that of URA97 and is a dual of it.

We prototyped URA97 behavior in Oracle RDBMS and implementation details are discussed in the appendix chapter of this thesis.

3.5 Related Work

Administration of RBAC has attracted considerable attention and remains a very active topic in RBAC research. The ARBAC97 model developed by Sandhu, Bhamidipati and Munawer [15] is probably one of the first comprehensive models to address the issue of RBAC administration. ARBAC97 comprises of URA97, PRA97 models discussed in this chapter and it also formally states RRA97 which deals with role-role assignment model. Several extensions to ARBAC97 have been proposed which try to add additional features and also address some of the perceived drawbacks or limitations in ARBAC07, examples of ARBAC07 extension include [44], [16, 17]. Some of other work on administration models include [45], [46], [18], [47].

URA97 behavior has been successfully prototyped in several systems including, databases, operating systems and intranets. Implementation details of these prototypes are discussed

in [48–51].

Closely related to our URA97 model work are URA99 and URA02 models. These models try to extend URA97 and address some of the shortcomings of it. Sandhu and Munawer propose URA99 [44] which is an extension to URA97. URA97 allows a person to be eligible for receiving other roles based on his membership in current roles. URA99 controls or limits this aspect of user’s ability to receive new roles based on his current assignment. URA99 introduces the concept of mobile and immobile membership. When a person is assigned to role with immobile membership he can use the permissions of role, however administrators cannot use this membership to make other role assignments. Mobile membership allows for a user to use permissions and administrators can use this membership to make other role assignments.

Sejong Oh and Sandhu propose URA02 [16, 17]. URA02 introduces concept of user pools and uses organization-structure (OS-U) to represent these pools. URA02 adopts *can-assign* and *can-revoke* relations from URA97, however it replaces the prerequisite roles with organization-structures. OS-U contains users who are pre assigned by Human resource group. OS-U is a tree structure and allows for inheritance. URA02 claims to mitigate multi step assignment, redundant assignments and restricted user-pools issues that could arise in URA97 model. We will analyze these claims below.

URA02 states that role assignment in URA97 tends to be multi step process. This statement is not always true. In URA97 number of steps needed depends on what role is being assigned, prerequisite conditions on the role and the administrator performing the assignment. In fact in our view URA02 does not completely address this issue either, it simply moves the problem to setting up correct OS-U and placing users at right nodes. URA02 assumes a human resources group or some other group will assign users to OS-U correctly.

Let us look at table 3.2 now say Tom who is newly hired employee needs to be put in role QE1. In URA97 if this action was performed by SSO. First he has to assign ED to Tom (Step1) and then he can assign QE1 to Tom (Step 2). Now lets see what SSO needs to do

in URA02. He needs to have HR group or some other related group responsible for OS-U structure put Tom into @ED node (Step 1) and then he can assign Tom to QE1 (Step 2). As one can see the number of steps involved are same in URA97 and URA02.

We agree that in some cases there could be more steps in URA97 based on who is performing the assignment and that could add little administrative overhead. In the context of examples discussed in this chapter PSO1 and PSO2 will have to perform more steps than DSO or SSO. This is probably a correct behavior as PSO1, PSO2 are junior administrators. They have restricted capabilities, which forces them to perform multiple steps. We feel this issue can be addressed by having senior administrators perform assignments when an employee needs to be directly placed into a more senior role. Junior administrators could handle the cases where there is a natural progression from ED to E1 to QE1 etc.

Now lets take a look at redundant assignment issue. Tom who is a member of QE1 could also have received explicit assignments to E1 and ED1 depending on how Tom got the role QE1. These explicit assignments can actually be useful in terms of role usage. If we look this from a role activation point of view, having this explicit role assignment is useful in inheritance architectures. User Tom does not always have to activate his more powerful role QE1 to perform tasks that could be done by E1 or ED.

Finally we look at the restricted user-pools. The authors of URA02 point out that in URA97 to accommodate cases where only specific set of users need to be assigned to a role, it may necessitate changes to role hierarchy. URA97 uses prerequisite roles and conditions in making assignment decisions. For cases that do not fall into existing criteria, it may need some changes. URA02 tries to mitigate this by having organization structures, which are independent of role hierarchy. In principle URA02 just moved the problem from roles to organization structures.

URA02 brings up some key issues that could arise in URA97, however we feel it does not address them completely. It moves these issues to organization-structure and its maintenance. One of URA97 goals is to use RBAC to manage itself. URA02 does not follow this principle and introduces organization structures for RBAC administration. Later in

this thesis we will define a framework that goes beyond URA02 in simplifying user-role administration by allowing for self-service and automation.

Other user-role assignment related work and analysis include [31, 52–54].

Chapter 4: Push Model

Administration of roles in a distributed environment is more intricate when compared to a centralized environment. There are many factors that contribute to the complex nature of role administration in a distributed environment. For example all the systems may not support role hierarchies. Also, every role may not be present on every server. In this chapter we identify some possible architectures for distributed RBAC. We do not claim this to be a complete analysis but rather the beginning of a framework to approach practical issues in this arena. In particular we present a push model, which shows how user-role assignment can be accomplished in a distributed environment. We assume that all the necessary authorizations that are required to perform the operation of assigning and revoking users are done by a separate model, such as the URA97 or some other suitable model. Our assignment model is independent of the administrative authorization model.

We emphasize that the focus of this work is on user-role assignment. There are numerous distributed systems issues that need to be addressed in an actual implementation but their discussion is outside the scope of this work. We assume that a core of distributed services will be available and do not consider their detailed implementation. The central contribution of this work is to identify some architectures¹ possible for RBAC in distributed systems and to present a push-based model for user-role assignment. We classify the architectures based on event notification, system policies, system capabilities and role classification. It is not our goal to discuss all possible architectures but rather to take a first step in identifying interesting alternatives.

The organization of chapter is as follows. In section 4.1 We discuss some of the possible architectures in a distributed environment. We base our classification on distribution of

¹In PEI framework enforcement layer represents architecture layer, so these can be viewed as enforcement models in that context.

resources, data and users across the sites and how they interact among themselves. Push model is defined in section 4.2 followed by summary in section 4.3 and finally related work in section 4.4.

4.1 Classification of architectures

Broadly speaking a distributed system consists of data, software and users spread across several sites connected by some form of network. We can have several possible architectures depending on how the resources, data and users are distributed and how they interact. We have identified certain architectures and we will discuss them in this section. We reiterate that these architectures are only a subset of all the possibilities. An exhaustive analysis is beyond the scope of this work.

The administration of RBAC has essentially three components: user-role assignment (UA), permission-role assignment (PA) and role hierarchy (RH). Generally user-role assignment is a logically centralized service since the same enterprise role may occur on multiple servers. Thus assigning Alice to the Engineer role should give her access to all servers where the Engineer role is authorized. This should be transparent to the administrator who assigns Alice to the Engineer role. On the other hand permission-role assignment is typically done independently at each local system. For example, the human-resources server and the marketing-department server can independently determine what privileges are available to the Engineer role.

In this work we treat the role hierarchy as a centralized service. A role hierarchy is mathematically a partial order [8]. Senior roles inherit permissions from junior roles. Treatment of the role hierarchy as centralized imposes the same seniority relationship at each server. Thus if the Manager role is senior to Worker that relationship will hold on all servers in the system. Clearly there are cases where this will not apply. For example, Manager may be senior to Worker in most servers but there may be some servers where Workers are senior (say those servers run by the Workers union). For simplicity we limit our discussion here to the case where the role hierarchy is uniform at all servers. We do

allow individual servers to introduce additional role-role relationships between roles that are incomparable in the original hierarchy, provided the net remains acyclic.

4.1.1 Architectures from UA Viewpoint

Next we discuss the various architectures that are possible from UA point of view. Event notification and information can be sent to distributed servers in various ways. In general these can be placed in four categories namely push model, pull model, lookup model and credential verification model.

- **Push Architecture:** In push architecture the UA information is pushed to all the systems notifying the events. An example of push architecture is active channels on the Internet. They push the information to client sites.
- **Pull Architecture:** In pull architecture every system contacts the central repository. Each system maintains a cache where it stores this information for a certain period of time. The cache can be refreshed based on system requirements. An example of Pull Architecture is Web Browsers. The browsers keep a disk and memory cache and depending upon the settings of the browser they contact the URL for the information or fetch it from their local cache.
- **Lookup Architecture:** Lookup architecture is a special case of pull architecture that does not maintain any cache. A server contacts the central system for information as needed.
- **Credential based Architecture:** In credential based architecture the server is presented with credentials in a verifiable form. Some of the examples are Kerberos tickets, X.509 Certificates, secure cookies, etc.

In this work we take a detailed look at push architectures for user-role assignment. The reason for this is twofold. Legacy servers are built to stand-alone, so each maintains its own user account and user-role databases. To integrate legacy servers into distributed systems push architecture is a natural choice. The importance of legacy systems cannot

Table 4.1: Homogeneous versus Heterogeneous Role Hierarchy

Global Hierarchy	Restricted Roles	Classification
Non-empty	all roles	Homogeneous
Non-empty	some roles	Heterogeneous
Non-empty	none	Heterogeneous
Empty	none	Heterogeneous (federated architecture)

be overestimated as the Y2K problem has demonstrated. We postulate that even in the future, servers that maintain their own user-role database will be required. Such servers can function even if the network has failed or is congested, so they will be deployed where justified. Thus overall push architectures are of great practical interest.

4.1.2 Architectures from RH Viewpoint

In this classification the architectures are distinguished by degree of homogeneity of the RBAC policies of the distributed servers. If all servers use identical role hierarchy definitions we call it a homogeneous architecture otherwise we call it a heterogeneous architecture. Another factor we consider in the classification is degree of local autonomy. Local autonomy allows servers to introduce relationship between global roles without violating any enterprise rules, that is without introducing relationship among incomparable roles that are designated as restricted. In a homogeneous architecture there is no local autonomy. If we allow complete autonomy without any restrictions and rules then every local system will have its own hierarchy definition and this system can be classified as a federated architecture. The classifications are listed in table 4.1.

- **Homogeneous Architecture:** In a homogeneous Architecture all the systems have the same hierarchy definition, for example if role x is senior to role y then it is true in all the systems. The homogeneity of the role-role relationship is maintained across all the systems. This architecture allows us to maintain a single global hierarchy. Each of the local system may contain complete or partial set of this global hierarchy and

local systems can not introduce additional role-role relationships.

- **Heterogeneous Architecture:** In a heterogeneous architecture the local systems can introduce relationships between roles if they are not restricted. There will be a part of global hierarchy that can not be modified and the local systems can introduce relationships between the roles which do not fall into the restricted class.
- **Federated architecture:** In federated architecture the role-role relationship is not same across all the systems i.e. if role x is senior to role y in one system it may not be the case in another system. There is no single global hierarchy in this kind of architecture.

The push architecture for user-role assignment developed later in this chapter can apply to any of these architectures. Users are assigned to global roles consistent with the global role hierarchy. Extensions to the global hierarchy must be taken into account at each individual server and does not directly impact user assignment to global roles.

4.1.3 Architectures from System Capability Viewpoint

This classification is based on the capabilities of the system. Some systems offer support for role hierarchies whereas others do not provide such capability. If a system does not support role hierarchies, we can simulate a role hierarchy by explicit assignment of a user to all junior roles when that user is assigned to a senior role. Otherwise we can rely on the role hierarchy and limit our assignment to just the senior role. In this work we provide user assignment algorithms for both cases.

4.1.4 Architectures from User and Role Occurrence Viewpoint

There can be four cases of user role occurrence on end systems as listed in table 4.2. The cases where all the users are present or all the roles are present are not desirable. Having all the users on all systems violates the least privilege principle. If these users have no access to resources on that system they should not have an account on the system that enables them to login. Similarly having all roles in all the systems is not appropriate as we do not

Table 4.2: User role occurrence

Users	Roles
On all systems	On all systems
On some systems	On all systems
On all systems	On some systems
On some systems	On some systems

wish to have roles that are not relevant to a server. For example, an Engineer role may not be needed in a sales database. In this work we assume the case that every system does not have all the roles and all the users. By addressing this general case we clearly cover all the other cases also.

4.2 Push Model

In this section we describe the user-role assignment model based on push architecture. User-role assignment is done globally and the local administrators at the local systems do the permission-role assignment. As user-role assignment and revocation is performed this information is pushed to individual servers. The assumption is that each server has its own database of users, roles and user-role assignments which it uses to enforce RBAC. Since most existing systems provide this capability a push architecture is more suited to them. As future systems get developed they may move to pull, lookup or credential based architectures. Nevertheless push architectures will be needed for some time to come, if only to accommodate legacy systems.

We assume that system provides underlying infrastructure needed for distributed processing. We present some of the features we believe should be supported by the underlying system.

- Single Sign On: The system should have single sign on capability for ease of user authentication and access.
- Network security and data encryption: The system should provide network security

and data encryption to ensure confidentiality and integrity of messages and data passing over the network.

- Two Phase Commit: In order to achieve transaction semantics the system should be capable of providing two phase commit to ensure consistency.
- Location Transparency: The system should provide location transparency to end user.
- Global name resolution: There should be a name resolution service to map the names of roles and users across the systems in a consistent manner.

We have only identified the most important distributed system services here. As mentioned earlier our focus is on authorization issues and a complete discussion of distributed system issues per se is outside the scope of this work.

In our model we have a central service called the Role Authorization Center (RAC). The administrators of RAC perform user-role assignment. We define four relations which encode the relationship between the users, roles and the databases in the distributed system. These relations are maintained by the RAC and are shown in table 4.3. The *user_role* relation maintains all the relationship between users and roles and it consists of user name, the database name, the *actual_role* and the *assigned_role*. The *database_role* relation maintains information about roles that are present in each system. *The role_role* relation maintains the role hierarchy relationship among the roles. The *database_user* relation maintains information about users that are present on each system.

The model deals with two operations, assignment of users to roles and revocation of roles from users. We will be discussing the algorithms used to perform these tasks below. Figure 4.1 shows an example global hierarchy and figure 4.2 gives the hierarchies in the local systems. These are exactly the same as the global one except they are limited to roles present on each local system.

Table 4.3: Relations for Push Model

User	Database	actual_role	assigned_role
John Doe	Finance	CEO	CEO
John Doe	Personnel	CEO	CEO
John Doe	Services	CEO	CEO
John Doe	Engg Dept	CEO	CEO
Joe Black	Finance	CFO	CFO
Joe Black	Personnel	CFO	EMP
Alice	Engg Dept	Dir	Dir
Alice	Engg Dept	Dir	EMP
...

(a) User_Role relation

Database	Role
Finance	CFO
Finance	Acct1
Personnel	HR Dir
Engg Dept	Eng1
...	...

(b) Database_Role Relation

Grantor_Role	Grantee_Role
CEO	CFO
CEO	VP
CIO	Eng Dir
Sales Dir	Sales
Ed	EMP
...	...

(c) Role_Role Relation

Database	User
Engg Dept	John Doe
Engg Dept	Alice
Finance	John Doe
Finance	Joe Black
Personnel	Joe Black
...	...

(d) Database_User Relation

4.2.1 Assigning Users to Roles

When a user is assigned to a role we have to execute an assignment algorithm for each of the database servers where the role is present. We have two assignment algorithms (see table 4.4). The assignment algorithm 1 assigns the role to a user if the role is present in the local system. If the role is not present in the local system then we walk down the global hierarchy from the initial role as the starting point and build a list of all junior roles. After the list is built we take the senior-most incomparable roles in the list that are present in the local hierarchy and assign them to the user. In algorithm 2 when a role is assigned, we walk down the global hierarchy and build a list of all roles junior to the role. Then we assign the roles that are present in the local systems to users. This results in redundant assignment, which can be useful if the end system does not support a hierarchy.

We will exemplify how the algorithms work with a small example. Let say we want to assign role PL1 (Project Lead1) to Bob. The Finance department and service department don't have PL1 role or any roles that are junior to PL1, so no action is performed on these systems. We see that Personnel department has EMP role that is junior to PL1 and Engg department has roles PE1, QE1, Eng1, and ED that are junior to PL1. If we use Assignment algorithm 1 we will find that the role that needs to be assigned to Bob in Engg Dept is only Eng1. If we use algorithm 2 the roles that need to be assigned to Bob are Eng1 and ED. In the case of Personnel department Bob will be assigned EMP role by both algorithms. Whether we use algorithm 1 or algorithm 2 the permissions that are available to Bob are the same. If we use algorithm 1 Bob gets only Eng1 role in Engg department but because of role inheritance he will inherit the permissions of ED role.

Once the appropriate assign algorithm is executed and the role list is built we need to do following operations.

- If the user who is being assigned role(s) in a particular system does not exist in a local server then the user account needs to be created.
- The user role assignment should be performed at the local database servers.



Figure 4.1: Global Hierarchy

- The relations in the RAC should be updated to reflect the changes.

Next we prove that algorithm 1 correctly selects the senior most incomparable roles to be assigned to the user at each local database server.

Theorem 1. *Assign Algorithm 1 is correct.*

Proof: In order to assign only the senior most incomparable roles the while loop in the assign algorithm 1 has to maintain the following invariants.

Invariant 1. $r \in \text{candidate_set} \Rightarrow \text{actual_role} \geq r$

Invariant 2. $r \in \text{assign_roleset} \Rightarrow r \in \text{Database} \wedge \text{actual_role} \geq r$

Invariant 3. $r_1, r_2 \in \text{assign_roleset} \Rightarrow r_1 \neq r_2$

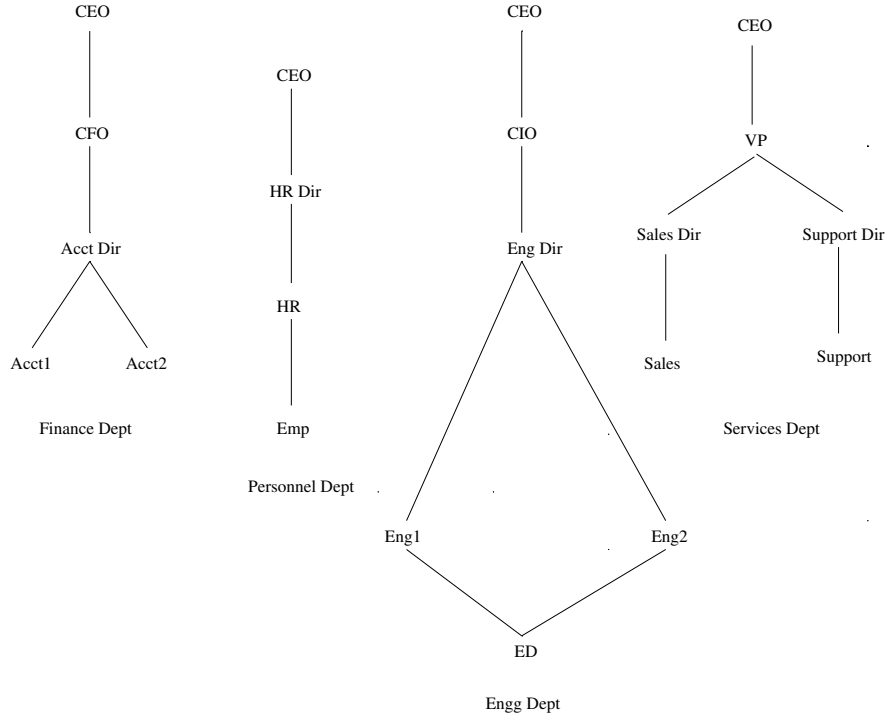


Figure 4.2: Local Hierarchies

Invariant 4. $r \in \text{assign_roleset} \Rightarrow \neg (\exists r' \in DB) \text{actual_role} > r' > r$

Invariant 5. $r_1, r_2 \in \text{candidate_set} \Rightarrow r_1 \neq r_2$

The proof of these invariants is by induction on the number of iterations of the while loop in algorithm 1. For the basis case consider the first iteration of the algorithm. The conditional else statement is satisfied and all the incomparable immediate children of the *actual_role* are placed in the *candidate_set*. Since the roles that are placed in the *candidate_set* are children of *actual_role* and incomparable, invariants 1 and 5 are satisfied. Other invariants are trivially satisfied as the *assign_roleset* is empty. Assume the invariants are true after K iterations and consider K+1 iteration. Invariants 1 and 5 will be true as the roles present *candidate_set* after the Kth iteration were immediate children of

Table 4.4: Assign Algorithms

<p>Procedure Mark(role) Take the role as the starting point and walk down the hierarchy and mark all nodes visited during the walk</p> <p>Procedure Unmark(role) Take the role as the starting point walk down the hierarchy and unmark all nodes visited during the walk</p> <p>Assign Algorithm1 <i>actual_role</i> \leftarrow Role to be assigned User \leftarrow User to which role is being assigned DB \leftarrow Database <i>assign_roleset</i> $\leftarrow \emptyset$ <i>candidate_set</i> $\leftarrow \{actual_role\}$ Unmark(<i>actual_role</i>) while <i>candidate_set</i> $\neq \emptyset$ do r \leftarrow get a member from <i>candidate_set</i> <i>candidate_set</i> \leftarrow <i>candidate_set</i> - {r} if r exists in DB then <i>assign_roleset</i> \leftarrow <i>assign_roleset</i> \cup {r} Mark{r} else <i>R'</i> \leftarrow Immediate Children of r <i>R''</i> \leftarrow Unmarked members of <i>R'</i> <i>R'''</i> $\leftarrow \{r \in R'' \mid \text{no senior of } r \text{ is in } candidate_set\}$ <i>candidate_set</i> \leftarrow <i>candidate_set</i> \cup <i>R'''</i> end do the assignment of roles in the <i>assign_roleset</i></p> <p>Assign Algorithm2 BEGIN <i>actual_role</i> \leftarrow Role to be assigned User \leftarrow User to which role is being assigned DB \leftarrow Database Mark(<i>actual_role</i>) <i>assign_roleset</i> \leftarrow all the nodes marked visited END do the assignment of the roles in the <i>assign_roleset</i></p>
--

actual_role and incomparable, so any roles that are present in the *candidate_set* are children of the *actual_role* and incomparable. The invariant 2 will hold true as the roles in the *candidate_set* are children of *actual_role* and because of conditional if statement in the algorithm. Since the roles in the *candidate_set* are incomparable the invariants 3 and 4 will be true.

Theorem 2. *Assign Algorithm 1 will terminate*

Proof: It is easy to see that the algorithm terminates when the *candidate_set* becomes empty. Consider the case when the *actual_role* is present. The algorithm will terminate after one iteration. Let us consider a case when the *actual_role* is not present. We can see that during every iteration of the loop the candidate set is decremented by one role. When ever the conditional else is satisfied the candidate set can increase by n number of roles. Since the role hierarchy is acyclic and number of roles is finite, due to invariant 5 the addition to *candidate_set* will stop after all the roles that are junior to the actual role are visited. After this point every iteration of loop will result in decrement of *candidate_set* by one role and eventually the *candidate_set* will become empty and the algorithm will terminate.

4.2.2 Revoking Roles from users

The Revoke operation does the revocation of a role from a user. The roles that need to be revoked from a user are determined by executing revoke algorithm. In revoke algorithm we search the *user_role* relation and build a list of all the tuples that contain the user and have the same *actual_role* value as the role that is being revoked. Once the list is built we delete the tuples from the *user_relation* that matches the tuples in the list. For every *assigned_role* in the list we see if the role was assigned to user by some other role assignment, if it is true then we delete those tuple(s) from the list. Once the algorithm is executed the following operations need to be performed.

- Databases in the revoke list should be notified of the roles that need to be revoked from the user.
- Each local system should perform the revocation of roles from users. After the revocation if the user doesn't have any roles in the local system then he should be dropped from the local system.
- The relations in the RAC should be updated to reflect the changes.

The revoke algorithm is independent of the assign algorithm that was used to assign the role to user. Each assign algorithm does the appropriate bookkeeping in the *user_relation*.

Table 4.5: Revoke Algorithm

<pre> Revoke Algorithm BEGIN <i>i</i> ← number of tuples in <i>user_role</i> relation <i>r</i> ← role to be revoked <i>u</i> ← user from which the role is being revoked while <i>i</i> ≠ 0 do <i>i</i> ← <i>i</i> - 1 if <i>user_role</i>(<i>i</i>){<i>actual_role</i>, <i>user</i>} = {<i>r</i>,<i>u</i>} then <i>revoke_set</i> ← <i>user_role</i>(<i>i</i>) <i>user_role</i> ← <i>user_role</i> - <i>user_role</i>(<i>i</i>) end <i>j</i> ← number of tuples in <i>revoke_set</i> while <i>j</i> ≠ 0 do <i>j</i> ← <i>j</i> - 1 <i>k</i> ← number of tuples in <i>user_role</i> relation while <i>k</i> ≠ 0 do <i>k</i> ← <i>k</i> - 1 if <i>revoke_set</i>(<i>j</i>){<i>assigned_role</i>, <i>database</i>, <i>user</i>} = <i>user_role</i>(<i>k</i>){<i>assigned_role</i>, <i>database</i>, <i>user</i>} then <i>revoke_set</i> ← <i>revoke_set</i> - <i>revoke_set</i>(<i>j</i>) end end do the revocation of roles in the <i>revoke_set</i>. END Revoke Algorithm </pre>
--

So we need only one revoke algorithm.

4.3 Summary

In this chapter we outlined some architectures possible in a distributed environment for RBAC administration and we developed a push-based model for user-role assignment. Our model is generic and it can accommodate different kinds of assignment structures. An authorization model can sit on top of our model to perform checks on the assignment authorization (for example, a model like URA97). We do not impose restrictions on the authorization model that needs to be used. A possible extension of the current work is to introduce a centralized control of permission assignment using packaged abilities. Each

administrator of the local system can package the permissions into abilities (permission only roles) and export them to the RAC. The administrators of the RAC are responsible for assigning these abilities to appropriate roles by means of a push architecture. Finally we emphasize that future systems may move towards a pull, lookup or credential-based architecture but the need for push architectures will remain due to the large number of deployed legacy systems, which inherently require a push architecture.

4.4 Related Work

Administration of RBAC in distributed environments is still a evolving topic and very few models try to address this issue directly. RBAC96 and NIST-ANSI RBAC standard do not address distributed RBAC administration directly. Deckker et al [55] present a model of administration of RBAC in distributed systems. Their model maintains a centralized administrative system and sends only relevant policy change updates to subsystems. Some other related work includes [56], [57].

Chapter 5: ASCAA Principles

Over the years numerous enhancements and extensions to RBAC96 and related models have been accomplished without having to change core concepts of the model. In the recent past new paradigms like usage control and rate limits have gained traction. Also accountability has recently received considerable attention driven by emerging requirements of secure information sharing and continued recognition of the insider threat. In order to accommodate these new ideas and to add features like automation and self assignment, we feel foundational principles of RBAC96 need to be expanded. We also believe that this expansion of foundational principles is critical for further advances in RBAC. In this chapter we offer five founding principles for next-generation RBAC, summarized as ASCAA for Abstraction, Separation, Containment, Automation and Accountability.

First we review the foundational principles of RBAC96 and then propose the new set of ASCAA principles for next-generation RBAC.

5.1 RBAC96 Principles

RBAC96 was motivated by four foundational principles, discussed below. RBAC96 does not actually “enforce” these principles, or require “conformance.” It is possible to completely ignore them and still technically do RBAC.

5.1.1 Abstraction

The abstraction principle refers to abstraction of permissions. In general, permissions and operations are system specific. For example operating systems typically support permissions such as read, write and execute. Database management system permissions could be select, delete, update, etc. While RBAC is useful in controlling permissions at these lower levels,

its real value lies in managing abstract application-oriented operations such as credit and debit operations on an account. Capturing and managing abstract operations allows for distinction of usage which are not possible if only system level permissions are managed by RBAC.

The constructive approach of RBAC accommodates the semantic distinction between credit and debit, familiar to anyone who has balanced a checkbook. The permissions for credit and debit can then be granted to different roles. Abstract permissions raise the level of policy consideration to application-level semantics, and accommodate real-world policies.¹ Operating systems and database management systems have provided mechanisms for implementing abstract permissions since the early 1970's by means of stored procedures or similar constructs, and these mechanisms have been widely used.

The abstraction principle remains unchanged in our proposed set of principles for next-generation RBAC.

5.1.2 Separation

Separation refers specifically to separation of administrative functions. This principle is not explicitly articulated in the original RBAC96 paper [8, 58], but it deserves recognition in retrospect because of its utility. One of the attractive aspects of RBAC is the separation of user-role assignment from permission-role assignment. These two tasks require different skills and are operationally distinct. Permission-role assignment requires deep knowledge of application semantics and security needs. This is best done by people who understand the application and the system that supports it. User-role assignment is a human resources and people management task which requires greater understanding of the human side and an appreciation of overall priorities which may need to be balanced with respect to individual decisions. Moreover, the deeper policy issues in managing the overall set of roles, the role hierarchy and constraints can be further separated into a business security organization.

¹The aggregation of multiple permissions into a role is itself a significant abstraction benefit of RBAC. The abstraction principle as such is focused on what are atomic permissions from an application perspective.

This organization can focus on the more important policy issues around RBAC while devolving day-to-day operational details to appropriate business and information technology people.

The separation principle also remains unchanged in our proposed set of principles for next-generation RBAC.

5.1.3 Least Privilege

Least privilege is a long-standing tenet of access control. RBAC supports it by having each role assigned with permissions appropriate to the business function of the role. Taken to its extreme least privilege is likely to result in unmanageable proliferation of roles so there is need for judicious balance. By letting the role designer determine the degree of role fragmentation versus least privilege RBAC provides support for achieving this balance.

The least privilege principle is subsumed by the more general containment principle in our proposed set of principles for next-generation RBAC.

5.1.4 Separation of Duty

Separation of duty has been a driving principle for RBAC. The fact that roles may be conflicting has been a long standing practice in commerce since ancient times. Modern business practices continue to build on these long-proven insights. A common example is the conflict between purchasing manager and accounts payable manager roles. A single person with both roles would have sufficient power to single-handedly commit fraud, hence the conflict. This requirement is commonly called static separation of duties. A more nuanced conflict exists between a cash register manager and cash register clerk. A single individual could legitimately take on both roles, but not at the same time on the same register. This is called dynamic separation of duties. A common example of separation of duties from the military sector is the use of two-person, or more generally n-person, rules which require two, or more, people to authorize critical actions such as launch of weapons of mass destruction.

The separation of duty principle is also subsumed by the more general containment principle in our proposed set of principles for next-generation RBAC.

5.2 ASCAA Principles

We propose the following five principles for next-generation RBAC. We refer to these as the ASCAA principles for Abstraction, Separation, Containment, Automation and Accountability.

5.2.1 Abstraction

The abstraction principle is essentially unchanged from RBAC96 and refers to abstraction of permissions.

5.2.2 Separation

The separation principle is also essentially unchanged from RBAC96 and refers to separation of administrative function.

5.2.3 Containment

The containment principle unifies the older principles of least privilege and separation of duty, and further incorporates additional constraints and usage control elements. The concept of containment seeks to limit the damage that a user, or a set of users, can perpetrate either by deliberate malice or by victimization from malicious malware. The individual techniques such as least privilege, separation of duty, cardinality constraints [8, 25] and usage limits [59, 60] are means to this end. They should be viewed as applicable mechanisms rather than motivating principles.

Least privilege and separation of duty have been discussed in the previous section and are familiar in the access control literature. There is not much more for us to say about these here. RBAC96 introduced an abstract open-ended notion of constraints to accommodate separation of duties but not be limited only to this specific mechanism. Containment

accommodates these previously published aspects of current-generation RBAC, beyond least privilege and separation of duty.

Looking to next-generation RBAC we believe it is important to incorporate usage and rate limit concepts from recent models for usage control [59]. Usage limits occur in various forms. We can restrict the number of times that a particular permission or role can be exercised. These limits may be absolute so the usage quota runs out at a certain point and requires replenishing by some means for further access. This case is more appropriate for digital rights management where access is purchased by some exchange of value, such as money. Rate limits control the number of times a particular permission or role can be exercised in a specified period of time. For example, many ATM networks limit a user to, say, three withdrawals per day. This limits loss due to misuse of an ATM card. Rate limits are particularly relevant to RBAC. For example, a customer service representative (CSR) can be limited to access a certain number of customer records commensurate with the anticipated workload. This contains the damage from a malicious CSR who is fishing for customer data. Perhaps more importantly the rate limit also contains the damage by malware. By restricting the rate to be within the rate reasonable for a human to consume the information, the human initiated activity is not disrupted but access at machine speeds by malware is cut off. These applications have been motivated in the usage control literature but are also relevant to RBAC, and should be supported in next-generation RBAC.

5.2.4 Automation

We believe that automation of access control administration is inevitable in next-generation access control simply to keep pace with scalability requirements of cyberspace. Assignment and revocation have traditionally been viewed as administrative actions requiring intervention by human administrators.² Current-generation RBAC offers substantial benefits in this regard by aggregating permissions into roles which can then be assigned in a single step instead of requiring multiple assignments. Moreover, additions and deletions of permissions

²The demand operation for acquiring privileges in the schematic protection model [61,62] and automated expiry of role delegation in the role-based delegation models of [29] are some exceptions to this tradition.

to and from a role have immediate effect with respect to role members. To this degree RBAC inherently supports automation by aggregation. We propose to extend automation to a much deeper level in next-generation RBAC.

The failure to remove privileges after they are no longer needed continues to be a major source of security problems. Privileges are too often left intact when users leave or are reassigned to different jobs within an enterprise. These unused and unnecessary residual permissions become an attractive pathway for attackers. The role-based delegation models of [29] incorporate a time limit on temporary delegations so that the delegation will expire by the specified time. We believe such limits should be embedded into other permission granting operations of RBAC. We feel this is particularly so for user-role assignment. In general we expect permission-role assignment to be fairly stable and automated revocation is less likely to be useful in this context. User-role assignment on the other hand is likely to change more rapidly over time. We can base automatic revocation in this context on a fixed time period, such as one year, or on time elapsed since last use or similar considerations. Recovery from revocation in this context can be based on explicit human intervention by an appropriate administrator to restore the revoked role. But automation can be used in this context too. We can allow the user to renew role-membership on their own after the proscribed period has expired. Thus users who need the role can keep it alive whereas those who do not can let it expire.³

Another aspect of automated revocation is cascading revocation. A single revocation action can result in multiple revocations as conditions for maintaining the permissions change. Automation of cascading revokes is relevant to RBAC but it should be interpreted quite differently from its usual interpretation in DAC.⁴

We also propose to apply automation to role assignment. One of the recognized benefits of RBAC is simplicity of administration. However, as roles proliferate the administration

³Anticipating the accountability principle we could require additional authentication to effect the renewal step so it can be attributed to the user rather than malware that simply keeps all the users roles alive.

⁴In chapter 3 we make the distinction between how RBAC treatment of revocation is different from DAC and we suggest a notion of strong versus weak revocation. Weak revocation does not require cascading revoke whereas strong does

burden grows. Several models for decentralized user-role assignment have been proposed [15–17]. Figures 3.1 and 3.2 respectively show a role-hierarchy and an administrative role hierarchy. Junior administrators such as PSO1 and PSO2 are limited in the roles that they can assign and in the users to whom they can assign these roles. The URA97 model provides the notion or prerequisite conditions for users to be eligible for assignment, while [16,17] argue for using organizational structure rather than roles for this purpose. Nonetheless granting and revocation of roles in these models requires explicit administrator action. Imagine these two figures expanded to have hundreds or thousands of projects, with engineers working on dozens of projects at any time with frequent re-assignment to different projects. Administration entirely by human intervention becomes impractical on such a scale. Instead, we could authorize users to take on role membership by self-assignment in a limited number of projects on their own, while restricting them from simultaneous membership in too many projects and too rapid a rate of change of projects. The ability to do this would be predicated on membership in basic roles such as ED or E. This would allow flexibility in which projects engineers can work on and can explore while limiting the damage due to malicious activity of users or malware. We could insist that higher level roles such as PE1 and QE1 require explicit administrator assignment or approval.

Another application of self-assignment can arise in a professional or social community context. Initial membership in the community could be made available on a purely self-assigned basis with minimal, if any, verification of self-asserted user attributes. Self-promotion of membership up to a certain level could be permitted with the understanding that taking on more senior roles implies obligation to participate in community activities, such as reviewing and mentoring in a professional society. The highest levels of membership could require an approval process by appropriate peers. Thus endorsement by a specified number of advanced members would be required for assignment of a user to advanced member status.

We discuss some of the models for self-assignment in the next chapter. In general we expect that more junior roles would be available for self-assignment with usage and rate

limits, and that more senior roles would require human approvals. We also anticipate the need to allow self-assignment of senior roles on a temporary basis in case the additional privileges are required in extreme situations. Thus a user with PE1 privileges may temporarily escalate to PL1 with appropriate accountability provisions, which brings us to our next principle of accountability.

5.2.5 Accountability

Accountability has recently received considerable attention driven by emerging requirements of secure information sharing and continued recognition of the insider threat. We offer the paradigm of adjustment as a means to achieve accountability. Adjustment acknowledges that not all authorized actions are identical. Sensitive operations require an enhanced level of auditing, notification or authentication. For example, it is common place for websites to require additional authentication and notification for sensitive operations such as change of address.

The primary goal of accountability is to make a human user take responsibility for actions that the individual performs in a system. This can be achieved in a combination of three basic ways. Sensitive operations can be subjected to a more detailed level of auditing but unless the audit records are brought to some other user's attention the audit trail is useful only as a forensic tool. Detailed audit trails can trigger fraud detection systems to direct their attention to suspicious activity but ultimately some user has to be alerted. Notification is a more direct approach to explicitly require sensitive operations to trigger a message to an appropriate user. Thus temporary self-assignment of the PL1 role by a PE1 user should trigger a message to all PL1 users alerting them to the circumstance. This will inhibit inappropriate use of this escalation privilege. Finally it is important to escalate the authentication required for sensitive operations. Thus a re-authentication may be required when a particularly large transaction is attempted. The re-authentication may fail if a human or malware attacker is attempting the operation and is unable to produce the necessary credentials on demand. Instead of or in addition to a re-authentication we

may require an alternate authentication based on credentials other than used for the earlier authentication.

In the context of RBAC we can measure sensitivity of operations with respect to the application semantics such as monetary amount of a transaction. We can also measure sensitivity with respect to roles and various stages of role assignment, activation, usage and release.

5.3 Summary

In this chapter we have proposed a new set of five principles called ASCAA for next-generation RBAC comprising Abstraction, Separation, Containment, Automation and Accountability. We believe these principles are applicable to access control in general, but our immediate focus has been on RBAC. Two of these, abstraction (i.e., abstraction of permissions) and separation (i.e., separation of administrative functions), are essentially unchanged from the RBAC96 principles. The third, containment, generalizes and unifies the two older principles of least privilege and separation of duties to include additional constraints and usage control elements. The final two, automation and accountability, are newly recognized. Automation applies to revocation and to the granting of roles and permissions. The accountability principle is illustrated in this work with respect to authentication adjustment, enhanced auditing and targeted notification.

It is our belief that next-generation RBAC should be based on this expanded set of principles so as to address real-world protection needs of next-generation systems.

Chapter 6: SSRBAC08 Model

We believe automation of administration will be inevitable and be a key factor in success of RBAC to keep pace with scalability requirements of cyberspace. Although current RBAC administrative models allow for decentralization they still require act of an administrator and need human intervention. Failure to remove privileges after they are no longer needed remains a key problem and models like URA97 do not address this issue. It is left to the role administrators to revoke roles from users. Some types of automated role revocations is addressed by role based delegation models [29] which impose time restrictions on temporary delegations. RBAC models should allow and support such limits inherently.

One way of achieving automation in RBAC and making it more agile is by allowing for some sort of self service in the aspect of user-role assignment. Enterprises with large number of employees typically have large numbers of users with different access requirements spread across several systems. The dynamic nature of changing user access needs necessitate an efficient user provisioning, activation and deactivation of system access control mechanisms. Static administered models like URA97 are inefficient and can increase administrative burden. Allowing self assignment does not mean unlimited assignment for unlimited time, the role assignment is performed and limited so as to allow user to perform his job functions. Typically users will be proactive in requesting and assigning the needed roles to do their job, however they may not be disciplined to relinquish unwanted roles, If not controlled self assignment will lead to role aggregation. In order to mitigate role aggregation, features like cardinality limits, revocation based on time limits, usage rates and audit reviews should be supported by the system.

Another key aspect that needs to be addressed by self service based models is usage of privileges. Simplifying user-role assignment by means of self assignment and request based assignments require some sort of monitoring of rate at which these actions are being

performed and approved. Looking to self service based RBAC we believe it is important to incorporate usage and rate limit concepts from recent models for usage control [59]. The rate at which automated actions are performed should be monitored to make sure they are at human speed and within tolerable levels. Any deviation from the accepted normal levels needs to be checked to limit the damage due to malicious activity of users or malware.

We feel self service models also need to support administrator initiated assignments and actions. In general we expect that more junior roles would be available for self-assignment with usage and rate limits, and that more senior roles would require human approvals and administrative actions.

In this chapter we modify RBAC96 model and propose a self service RBAC framework called SSRBAC08. We narrow our discussion to user-role assignment aspect and concentrate on developing the model to support a self service based assignment based on ASCAA principles. Later we outline an example user-role assignment model called SSURA08 based on SSRBAC08. The principal contribution of SSRBAC08 is to lay the groundwork for developing self service based models.

6.1 The SSRBAC08 Framework

In this section we outline a basic framework for self service RBAC called SSRBAC08, which is slightly modified version of RBAC96. One of our primary design goals is to have a model based on ASCAA principles. Also in this framework we narrow our focus towards user-role relationship, as we believe this relationship is one that can benefit most from automation. We assume permission-role assignment will remain unchanged from RBAC96.

Figure 6.1 shows the SSRBAC08 relationships and core characteristics. We left out permission-role relationship in this figure for brevity purposes and leave it unchanged from RBAC96 model (see figure 2.1). SSRBAC08 is similar to RBAC96 however there are two key differences. The first difference is how the constraints are classified and categorized. In RBAC96 constraints are very broad and have an open-ended notion, we propose to categorize constraints into four categories namely assignment criteria, usage criteria, revocation

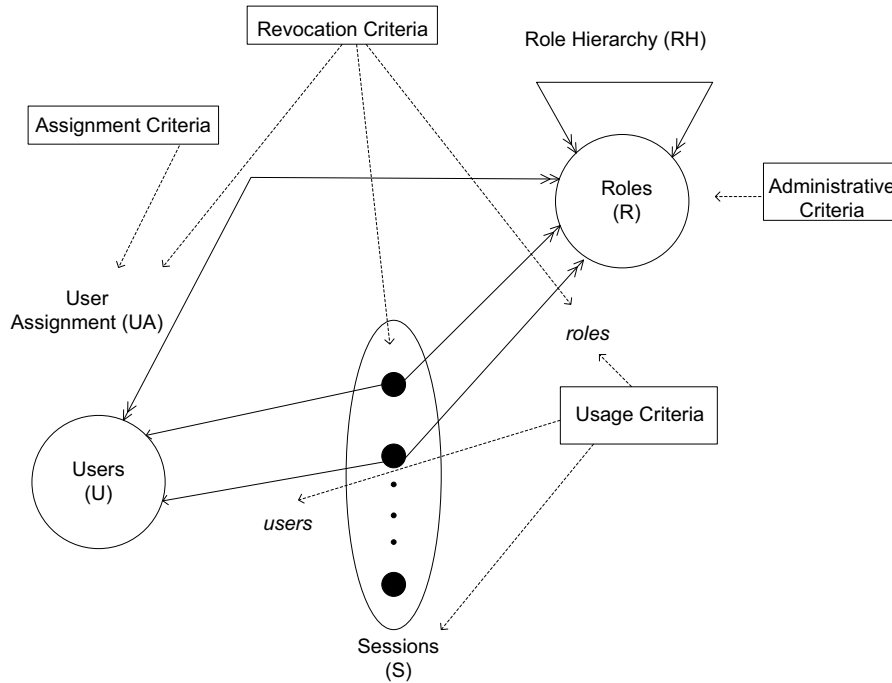


Figure 6.1: SSRBAC08 components

criteria and administration criteria. Second key difference is regarding role attributes. In RBAC96 the concept of role attributes is not stated explicitly, however when one creates role named “emp”, the name can be viewed as attribute of role holding the value “emp”. We propose to state this explicitly in SSRBAC08 and extend to add few more attributes in addition to name. Role attributes are used to associate with constraint categories.

Constraints are powerful means to layout higher level organizational policies. For example, Osborn et al. [63] show how to enforce mandatory and discretionary access control policies in RBAC using constraints. Gail and Sandhu [25] discuss constraints in the context of separation of duties, importance of constraints to support applications is discussed by Jaeger [64] and other notable work includes [23,24]. Constraints allow for decentralization and delegation of user-role assignment without having to worry about compromising the policies. In circumstances where RBAC is managed by a single trusted source one can argue that having constraints is more of convenience rather than a need. As a person performing

management is trusted to interpret and apply organizational policies appropriately. However in self service centric models it is not reasonable to assume that RBAC management will be centralized under one person. Constraints become a necessity if one has to apply and enforce organizational policies uniformly across the board. Constraints can be applied to user-role, role-permission and role-role relationships. We focus on constraints as they apply to user-role relationship and classify them into following categories.

- Assignment Criteria

Assignment Criteria (AC) defines the constraints that need to be satisfied during user-role assignment, these are evaluated when a user is assigned to role and must be satisfied before assignment can take place.

- Usage Criteria

Usage Criteria (UC) defines the constraints that need to be satisfied during role usage. These criteria are evaluated during the actual role usage and must be satisfied in order for them to be used in a session.

- Revocation Criteria

Revocation Criteria (RC) defines the constraints that will be checked to make revocation of role from user decisions. These criteria are more geared towards automation of revocation.

- Administration Criteria

Administrative Criteria (ADC) defines the constraints that specify how and who can manage the administrative functions of a role.

We define role attributes to hold values for characteristics of a role. For example, the name of a role. Although RBAC96 doesn't explicitly state attributes, name of a role can be viewed as implicitly defined role attribute. We extend this in our SSRBAC08 model and introduce four additional attributes called assignment, revocation, usage and administration. Assignment attribute holds the AC, similarly usage holds UC, revocation holds RC

and administration holds ADC.

The following definition summarizes the above discussion.

Definition 11. The SSRBAC08 model has the following components:

- U is a set of users,
- R is a set of roles,
- P is a set of permissions,
- $UA \subseteq U \times R$, is a many-to-many user to role assignment relation,
- $PA \subseteq P \times R$, is a many-to-many permission to role assignment relation,
- $RH \subseteq R \times R$ is partially ordered role hierarchy (written as \geq in infix notation),
- S is a set of sessions,
- $user : S \rightarrow U$, is a function mapping each session s_i to the single user $user(s_i)$ and is constant for the session's lifetime,
- $roles : S \rightarrow 2^R$ is a function mapping each session s_i to a set of roles $roles(s_i) \subseteq \{r \mid (\exists r' \geq r)[(user(s_i), r') \in UA]\}$ (which can change within a single session) so that session s_i has the permissions $\cup_{r \in roles(s_i)} \{p \mid (\exists r'' \leq r)[(p, r'') \in PA]\}$,
- assignment criteria is a collection of constraints controlling user to role assignment,
- usage criteria is a collection of constraints stipulating which values of the various components enumerated above are allowed or forbidden,
- revocation criteria is a collection of constraints that determine when a user-role assignment needs to be revoked.
- administration criteria is a collection of constraints that control the role administration functions like who can assign the role, who can control role relationship, who controls permission assignment etc.

- *rolename* this attribute contains name of the role,
- *assignment* this attribute holds assignment criteria,
- *usage* this attribute holds usage criteria,
- *revocation* this attribute holds revocation criteria and
- *administration* this attribute holds values of users and or roles that have administrative rights on the role. □

6.2 Criteria details

We now discuss each of these criteria in more detail and provide some examples to aid in understanding utility of these. We understand that one needs a policy language for expressing these criteria so that they are machine-readable. Our goal in this work is to illustrate the model. We believe XACML or other policy languages can be utilized to capture these criteria.

A key element that is present in each of the criteria is abstract notion of approval. In our discussion approval typically involves human interaction, which specifies whether a particular action should be allowed, or not. By means of approval we can bring in human element to intervene and provide exceptions to system limits or rules. Approval may also be required to perform certain assignments, revocations and usage. We envision approvals to be achieved via lightweight notification and workflow mechanism, which capture actions and results.

We also introduce notion of rate limits to impose restrictions on number of assignments and revocations that can be performed by a user or number of assignments that a user can receive in a stipulated period of time etc. Our goal is to allow for automation and self service to the extent possible and we want to make sure this is happening at human speed. Although it may not be possible to entirely eliminate rogue elements from performing actions with malicious intent, we can restrict such activity by imposing rate limits on the

assignment and revocation operations. Auditing and alerting mechanisms can be put in place to track such violations.

Assignment criteria define what an assignee should possess and requirements that need to be satisfied before the role can be assigned. Simply put these are the qualifications needed to acquire the role. The criteria can include membership or non-membership in other roles, obligations on part of assignee, approval from persons who have decision making authority and rate limits etc. In our discussion typically a prerequisite condition holds requirement of membership or non-membership in other roles on the part of assignee. Obligations hold assignee obligations. Approval holds requirements on when a approval is needed and who can do it. There could be several other conditions based on the organization policy needs and they can be incorporated as needed. It is not our intent nor we can provide a exhaustive list of all possible conditions that should be part of assignment criteria. Assignment criteria provides the means to list these conditions, obviously the list needs to be finite and just enough to capture the organizational policy. Also their needs to be a rule that basically spells out what combination of criteria needs to be met to consider AC as satisfied. It is possible to have assignment criteria to be null or not have any conditions there by allowing assignment without any checks on assignee requirements. We list few example assignment criteria below.

- Location or Geographical conditions
- Age restrictions
- Experience
- Static separation of duties
- Temporal restrictions
- Obligations
- Prerequisite roles and conditions

- Rate limits
- Cardinality restrictions

There could be situations where enforcement of some of the conditions needs to be overridden. For example lets assume in a hospital they have a assignment policy that states for a person to assume ER-Doctor role he needs to have completed atleast two years of residency. There could be times when this rule needs to be relaxed due to surge in ER cases or shortage of doctors due to vacations, case overloads etc. As a result one may need residents with less than two years experience take on ER-Doctor role. There are few ways one can address this issue. Remove the assignment rule which prohibits residents with less than 2 years experience to assume the ER-Doctor role or have some kind of overriding rule which will allow for such assignment to take place or do nothing. We believe such situations can be handled more elegantly by bringing human element into picture by means of having an approval to override the rule. We can state an assignment condition that says “Chief Doctor approval needed to override any ER-Doctor assignment rules”. Having approvals to override conditions allows one to relax the assignment criteria on a need basis without having to change the long-term policy, and since humans perform such approvals, we can audit the activity and make individuals performing actions accountable.

Administrative criteria define the requirements that need to be satisfied before assignment or revocation of roles can take place. Typically these criteria impose conditions that an assigner needs to meet before assignment or revocation can be allowed and also more general conditions that apply beyond a individual or group of assigners. Some examples of these include persons allowed to perform grants/revokes, whether self assignment is allowed, rate limits, cardinality conditions to specify maximum number of users allowed for the role, obligations that assigner needs to meet, approvals that are needed before assignment can be performed. For a role assignment to be successful it needs to satisfy the role assignment policy which is derived by combining the administrative criteria that are relevant to assignment and the assignment criteria of the role.

Usage criteria specify the conditions that need to be satisfied before user can activate

a role and use it. These checks could include mutual exclusivity of role with other roles, the temporal limits on when a role can be used, environmental conditions to limit users ability to activate a role based on his location, obligations that user needs to satisfy before using the role and approvals needed before activation etc. The main goal of usage criteria is support containment principles.

There could be cases in which actual usage of a privilege can be prevented even when the usage criteria are satisfied to activate the role. Say Alice has role called “Manager” which authorizes her to assign role “Engineer” to users, however she may not be able to use her privilege if assignment policies are violated (say limit on number of users allowed for the role has reached its allowed maximum). She was prevented from using her privilege not because she has not met the usage criteria of the role but due to the fact that her action was not supported by the assignment policy of the role she was trying to assign. Activation of role is intent to use it. In our criteria we check if such intent is allowed or not and then let activation happen. The ability to use the role’s privileges may be limited due to conditions that are independent of the activation. Activation of role is a pre authorization step and ability to use privilege is determined by ongoing authorization mechanisms.

Also we can have situations when either AC, ADC, or UC can enforce a organizational policy. We exemplify one such instance. In a self assignment scenarios the rate at which an individual performs role assignment should be at human speed and within allowable guidelines set by the organization. To control rate at which a user performs self assignments we could place an administrative condition saying “self assignment allowed if individual’s rate limit not exceeded”. Although administrative criteria are typically for the role that is being assigned, in this scenario we are checking the users prior actions to determine if assignment should be allowed. Alternatively self assignment could be made a privilege and assigned to users. Users have to activate it before self assignment can be requested. In this scenario for role assignment to be successful, administrative criteria of requested role should allow for self assignment, and rate limit can be enforced by usage condition which says “self assignment privilege can not be used if rate limits are exceeded”.

Revocation criteria typically specify the conditions that trigger automatic revocation. These include time limits, inactivity period, approval needed to revoke etc. Automation of revocation is key aspect of SSRBAC08, user-role assignment is bound to change over time as user needs and job functions change. If needed recovery from automatic revocation can be achieved via human intervention by appropriate administrator to restore the revoked role. Administrators can revoke roles from users any time as needed. In principle, administrator driven revoke follows same philosophy we outlined in chapter 3. As needed rate limits can be applied to administrator driven revocations. Self revokes can be allowed and they can have restrictions. When a user takes on a role it could imply that he has agreed to perform certain duties and to excuse himself from such responsibility without just cause may not be allowed. For example, when a student enrolls in a class he can not drop from the class after certain time without getting permission from the dean. Although this situation may not involve any roles, similar situations can occur in RBAC context also. Administration criteria combined with revocation criteria determine the role revocation policy.

6.2.1 Examples

Let us look at Figure 6.2 this expands figure 3.1 to include sales, support and HR department roles. Since SSRBAC08 does not have any administrative¹ roles, there is no accompanying administrative role hierarchy.

Now say only members of HR manager role can add users to EMP role. This rule becomes part of administrative criteria of EMP role and is evaluated every time before EMP role is assigned to any person. Let us assume that every user typically performs duties in his departments but will need additional roles from other departments to perform his job functions. Say Joe and Frank are consultants and they are part of sales department. As part of their job Joe and Frank perform product demos, help customers in implementations

¹In URA97 there is a distinction between administrative roles and regular roles. Administration of user-role assignment is controlled by administrative roles. The assignment decisions are based on aspects like prerequisite conditions and administrative role ranges. Similarly revocation is based on administrative role ranges. In SSRBAC08 we don't distinguish between administration and regular roles, the administrative rules and criteria are encapsulated in administrative criteria and the administration attribute of the role holds these criteria.

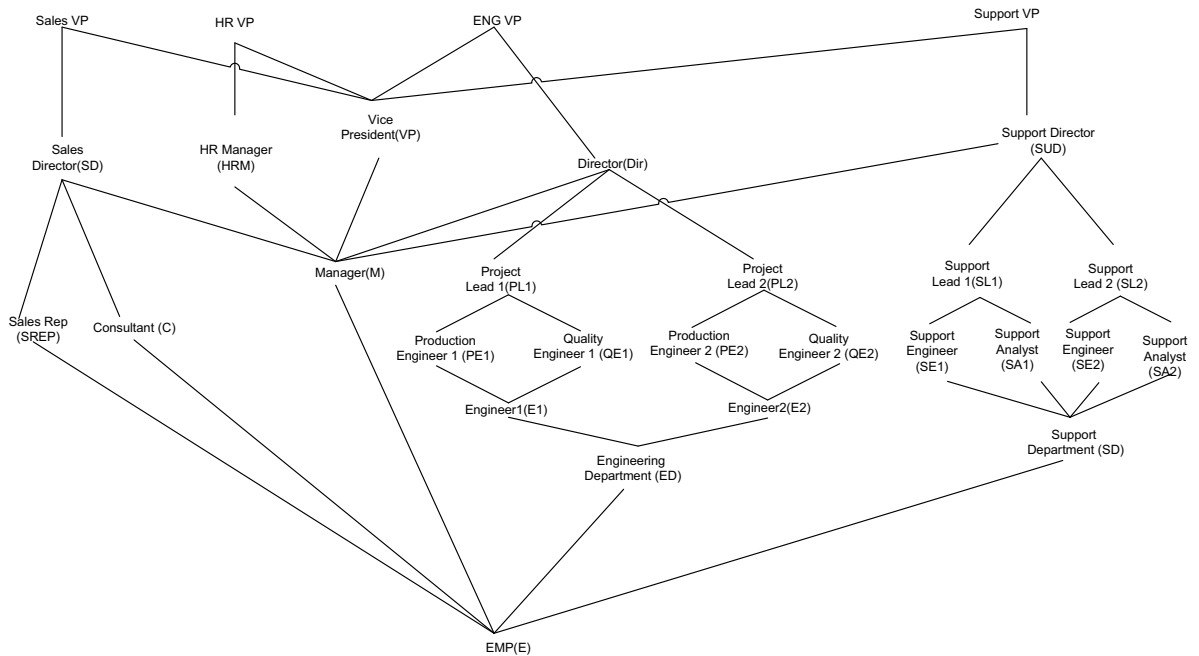


Figure 6.2: An Example Role Hierarchy 2

and troubleshoot issues, to successfully perform these they need roles from engineering and support from time to time. Policies can be framed to allow Joe and Frank who are consultants to get roles from engineering and support departments if they satisfy the assignment criteria. Any such assignment should be under reasonable rate limits and any deviation from such limits may need human intervention. How long and under what circumstances Joe can use these engineering or support roles will be stipulated by usage and revocation criteria.

For sake of illustration we define example AC,UC,RC,ADC for some of the roles in figure 6.2. We will use these example conditions to discuss the assignment, usage, revocation and administrative criteria. As we pointed out earlier the conditions are laid out in English and a policy language will be needed to make them machine readable, some simple conditions like prerequisite conditions can be evaluated using boolean expressions.

Example 1: We now look at some example criteria that are placed on role E1 and walk through the scenarios of assignment, usage and revocation under these criteria.

Lets us say the ADC and AC of E1 role have following conditions to control the assignment.

ADC conditions for E1:

1. Users who have consultant role can perform self assignment.
2. Members with PL1 or higher role can assign E1 role to others.
3. No more than 2 assignments in a day can be made by members of roles junior to Eng VP role.
4. No more than 20 assignments of this role per week.
5. There is a cardinality limit of 30 members for users with consultant role.
6. To override cardinality rule an approval from person with Eng VP role is needed.
7. Members with Dir or higher can perform revocation.

AC Conditions for E1:

1. Assignee should satisfy the prerequisite condition of having role EMP and not belong to E2.
2. Approval of Dir or higher needed to override prerequisite condition.
3. Assignee should not have exceeded 5 self assignment operations in the current month.
4. Obligation to complete mandatory system training 5 days from assignment.

In the context of these conditions lets see how assignments of role E1 can take place. Say Joe who is a member of consultant role needs to have role E1. Since Joe is a member of consultant role he can assign himself role E1, however he has to satisfy other ADC and AC criteria also. He can self assign without any additional approval's if he meets ADC conditions 3,4 and AC conditions 1,3. He needs to get appropriate approval if either ADC 4

or AC 1 is not satisfied. In this example we have allowed overriding approvals for prerequisite condition and cardinality conditions but not for rate limits.

Role E1 can also be assigned to users by person with PL1 or higher role and the assignment needs to satisfy ADC and AC. Lets say Jack who has Eng VP role is granting role E1 to Alice, in this case as long as rate limit conditions (condition 3 of ADC and AC) are satisfied the assignment will go through as Jack has overriding approval authority for cardinality and prerequisite conditions. Now one can make the overriding approval stricter by saying that the person assigning the role can not be a overriding approver. This will make Jack get approval from another person before he can perform the assignment.

Also in this example we placed rate limit conditions (conditions 3, 4 of ADC and 3 of AC) to show how one can control the rate of assignments. There is also an obligation on part of the person receiving E1 role to complete mandatory system training within 5 days of assignment.

Now we will look at usage criteria of role E1 that says.

1. Cannot activate if E2 is active.
2. Extranet activation allowed if role C is active.
3. Activation of role allowed from extranet only during 9AM-5PM Mon-Fri.

These criteria specify that activation of role E1 is allowed as long as E2 is not active (dynamic separation of duties). Also there is an environmental condition which allows for users with consultant role to activate E1 from extranet. There are also temporal limits when the role is activated in an extranet environment. So Joe who is a member of consultant role can activate role E1 from extranet during hours 9-5 from Monday to Friday as long as he does not have E2 active.

Finally we look at revocation criteria for role E1.

1. Revoke if assignment obligations are not met.
2. Revoke 1 year from assignment date.

3. Need extension approval from Dir or above to override 1 year rule.
4. Revocation if no activation in 3 months period.
5. Self revoke requires Dir approval.

Role E1 is automatically revoked if assignment obligation is not met which required the assignee to complete mandatory system training in 5 days from assignment. Condition 2 specifies the time limit on assignment and this can be overruled by having extension approval in place prior to revoke. Revocation is also performed when there is no activity for three consecutive months. Self revoke operations require approval and members of Dir or higher role can perform revoke any time.

Example 2: Lets say members of support department perform product maintenance fixes and customer assistance tasks. People with SE1 role perform maintenance fixes and are internal facing working with engineering team. Support analysts provide approved fixes to customers. In this context we show some example criteria for SE1 role. These criteria are listed in table 6.1 and we will discuss some of these criteria, which are slightly different from the previous discussion. We can see that self assignment is allowed by satisfying a prerequisite condition. These conditions place very strict control on consultants for getting this role, both assignee and assignor need approvals and assignee needs to fulfill obligation. As consultant is an external customer facing entity, support department wants to place additional guards before allowing consultant to get a support role. Also if we look at Revocation it says revoke if there is an approval, this translates to indefinite assignment until there is a approval for automatic revocation and non support personnel can have it for 2 months.

6.2.2 Conformance to Design Principles

We outline below how our model conforms to ASCAA principles.

- Abstraction

Table 6.1: Example Criteria for SE1

ADC:

1. Self assignment allowed if user has SD role and does not have C role.
2. Members of SUD can assign to users.
3. Support VP approval needed for people with consultant role.

AC:

1. Should not have SA1 if assignee is consultant.
2. Sales VP needs to approve if assignee is consultant.
3. Obligation to sign non-disclosure if assignee has C role.

UC:

1. Activate only if SA1 or PE1 not active.
2. Obligation of mandatory support training needs to be met before activation.
3. Activation is allowed from Intranet only.

RC:

1. Remove if there is an approval for SUD.
2. Expiration 2 months from assignment for non support personnel.

The abstraction principle typically refers to abstraction of permissions. Our framework does not detail permission assignment. However we have shown how abstraction of constraints can be performed.

- Separation

Separation refers specifically to separation of administrative functions.

In our model user-role and permission-role administrative functions can be easily separated, as they need different skill sets and are operationally different. This separation can be achieved by specifying the appropriate administrative rights values for the administration attribute. Further deeper policy issues in managing role hierarchy and the constraints can also be easily supported by same attribute. For example we can have administrative attributes of role “PE2” to hold a value which says Alice, Bob or anyone who has PL1 role can perform user-role assignment, Charlie or Dorothy who are business users can perform permission assignment and Eve or Frank who are system architects can manage constraints etc. These values can be evaluated against user session and a decision can be made on whether particular administrative function

is allowed or not.

Another aspect of separation in this framework is management of role criteria. Management of assignment, administration, usage and revoke criteria can be separated and assigned to appropriate groups of individuals to specify, alter and maintain criteria for roles.

- Containment

The containment principle unifies the principles of least privilege and separation of duty and further incorporates additional constraints and usage control elements. The concept of containment seeks to limit the damage that a user, or a set of users, can perpetrate either by deliberate malice or by victimization from malicious malware.

Our model supports containment using assignment criteria and usage criteria. Assignment criteria can be used to specify static separation of duties, cardinality constraints and mutual exclusions. The usage criteria can be used to specify dynamic separation of duties, rate of usage of roles, geographic conditions like whether user can use the role when accessing from extranet, temporal conditions, user obligations etc.

- Automation

Automation is one of the key driving principles behind SSRBAC08. Assignment and revocation have traditionally been viewed as administrative actions requiring human interventions. Failure to remove unwanted privileges that are no longer needed is a problem. This problem needs to be addressed in a automated fashion along with human intervention as needed. To this effect one needs mechanisms to automate revocation of roles that are no longer needed by the users. Another aspect of automation is to allow for self assignment of roles.

In our model we support automation via assignment criteria and revocation criteria. One can specify in the assignment criteria what all a user needs to have to get the role assigned. Further administration criteria can be used to specify if self assignment is allowed and approvals needed if any.

Similarly revocation can be handled using revocation criteria. the revocation criteria can be based on how long the assignment will be valid, user failure to meet required obligations etc.

- **Accountability**

The primary goal of accountability is to make a human user take responsibility for actions that the individual performs in a system.

Our model supports this by the assignment, revocation and usage criteria. For example, when a user assigns himself a role or removes himself from a role we can have a criteria that says “notify user’s manager and role administrator regarding the assignment or revocation”. Usage criteria can be used to make user provide more credentials or re-authenticate when using sensitive roles or operations.

6.3 Casting URA97 in SSRBAC08

In this example we show how URA97 model described in chapter 3 can be simulated in SSRBAC. In URA97 administrative tasks are performed by people with administrative roles, in SSRBAC we don’t have a notion of separate administrative roles. As explained earlier in SSRBAC08 the subjects that can perform the assignment operation are defined in ADC. So for the purpose of this example we will assume there are 4 roles namely PSO1, PSO2, DSO and SSO and these roles will be listed in ADC as the roles that are allowed to perform the assignment operation. First we will look at prerequisite role example in table 3.1. According to this table an administrator with role PSO1 can assign roles E1, PE1, QE1 to users who are already members of ED role. This requirement can be captured in SSRBAC08 for role E1 as shown below.

ADC for E1:

Assignor with PSO1 can assign this role to others

AC for E1:

Assignee needs to have ED

Table 6.3 shows how the tuples from prerequisite role example table can be stated in AC and ADC notion.

Table 6.2: Prerequisite Role in SSRBAC08

Role	ADC	AC
E1	Assignor needs to have PSO1	Assignee needs to Have ED
PE1	Assignor needs to have PSO1	Assignor needs to have PSO1
QE1	Assignor needs to have PSO1	Assignee needs to Have ED
PL1	Assignor needs to have DSO	Assignee needs to Have ED
ED	Assignor needs to have SSO	Assignee needs to Have E
DIR	Assignor needs to have SSO	Assignee needs to Have ED

As illustrated the prerequisite role requirement in URA97 becomes part of AC of the role and similarly administrative roles that have authority over the role are encoded in ADC. Table 6.3 shows how *can-assign* example with prerequisite condition in table 3.1 can be simulated in SSRBAC.

Table 6.3: Prerequisite Condition in SSRBAC08

Role	ADC	AC
ED	Assignor needs to have SSO	Assignee needs to have E
E1	Assignor needs to have PSO1 or senior	Assignee needs to have ED
PE1	Assignor needs to have PSO1 or senior	If assignor is PSO1 then assignee should have ED and not have QE1. If assignor is DSO or higher then assignee should have ED.
QE1	Assignor needs to have PSO1 or senior	If assignor is PSO1 then assignee should have ED and not have PE1. If assignor is DSO or higher then assignee should have ED.
PL1	Assignor needs to have PSO1 or senior	If assignor is PSO1 then assignee should have QE1 and PE1. If assignor is DSO or higher then assignee should have ED.
DIR	Assignor needs to have SSO	Assignee needs to have ED

URA97 does not specify any usage limits so we do not have any usage criteria, similarly

Table 6.4: URA97 revoke in SSRBAC08

Role	ADC
ED	SSO or higher can revoke
E1	PSO1 or higher can revoke
PE1	PSO1 or higher can revoke
QE1	PSO1 or higher can revoke
E2	PSO2 or Higher can revoke
PE2	PSO2 or higher can revoke
QE2	PSO2 or higher can revoke
PL1	DSO or higher can revoke
PL2	DSO or higher can revoke
DIR	SSO or higher can revoke

there are no revocation criteria either. People who can perform revoke operation are included in ADC. We take table 6.4 from URA97 and show equivalent ADC in SSRBAC08. The ADC specified in this table will be in addition to the ones specified for assignment purposes.

We have showed how URA97 can be simulated in SSRBAC08. SSRBAC08 provides what URA97 can do and more, URA97 becomes a special case in context of SSRBAC08.

6.4 Example User-Role Assignment Model

In this section we outline an example model for user-role assignment called SSURA08 based on SSRBAC08. This model is a request driven model where assignments are performed after receiving request² from the user. Our goal is to make user role assignment to be a more user initiated exercise rather than an administrator driven one. In the simplest case of self assignment a user assigns to self the needed roles, uses assigned roles and then revokes roles not needed. Figure 6.3 depicts basic steps involved in a request based self assignment and role usage. Self-assignment and usage involves users discovery of needed roles, then requesting needed roles followed by assignment, usage and finally revocation of the roles

²We understand there will be cases where a role assignment can be performed with out having a user's request for it.

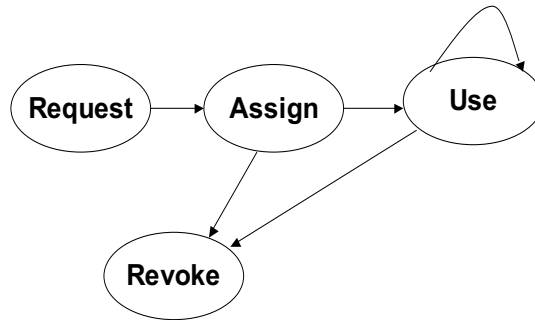


Figure 6.3: Basic Self assignment Process

that are not needed.

One of the ways to achieve a well managed and controlled self assignment administrative model is to incorporate workflow mechanism. There are several components that are need for efficient functioning of workflow mechanism. We identify some of the functions that it should support at minimum.

- Verification and validation of users who are requesting.
- Provide means for role discovery.
- Routing of requests appropriately for approval and fulfillment.
- Provide escalation and manual override approval authority support.
- Provide request statuses to requesters, approvers and administrators.
- Provide ability to audit requests, approvals, escalation and overrides.

Verification of users is achieved by available authentication mechanisms of the system. Validating a request allows to make sure that user is actually making the request and not some malware or rogue program. There can be several ways this can be achieved for

example an email notification can be sent to user and he needs to respond in a timely fashion. Alternatively user can be challenged to provide additional credentials to prove that he has initiated the request.

In Role discovery phase a user queries available roles and requests the role(s) from the available set for assignment.

Routing of requests for approval and fulfillment is another important function, when a user initiates the request for assignment some roles may need additional approvals before they can be assigned. The users or members of roles that can approve such requests can be stored as part of ADC and AC and the request should be routed for approval and successful approval will ultimately result in the assignment.

To address situations where requests are not fulfilled due to lack of response or due to non-availability of the approver, an escalation method should be provided to route request to an alternate approver.

The workflow mechanism should provide timely status notifications to involved parties at various stages of the request fulfillment and also it should allow for audit reviews of requests, approvals and assignments as needed.

6.4.1 User-Role Assignment

In the simplest case all the user-role assignments are achieved by self assignment, this may be feasible but in reality it will not be a desired goal. There will be some need for human intervention either to approve assignments or to perform actual assignments. We believe amount of human intervention can be minimized, however it cannot be made zero.

In our model we identify two ways of performing user-role assignment, which are administrator assigned and user assigned.

- Administrator assigned

In this type users initiate the request for assignment, however the assignment needs administrative action. This is a semi self assignment case as an administrator performs actual assignment after receiving the request from user.

- User assigned

In this type users initiate assignment request and role is granted to user if he meets the required criteria. This type of assignment is complete self assignment.

Following relations authorize the assignments discussed above.

Definition 12. Administrator assigned user-role assignment requests are controlled by the relation

$$can\text{-}assign\text{-}admin\text{-}ini(x,y,z). \quad \square$$

The meaning of $can\text{-}assign\text{-}admin\text{-}ini(x,y,z)$ is that a user x can assign role z to user y if x satisfies administrative criteria of role z and y satisfies assignment criteria of role z .

Definition 13. Self assigned role assignments are controlled by the relation

$$can\text{-}assign\text{-}self\text{-}ini(u,x). \quad \square$$

The meaning of $can\text{-}assign\text{-}self\text{-}ini(u, x)$ is that a user u can self assign role x if he satisfies the assignment criteria of role x and role x allows self assignment.

We defined two relations that control user-role assignment. After user initiates a request for assignment of a role appropriate relation is invoked at execution time to accomplish assignment.

6.4.2 User-Role Revocation

We now turn to consideration of the user-role revocation. In a request based assignment models like SSURA08, one can safely assume that users will be proactive in requesting and assigning the needed roles to do their job, however same assumption can not be made when it comes to revocation. Users may or may not revoke not needed roles in timely fashion and this will lead to role aggregation as job functions of a user tend to change overtime. User's having more roles and privileges than needed to do their job is not only a security but also a compliance problem.

In RBAC users are made members of roles because of their job function and revocation of roles should happen when users no longer need the role or their job function has changed.

In models like ARBAC97 administrators manage revocation and what can be revoked from whom is based on administrative roles. In SSURA08 users initiate the assignment of roles so its natural to ask if users should initiate revocation, however just having revocation based on user initiated request will not address the fundamental principle behind revocation. The revocation model should be based on containment, automation and accountability. Some possible ways of invoking revocation are detailed below.

- Revocation by expiration

A simple way to automate revocation is by placing time limits on role assignment. Placing time limits on role assignment provides means to system administered revocation and aids in preventing users from keeping roles that are no longer needed. The automated revocation concept is similar to many of the commonly deployed schemes like account lockout or account expiration after a prolonged period of inactivity.

The triggering event for role revocation can be based on period of inactivity or duration of role assignment, such limits can be placed during the assignment as part of revocation criteria and users should be made aware of them.

An optional notification can be added to warn users of upcoming revocation, so that they are aware of it and can take corrective action if needed. Once a revocation is done an administrative action is needed to undo the operation.

- Revocation by request

Revocation by request allows users to remove themselves from role(s) they no longer need. Application owners or security administrators can also make such requests if they determine a user no longer needs to be a member of role.

- Revocation by review

Privilege accumulation and role aggregation adds to compliance issues on top of violating widely regarded least privilege security principle, while properly administered revocation by expiration and user initiated revocation can help in minimizing the issue they may not be sufficient. Revocation by review is driven by the fact that there

needs to be periodic audits on what access user(s) have and performing revocations as needed to satisfy regulatory compliance and internal security requirements. Any one who has appropriate authority can initiate such a review and revocation, some of the relevant people that would perform such reviews and revocation would include

- Managers of user(s)
- Security administrators
- Application owners

Organizational policies should make some combination of these groups of users to run periodic checks on user(s) to validate that user has only needed set of roles. To make them accountable for such reviews everyone performing reviews need to certify it and these can be further reviewed as needed. The motivation for security administrators and application owners to perform such audit reviews may be simple as they are the stakeholders and are accountable for application and security needs. However to make individual user managers to perform audits on the subordinates may need some strong top down organizational policies that will motivate them to do so. We feel decisions on when, how and who should initiate these reviews should be dictated by scope, scale and needs of the organization.

We now introduce our notion for authorizing revocation

Definition 14. $\text{can-revoke}(x,y,z)$ relation controls user initiated revocations. □

The meaning of $\text{can-revoke}(x,y,z)$ is a user x can revoke role z from user y if x satisfies the ADC and RC of role are satisfied.

In the context of automatic revocation initiated by system based on expiration the revocations are performed according to definition 15.

Definition 15. Automatic revocation Algorithm

- check if role has automatic revocation condition.

- get all the members assigned to the role.
- revoke the role from users who satisfy automatic revocation criteria. □

There could be several triggering events based on user actions that may necessitate need to invoke revocation of roles, such usage based decisions can be addressed to some degree by incorporating appropriate UC.

Sometimes taking on senior roles implies obligations that need to be met in such a context self revocation of roles may not be allowed until obligations are met. These kinds of situations can arise in professional or social community context. Such conditions need to be properly captured in RC.

Another situation that could arise if not controlled properly is cycle of assignments and revocations. For example Frank self assigns himself E1 role and later Dorothy revokes E1 from Frank. Frank can go ahead self assign E1 role if there are no checks. The fact that Dorothy revoked a self assigned role from the user should be noted and make Frank obtain additional approvals before he can be assigned E1 role again.

6.5 Summary

In this chapter we outlined a framework for self service user-role assignment called SSRBAC08. SSRBAC08 is a modification to RBAC96 model. SSRBAC08 classifies constraints into four categories namely assignment, administrative, usage and revocation criteria. Each of these criteria holds the conditions that need to be satisfied for the purpose of role assignment, usage and revocation. SSRBAC08 is based on ASCAA principles and our intent in this work is to show how we can achieve these principles by means of AC, ADC, UC and RC. SSRBAC08 allows for automating user-assignment tasks in a controlled manner, specifying needed conditions in the appropriate criteria allows for controlling the flexibility. We also demonstrated how URA97 could be achieved in SSRBAC08 context. We finally showed a simple request based user-role assignment model to perform assignment and revocation in SSRBAC08 framework.

6.6 Related Work

Attribute based user-role assignment work done by Al-Kahtani and Sandhu [54] is somewhat related to our work in the context of automating user role assignment. In their work they make assignment decision purely based on rules in an implicit mode. However we allow for human element in our work and although it is possible to make our model completely self assigned, our goal is to keep it semi automated and request driven model.

Chapter 7: Conclusion

In this section we will summarize our contributions and provide directions for future work

7.1 Contributions

In this thesis we developed a user-role assignment model called URA97 for assigning users to roles and revoking users from roles. We believe URA97 is probably among the first published models to address administration in RBAC. URA97 is designed in context of the RBAC96 model. However, it should apply to almost any RBAC model, because user-role assignment is a basic administrative feature which will be required in any RBAC model. Authorization to assign and revoke users to and from roles is controlled by administrative roles. The model requires users must previously satisfy a designated prerequisite condition (stated in terms of membership and non-membership in roles) before they can be enrolled via URA97 into additional roles. URA97 applies only to regular roles. Control of membership in administrative roles remains entirely in hands of the chief security officer. We have identified strong and weak revocation operations in URA97 and have defined their precise meaning. We have also described an implementation of URA97 using Oracle stored procedures. We developed a permission-role assignment model called PRA97 on the same lines as URA97.

Then we developed a push based user-assignment model for distributed systems. Push model is an enforcement model in the context of PEI models. Push model does not place any restriction on the policy model that needs to be used. Any policy model that controls user-role assignment authorization (for example URA97) can be used with Push model. In push model we use a role authorization server to store relationship between the users, roles and databases. There is a single global hierarchy and local system support full or part of the hierarchy. The relationships between the roles remain same across the systems. Push

model has two algorithms to perform the user-role assignment. Assignment algorithm 1 assigns the senior-most incomparable junior roles when a role is not present in the local system. Assignment algorithm 2 assigns all the junior roles when a role is not present. We define a revoke algorithm to perform revocations in local systems.

RBAC96 was found on three principles *least privilege, separation of duties, abstraction* these were sufficient during the early years of RBAC96 adaptation. We believe to further extend RBAC and develop next-generation of RBAC models the founding principles needs to be expanded. To this effect we proposed five founding principles for next generation RBAC named ASCAA for abstraction, separation, containment, automation and accountability. Abstraction and separation remain same from RBAC96, we combined separation of duty and least privilege into containment principle and included concepts of usage and rate limits. Automation and accountability are new principles.

Our final contribution of this thesis is that we developed a framework for self service user-role assignment called SSRBAC08. the driving principle behind SSRBAC08 is to bring automation and self assignment capabilities to RBAC systems. We modified RBAC96 and exclusively focused on user-role assignment aspect in SSRBAC08. We classified constraints into four categories called assignment, administration, usage, and revocation criteria. We introduced explicate notion of role attributes which hold these criteria. We exemplified how these criteria can be used to encode organizational polices to allow for user-role assignment to be more self service and request driven rather than administrative controlled one. We also showed how our previous work of URA97 can be stated in SSRBAC08 framework. Then we finally proposed a simple administrative model for self assignment of roles in SSRBAC08 framework.

7.2 Future Work

Based on the research work in this dissertation, we propose the following future research directions and issues

- In this dissertation we showed how self service based user role assignment can be realized by means of SSRBAC08. In our work we used plain English to state AC,ADC,UC and RC. Natural language specification has the advantage of ease of comprehension by human beings, however to be machine-readable one needs a formal policy language to express these criteria. In future, we would like to define a formal policy language to specify AC,ADC,UC and RC.
- Al-Kahtani proposed rule-based user-assignment models in [54]. In these models the user-role assignment is an implicit operation. A user is authorized to use the role if he satisfies the rules. In future we would like to see how these attribute based models can be extended to incorporate concepts of usage control, rate limits, approval and obligations that are present in SSRBAC08.
- Zhang proposed [65] a family of extended RBAC models called Role and Organization based access Control(ROBAC). In future we would like to see if we could apply SSRBAC08 framework to ROBAC.

Appendix A: URA97 Implementation

To implement URA97 we define Oracle relations which encode the *can-assign* and *can-revoke* relations of URA97. The *can-assign* relation of URA97 is implemented in Oracle as per the entity-relation diagram of figure A.1. We assume that the prerequisite condition is converted into disjunctive normal form using standard techniques. Disjunctive normal form has the following structure.

$$(\dots \wedge \dots \wedge \dots \wedge \dots) \vee (\dots \wedge \dots \wedge \dots \wedge \dots) \vee \dots \vee (\dots \wedge \dots \wedge \dots \wedge \dots)$$

Each \dots is a positive literal x or a negated literal \bar{x} . Each group $(\dots \wedge \dots \wedge \dots \wedge \dots)$ is called a disjunct. For a given prerequisite condition *can-assign2* has a tuple for each disjunct. All positive literals of a single disjunct are in *can-assign3*, while negated literals are in *can-assign4*.

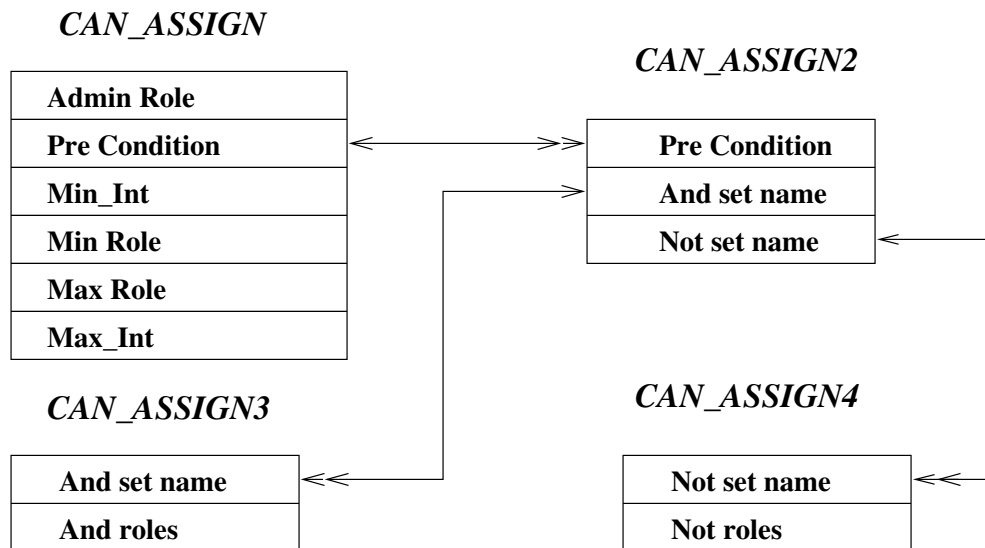


Figure A.1: Entity-Relation Diagram for *can-assign* Relation

The four PSO1 tuples of table 3.2 are represented by this scheme as shown in table A.1. The prerequisite conditions in this case all have a single disjunct. An example with multiple disjuncts is shown in table A.2.

Table A.1: Oracle *can-assign* Relations for PSO1 from Table 3.2

AR	PC	Min_Int	Min_Role	Max_Role	Max_Int
PSO1	C1	[E1	E1]
PSO1	C2	[PE1	PE1]
PSO1	C3	[QE1	QE1]
PSO1	C4	[PL1	PL1]
...

(a) *can-assign*

PC	and_set_name	not_set_name
C1	ASET1	null
C2	ASET2	NSET2
C3	ASET3	NSET3
C4	ASET4	null
...

(b) *can-assign2*

and_set_name	and_roles
ASET1	ED
ASET2	ED
ASET3	ED
ASET4	PE1
ASET4	QE1
...	...

(c) *can-assign3*

not_set_name	not_roles
NSET2	QE1
NSET3	PE1
...	...

(d) *can-assign4*

Table A.2: Oracle *can-assign* Relations for Prerequisite Condition $(A \wedge D \wedge \bar{E}) \vee (B \wedge \bar{D} \wedge \bar{F})$

AR	PC	Min_Int	Min_Role	Max_Role	Max_Int
SO1	C1
...

(a) *can-assign*

PC	and_set_name	not_set_name
C1	ASET1	NSET1
C1	ASET2	NSET2
...

(b) *can-assign2*

and_set_name	and_roles
ASET1	A
ASET1	D
ASET2	B
...	...

(c) *can-assign3*

not_set_name	not_roles
NSET1	E
NSET2	F
NSET2	D
...	...

(d) *can-assign4*

Table A.3: Oracle *can-revoke* Relation

AR	Min_Int	Min_Role	Max_Role	Max_Int
PSO1	[E1	PL1)
PSO2	[E2	PL2)
DSO	(ED	DIR)
SSO	[ED	DIR]

The *can-revoke* relation of URA97 is represented by a single Oracle relation. For example table 3.3 is represented as shown in table A.3.

The *can-assign*, *can-assign2*, *can-assign3*, *can-assign4*, and *can-revoke* relations are owned by the DBA who also decides what their content should be. In addition we have three accompanying procedures and a package to support these. There is one procedure each for assigning a user to a role, doing a weak revoke of membership and doing a strong revoke of membership, respectively as follows.

- ASSIGN
- WEAK_REVOKE
- STRONG_REVOKE

Execute privilege on these procedures is given to all administrative roles. We achieve this by introducing a junior-most administrative role, say GSO (generic security officer), and assigning it the permission to execute these procedures.

These relations and accompanying procedures and packages are owned by the DBA. Our implementation also maintains an audit relation which keeps a log of all attempted assignment and revoke operations and their outcome. The audit relation is also owned by the DBA.

Oracle does not provide convenient primitives for testing whether or not a user is an implicit member of a particular role. Testing explicit membership is straightforward since explicit membership is encoded as a tuple in Oracle's system relations. To test implicit membership, however, we need to chase the role hierarchy. Oracle also does not provide direct support for enumerating roles in a range set. We built a PL/SQL package to support these requirements and assist in writing our stored procedures, as discussed below.

In our implementation of URA97 a user invokes the stored procedure to grant or revoke a role from or to another user. The procedure calls are then as follows.

- ASSIGN(user, trole, arole)

- WEAK_REVOKE(user, trole, arole)
- STRONG_REVOKE(user, trole, arole)

The parameters user and trole (target role) specify which user is to be added to trole, or to be weakly or strongly revoked from trole. The arole parameter specifies which administrative role should be applied (with respect to the user who is invoking the URA97 procedure). The procedure code will check whether or not the user who calls the procedure has turned on the arole.¹

All the three procedures follow three basic steps.

1. If the user executing the procedure is an explicit or implicit member of arole then proceed to step 2, else stop execution and return an error message indicating this is not an authorized operation.
2. The tuple(s) from *can-assign* (for assign procedure) or *can-revoke* (for revocation procedures) are obtained where AR role value equals or is junior to the arole parameter specified in the procedure call.
3. If trole is in the specified range for any one of the tuples selected in step 2, then assign or revoke the trole else return an appropriate error message.

In case of ASSIGN also check whether the user being assigned to trole satisfies the prerequisite condition specified in the authorizing *can-assign* tuple or not.

In case of STRONG_REVOKE the operation may still fail due to all-or-nothing semantics.

The implementation of steps 1 and 3 involves complex queries built on Oracle internal tables. These queries are performed dynamically at runtime. In order to check whether the user is a member of arole (in step 1) and whether the role is in the specified range for one of the relevant *can-assign* or *can-revoke* tuples (in step 3), we use Oracle CONNECT BY clause in our queries. By using CONNECT BY clause, one can traverse a tree structure

¹It is relatively straightforward to specify a set of administrative roles instead of a single arole.

corresponding to the role hierarchy in one direction. One can start from any point within the role hierarchy and traverse it towards junior or senior roles. But there is no control on the end point of the traversal. Specific branches or an individual node of the tree can be excluded by hard coding their values. Such hard coding is not appropriate for a general purpose stored procedure. In our implementation we overcome this problem by performing multiple queries and intersecting them to get the exact range. We specifically do not hard code any parameters in our queries.

In order to modularize our implementation we developed a package which performs the necessary checks involved in steps 1 and 3. All the procedures call this package to do the verification. The package contains several functions. Each one is designed to perform certain tasks, for example we have a function called *user_has_admin_role*. This function takes the parameters from the procedure which has called it and returns the results to the calling procedure. There are other functions which determine the range for a given role.

Our implementation is convenient for the DBA since the stored procedures and packages we provide are generic and can be reused by other databases. The DBA only needs to define the roles and administrative roles, and configure the *can-assign* and *can-revoke* relations.

Bibliography

Bibliography

- [1] D. Ferraiolo and R. Kuhn, "Role-based access controls," in *Proceedings of 15th NIST-NCSC National Computer Security Conference*, Baltimore, MD, October 13-16 1992, pp. 554–563.
- [2] D. Ferraiolo, J. Cugini, and R. Kuhn, "Role-based access control (RBAC): Features and motivations," in *Proceedings of 11th Annual Computer Security Application Conference*, New Orleans, LA, December 11-15 1995, pp. 241–48.
- [3] L. Guiri, "A new model for role-based access control," in *Proceedings of 11th Annual Computer Security Application Conference*, New Orleans, LA, December 11-15 1995, pp. 249–255.
- [4] L. Guiri and P. Iglío, "A formal model for role-based access control with constraints," in *Proceedings of IEEE Computer Security Foundations Workshop 9*, Kenmare, Ireland, June 1996, pp. 136–145.
- [5] I. Mohammed and D. M. Dilts, "Design for dynamic user-role-based security," *Computers & Security*, vol. 13, no. 8, pp. 661–671, 1994.
- [6] M.-Y. Hu, S. Demurjian, and T. Ting, "User-role based security in the ADAM object-oriented design and analyses environment," in *Database Security VIII: Status and Prospects*, J. Biskup, M. Morgernstern, and C. Landwehr, Eds. North-Holland, 1995.
- [7] M. Nyanchama and S. Osborn, "Access rights administration in role-based security systems," in *Database Security VIII: Status and Prospects*, J. Biskup, M. Morgernstern, and C. Landwehr, Eds. North-Holland, 1995.
- [8] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, February 1996.
- [9] S. H. von Solms and I. van der Merwe, "The management of computer security profiles using a role-oriented approach," *Computers & Security*, vol. 13, no. 8, pp. 673–680, 1994.
- [10] C. Youman, E. Coyne, and R. Sandhu, Eds., *Proceedings of the 1st ACM Workshop on Role-Based Access Control, Nov 31-Dec. 1, 1995*. ACM, 1997.
- [11] ANSI INCITS 359-2004, *Standard for Role Based Access Control*.
- [12] M. Nyanchama and S. Osborn, "Modeling mandatory access control in role-based security systems," in *Database Security VIII: Status and Prospects*. Chapman-Hall, 1996.

- [13] R. S. Sandhu, "Role hierarchies and constraints for lattice-based access controls," in *Proc. Fourth European Symposium on Research in Computer Security*, E. Bertino, Ed. Rome, Italy: Springer-Verlag, 1996, published as *Lecture Notes in Computer Science, Computer Security-ESORICS96*.
- [14] R. Sandhu and Q. Munawer, "How to do discretionary access control using roles," in *Proceedings of 3rd ACM Workshop on Role-Based Access Control*. Fairfax, VA: ACM, October 22-23 1998, pp. 47-54.
- [15] R. Sandhu, V. Bhamidipati, and Q. Munawer, "The ARBAC97 model for role-based administration of roles," *ACM Transactions on Information and System Security*, vol. 2, no. 1, pp. 105-135, 1999.
- [16] S. Oh and R. Sandhu, "A model for role administration using organization structure," *Proceedings of the seventh ACM symposium on Access control models and technologies*, pp. 155-162, 2002.
- [17] S. Oh, R. Sandhu, and X. Zhang, "An effective role administration model using organization structure," *ACM Transactions on Information and System Security (TISSEC)*, vol. 9, no. 2, pp. 113-137, 2006.
- [18] J. Crampton and G. Loizou, "Administrative scope: A foundation for role-based administrative models," *ACM Transactions on Information and System Security (TISSEC)*, vol. 6, no. 2, pp. 201-231, 2003.
- [19] E. Bertino, P. A. Bonatti, and E. Ferrari, "TRBAC: A temporal role-based access control model," *ACM Transactions on Information and System Security*, vol. 4, no. 3, pp. 191-233, 2001.
- [20] J. Joshi, E. Bertino, and A. Ghafoor, "Temporal hierarchies and inheritance semantics for gtrbac," *Proceedings of the seventh ACM symposium on Access control models and technologies*, pp. 74-83, 2002.
- [21] J. Bacon, K. Moody, and W. Yao, "A model of OASIS role-based access control and its support for active security," *ACM Transactions on Information and System Security*, vol. 5, no. 4, pp. 492-540, 2002.
- [22] D. Kuhn, "Mutual exclusion of roles as a means of implementing separation of duty in role-based access control systems," *Proceedings of the second ACM workshop on Role-based access control*, pp. 23-30, 1997.
- [23] R. Simon and M. Zurko, "Separation of duty in role-based environments," *Proceedings of the 10th Computer Security Foundations Workshop (CSFW'97)*, p. 183, 1997.
- [24] V. Gligor, S. Gavrilu, and D. Ferraiolo, "On the formal definition of separation-of-duty policies and their composition," *Proceedings IEEE Symposium on Security and Privacy*, pp. 172-183, 1998.
- [25] G. Ahn and R. Sandhu, "Role-based authorization constraints specification," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 4, pp. 207-226, 2000.

- [26] E. Bertino, E. Ferrari, and V. Atluri, “The specification and enforcement of authorization constraints in workflow management systems,” *ACM Transactions on Information and System Security*, vol. 2, no. 1, pp. 65–104, 1999.
- [27] G. Ahn, R. Sandhu, M. Kang, and J. Park, “Injecting RBAC to secure a Web-based workflow system,” *Proceedings of the 5th ACM Workshop on Role-Based Access Control*, pp. 1–10, 2000.
- [28] S. Kandala and R. Sandhu, “Secure role-based workflow models,” *IFIP TC11/WG11.3 Fifteenth Annual Working Conference on Database and Application Security*, 2001.
- [29] E. Barka and R. Sandhu, “Framework for role-based delegation models,” *Proceedings of the 16th Annual Computer Security Applications Conference*, 2000.
- [30] L. Zhang, G. Ahn, and B. Chu, “A rule-based framework for role-based delegation and revocation,” *ACM Transactions on Information and System Security (TISSEC)*, vol. 6, no. 3, pp. 404–441, 2003.
- [31] A. Herzberg, Y. Mass, J. Michaeli, Y. Ravid, and D. Naor, “Access control meets public key infrastructure, or: Assigning roles to strangers,” in *SP ’00: Proceedings of the 2000 IEEE Symposium on Security and Privacy*, 2000.
- [32] N. Li, J. Mitchell, and W. Winsborough, “Design of a role-based trust-management framework,” *Proceedings IEEE Symposium on Security and Privacy*, pp. 114–130, 2002.
- [33] R. Sandhu, “Rationale for the RBAC96 family of access control models,” in *Proceedings of the 1st ACM Workshop on Role-Based Access Control*. ACM, 1997.
- [34] R. Sandhu, “Role activation hierarchies,” *Proceedings of the third ACM workshop on Role-based access control*, pp. 33–40, 1998.
- [35] R. Sandhu, K. Ranganathan, and X. Zhang, “Secure information sharing enabled by trusted computing and pei models,” in *ASIACCS ’06: Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, 2006, pp. 2–12.
- [36] R. Sandhu, “Engineering authority and trust in cyberspace: the om-am and rbac way,” in *RBAC ’00: Proceedings of the fifth ACM workshop on Role-based access control*, 2000, pp. 111–119.
- [37] P. Griffiths and B. Wade, “An authorization mechanism for a relational database system,” *ACM Transactions on Database Systems*, vol. 1, no. 3, pp. 242–255, 1976.
- [38] R. Fagin, “On an authorization mechanism,” *ACM Transactions on Database Systems*, vol. 3, no. 3, pp. 310–319, 1978.
- [39] T. Lunt, “Access control policies: Some unanswered questions,” in *Proceedings of IEEE Computer Security Foundations Workshop II*, Franconia, NH, June 1988, pp. 227–245.
- [40] E. Bertino, P. Samarati, and S. Jajodia, “Authorizations in relational database management systems,” in *Proceedings of 1st ACM Conference on Computer and Communications Security*, Fairfax, VA, November 3-5 1993, pp. 130–139.

- [41] E. B. Fernandez, J. Wu, and M. H. Fernandez, "User group structures in object-oriented database authorization," in *Database Security VIII: Status and Prospects*, J. Biskup, M. Morgernstern, and C. Landwehr, Eds. North-Holland, 1995.
- [42] E. Gudes, H. Song, and E. B. Fernandez, "Evaluation of negative, predicate, and instance-based authorization in object-oriented databases," in *Database Security IV: Status and Prospects*, S. Jajodia and C. Landwehr, Eds. North-Holland, 1991, pp. 85–98.
- [43] F. Rabitti, E. Bertino, W. Kim, and D. Woelk, "A model of authorization for next-generation database systems," *ACM Transactions on Database Systems*, vol. 16, no. 1, 1991.
- [44] R. Sandhu and Q. Munawer, "The arbac99 model for administration of roles," *acsac*, p. 229, 1999.
- [45] A. Kern, A. Schaad, and J. Moffett, "An administration concept for the enterprise role-based access control model," in *SACMAT '03: Proceedings of the eighth ACM symposium on Access control models and technologies*. ACM, 2003, pp. 3–11.
- [46] N. Li and Z. Mao, "Administration in role-based access control," in *ASIACCS '07: Proceedings of the 2nd ACM symposium on Information, computer and communications security*. ACM, 2007, pp. 127–138.
- [47] R. Bhatti, B. Shafiq, E. Bertino, A. Ghafoor, and J. B. D. Joshi, "X-grbac admin: A decentralized administration model for enterprise-wide access control," *ACM Trans. Inf. Syst. Secur.*, vol. 8, no. 4, pp. 388–423, 2005.
- [48] G.-J. Ahn and R. S. Sandhu, "Decentralized user group assignment in windows nt," *Journal of Systems and Software*, vol. 56, no. 1, pp. 39–49, 2001.
- [49] R. Sandhu and J. S. Park, "Decentralized user-role assignment for web-based intranets," in *RBAC '98: Proceedings of the third ACM workshop on Role-based access control*, 1998, pp. 1–12.
- [50] R. Sandhu and G. Ahn, "Decentralized group hierarchies in unix: An experiment and lessons learned," in *Proceedings of 21st NIST-NCSC National Information Systems Security Conference*, 1998, pp. 486–502.
- [51] G.-J. Ahn and S.-P. Hong, "Group hierarchies with constrained user assignment in linux," in *WOSIS*, 2004, pp. 13–22.
- [52] S. Jha, N. Li, M. Tripunitara, Q. Wang, and W. Winsborough, "Towards formal verification of role-based access control policies," 2008.
- [53] N. Li and M. V. Tripunitara, "Security analysis in role-based access control," in *SACMAT '04: Proceedings of the ninth ACM symposium on Access control models and technologies*, 2004, pp. 126–135.
- [54] M. Al-Kahtani and R. Sandhu, "A model for attribute-based user-role assignment," *Proceedings 18th Annual Computer Security Applications Conference*, pp. 353–362, 2002.

- [55] M. Dekker, J. Crampton, and S. Etalle, “Rbac administration in distributed systems,” in *SACMAT '08: Proceedings of the 13th ACM symposium on Access control models and technologies*, 2008, pp. 93–102.
- [56] E. Freudenthal, T. Pesin, L. Port, E. Keenan, and V. Karamcheti, “drbac: Distributed role-based access control for dynamic coalition environments,” in *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, 2002.
- [57] N. Li, W. H. Winsborough, and J. C. Mitchell, “Distributed credential chain discovery in trust management: extended abstract,” in *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*, 2001, pp. 156–165.
- [58] R. Sandhu, “Rationale for the RBAC96 family of access control models,” *Proceedings of the first ACM Workshop on Role-based access control*, 1996.
- [59] J. Park and R. Sandhu, “The UCON_{ABC} usage control model,” *ACM Transactions on Information and System Security*, vol. 5, no. 6, 2007.
- [60] A. Pretschner, M. Hilty, and D. Basin, “Distributed usage control,” *Communications of the ACM*, vol. 49, no. 9, pp. 39–44, 2006.
- [61] R. Sandhu, “The schematic protection model: its definition and analysis for acyclic attenuating schemes,” *Journal of the ACM*, vol. 35, no. 2, pp. 404–432, 1988.
- [62] R. Sandhu, “The demand operation in the schematic protection model,” *Information Processing Letters*, vol. 32, no. 4, pp. 213–219, 1989.
- [63] S. Osborn, R. Sandhu, and Q. Munawer, “Configuring role-based access control to enforce mandatory and discretionary access control policies,” *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 2, pp. 85–106, 2000.
- [64] T. Jaeger, “On the increasing importance of constraints,” in *RBAC '99: Proceedings of the fourth ACM workshop on Role-based access control*. New York, NY, USA: ACM, 1999, pp. 33–42.
- [65] Z. Zhang, “Scalable role and organization based access control and its administration,” Ph.D. dissertation, Fairfax, VA, USA, 2008.

Curriculum Vitae

Venkata Bhamidipati was born in 1972, in India and is a citizen of India. He received Bachelors degree in Mechanical Engineering from Andhra University, India in 1994. He received his Masters degree in Information Systems from George Mason University in 1996. He is currently employed at Oracle USA, Inc. as a Technical Director of Consulting.