

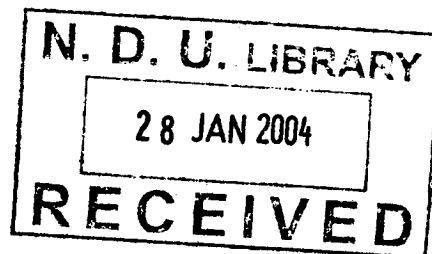
Role Base Access Control and its Administrative Implementation with Windows 2000

By
Chukri Akhras

A Thesis

Submitted in Partial Fulfillment of the Requirements for
the Degree of Master of Science in Computer Science
Computer Information Systems Concentration
Department of Computer Science

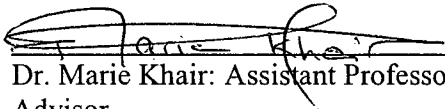
Faculty of Natural and Applied Sciences
Notre Dame University - Louaize
Zouk Mosbeh, Lebanon
Fall 2003



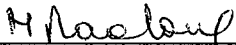
Role Base Access Control and its Administrative implementation with Windows 2000

By
Chukri Akhras

Committee Members



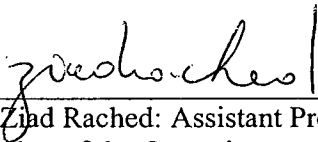
Dr. Marie Khair: Assistant Professor of Computer Science.
Advisor



Dr. Hoda Maalouf: Assistant Professor of Computer Science and Chairperson.
Member of the Committee



Dr. Omar Rifi: Assistant Professor of Computer Science.
Member of the Committee



Dr. Ziad Rached: Assistant Professor of Mathematics.
Member of the Committee

Date of Thesis Defense: December 3rd, 2003

ACKNOWLEDGMENTS

I want to thank the Lord who gave me the strength to make all this possible.

Next, I owe a big thanks to my wife, Caroline, who encouraged me to start this task and who gave me all the support especially when I needed it most. I would like to thank my wonderful children who loved and tolerated me through this process, my mom and sisters, and Christo and Betty. Moreover, I have a special thanks to Christine, the ultimate supporter in every circumstance.

I have a very special gratitude for my thesis supervisor, Dr. Marie Khair, . Furthermore, I have to thank Mr. Oskar Bou Shaia, for his technical guidance. Numerous other people have helped to contribute to the completion and ultimate success of this project, and I owe them sincere appreciation.

ABSTRACT

Role Base Access Control and its Administrative Implementation with Windows 2000 ®

Administration of access control was and still is a crucial, critical and complex aspect of Security Administration. Many models were developed and used to effect this administration such as Mandatory Access Control (MAC), Discretionary Access Control (DAC) and Role Based Access Control. The latter, RBAC which is a flexible and policy-independent access control, represents a natural structure of an organization where functions are grouped into roles and users are permitted to one or more of these roles. In large organizations with relatively large systems, with hundreds of roles and users and thousands and more of permission(s), managing all the roles, users, and permission(s) is not an easy task that can be centralized in a small team of security administrators.

While it is not a new concept, Role Based Access Control continues to gain wider commercial acceptance as it simplifies and enhances definition, auditing and administration of security access rights. Moreover, it has been implemented in different areas, such as ORACLE and Solaris.

In this thesis RBAC is applied to Windows 2000 in order to simplify the management of security, through using a simulation of ARBAC administration capabilities on Windows 2000 implementing groups hierarchies and the decentralization of group assignments.

TABLE OF CONTENTS

ACKNOWLEDGMENT.....	III
ABSTRACT.....	IV
TABLE OF CONTENT.....	V
LIST OF FIGURES.....	VIII
LIST OF TABLES.....	IX
1. INTRODUCTION AND PROBLEM STATEMENT.....	1
1.1. Security.....	2
1.2. Access Control Models.....	3
1.2.1. Discretionary Access Control.....	3
1.2.2. Mandatory Access Control.....	4
1.2.3. Role Base Access Control.....	4
1.3. Problem Statement.....	5
1.4. Organization of Thesis.....	6
2. RBAC96 MODEL FOR ROLE BASE ACCESS CONTROL.....	7
2.1. Role Base Access Control.....	7
2.1.1. Role Definition.....	7
2.1.2. Role Perspectives.....	9
2.1.3. Role Hierarchies.....	9
2.2. Role Authorization.....	10
2.2.1. Role Subsidiary.....	11
2.2.2. Integrity.....	11
2.2.3. Cardinality.....	11
2.2.4. Least Privilege.....	12
2.2.5. Separation of Duties.....	12
2.3. Role Activation.....	13
2.4. Role Based Administration.....	14
2.5. Advantages of ARBAC.....	16
2.6. RBAC96 Model.....	17
2.6.1. RBAC96 Administrative Model.....	21
3. ARBAC97 ADMINISTRATIVE MODEL FOR ROLE BASE....	22
ACCESS CONTROL	
3.1. Introduction.....	22
3.2. ARBAC97 Administrative model.....	22
3.2.1. URA97: User-Role Assignments.....	25
3.2.2. PRA97: Permission(s)-Role Assignments.....	27
3.2.3. RRA97: Role-Role Assignments.....	28
3.3. Conclusion.....	30

4. ARBAC99 ADMINISTRATIVE MODEL FOR ROLE BASE...	32
ACCESS CONTROL	
4.1. Reasons for Enhancements.....	32
4.1.1. Mobility and Immobility membership.....	33
4.2. URA99 Model.....	33
4.2.1. URA99 can_assign.....	34
4.2.2. URA99 can_revoke.....	35
4.3. PRA99 Model.....	36
5. WINDOWS 2000 Security.....	37
5.1. Introduction to Window 2000 Security.....	37
5.2. The Windows 2000 Security Model.....	38
5.2.1. Authentication Protocols.....	38
5.2.2. Access Controls.....	39
5.3. Understanding User and Group Accounts.....	39
5.3.1. Differences Between User and Group Accounts.....	40
5.3.2. User Accounts.....	40
5.3.3. Groups.....	41
5.4. Account Capabilities.....	44
5.4.1. Privileges.....	44
5.4.2. Logon Rights.....	44
5.4.3. Built-In Capabilities for Groups in Active Directory...	45
5.4.4. Access Permission(s).....	45
5.5. Windows 2000 Active Directory.....	45
5.5.1. Domains.....	45
5.5.2. Organizational Units.....	46
5.5.3. Trees and Forests.....	46
5.6. Controlling Access to Active Directory Objects.....	47
5.6.1. Active Directory Permission(s).....	47
5.6.2. Using Permission(s) Inheritance.....	47
5.7. Group Policy.....	48
6. ARBAC APPLIED OVER WINDOWS 2000.....	49
6.1. Windows 2000 In a Nutshell.....	49
6.2. Inside Windows 2000.....	50
6.2.1. Administering Windows 2000 and Active Directory	50
6.2.2. RBAC, Discretionary Access Control and Windows 2000	51
6.2.3. Problems Facing Administrating Windows 2000....	52
6.2.3.1. Handling Hierarchy in Windows 2000.....	52
6.2.3.2. Centralization of Administrating Windows 2000	53
6.3. Proposed Solution.....	53
6.3.1. Group Hierarchies.....	54
6.3.2. Decentralization of Groups.....	54
6.3.2.1. User-Group Assignment.....	55
6.3.2.2. User-Group Revocation.....	55
6.4. Implementation of the Proposed Solution.....	55

6.4.1. The GUI Interface.....	57
6.4.2. The RCP and Batch Files Used.....	58
6.4.3. Running The Implementation.....	59
6.4.4. Implementation of The URA97 Model on Windows 2000	60
6.4.4.1.Can_Assign Relation.....	61
6.4.4.2. Can_Revoke Relation.....	62
6.4.4.3. Prerequisite Conditions and Constraints.....	63
6.5. Conclusion.....	63
7. CONCLUSION.....	65
7.1. Conclusion.....	65
7.2. Recommendation for Future Work.....	66
REFERENCES.....	67

LIST OF FIGURES

2.1 Example of a Role Hierarchy.....	10
2.2 RBAC96 Model	17
3.1 (a) Roles.....	23
3.1 (b) Administrative Roles.....	24
5.1 How to Use Groups in a Single Domain.....	43
6.1 Role & Administrative Role Hierarchy.....	56
6.2 RBAC GUI.....	57
6.3 Decentralized Administration of URA97.....	60

LIST OF TABLES

3.1 Example of can_assign.....	26
3.2 Example of can_revoke.....	26
3.3 Example of can_assignp.....	27
3.4 Example of can_revokep.....	27
3.5 Example of can_modify.....	29
4.1 Example of can_assign_M.....	35
4.2 Example of can_assign_IM.....	35
4.3 Example of can_revoke_M.....	35
4.4 Example of can_revoke_IM.....	36
6.1 Batch Files.....	58
6.2 Text Files.....	58
6.3 Command-Line Tools for Active Directory.....	59
6.4 Example of can_assign.....	61
6.5 The File Assigned.txt.....	62
6.5 The File Revoked.txt.....	63

CHAPTER 1

INTRODUCTION AND PROBLEM STATEMENT

Information management systems and databases are being increasingly used by many enterprises and organizations to store their vital information.

In any enterprise or organization when considering sharing of data and information by several users, many factors have to be considered by the system administrator.

On a small scale, usually the system is maintained by the person or the individual who creates, owns, and uses it. However, enterprises and organizations' systems and information are big and complex enough that the designing, creation, and maintenance of the system are assigned to professional people called the system administrators. Those system administrators are responsible for many crucial tasks, including the design of the conceptual and physical schemas, where the system administrator has to understand the system, what data to be stored and where to be stored. The system administrator has also to understand who is going to use the data (USERS) and how they are going to use it. Based on this knowledge the system administrator has to design the conceptual schema, decide the relations that have to be stored, the physical schemas, and how to store those relations.

Another important responsibility the system administrator has to take is the authorization and security. The system administrator has to make sure that

- a) No one can access the system but through the appropriate channels.
- b) No one can access the system unless he has the authority to do so.
- c) Security checks are performed whenever are needed and every time there is an attempt to access the data.
- d) Different users can access data with different access types (Modify, Read, Delete, etc).

In almost every organization, overqualified and overloaded system administrators are spending significant amounts of time and energy performing daily routine actions. Such administrative tasks as creating and deleting accounts, changing passwords, and updating user properties should not require the attention of a well-trained and highly paid administrator. The ability to assign these tasks to junior IT staff or department administrators is critical as a means of allowing the senior IT staff

to focus on crucial business initiatives and activities that increase performance and productivity.

Delegated administration in a network means that selected users are given permission to manage certain properties of objects such as users, computers and domains. In many organizations where administration is delegated, too many people have too much administrative access and control over the network. To ensure system integrity and security, permission(s) delegation must be closely targeted without an increase of membership in the administrators group. For example, database administrators often need authority to start and stop database services, but providing this authorization results in them having the ability to start and stop all the services running on that server. IT organizations need a simple and effective way to delegate precise control without compromising security or manageability.

1.1 Security

Whenever a system administrator has to evaluate the security of an enterprise, s/he should start by stepping back and taking a look at the big picture. S/he has to try to view things in the context of an overall security framework. To successfully implement the security services in an enterprise the administrator has to consider: confidentiality, integrity, availability, and accountability. These are the core things that any administrator has to take care about when designing and implementing security for an enterprise.

- **Confidentiality:** Keeping information and resources from being disclosed to someone who hasn't been explicitly granted access.
- **Integrity:** Ensuring that information and resources remain complete and unchanged from a previous state.
- **Availability:** Ensuring that information and resources can be used whenever they're needed.
- **Accountability:** Assigning and tracking responsibility for the actions of users and resources.

When the issue is about security, what is important is how the organization can provide the basic and essential means for protecting its computer systems and information from damage, abuse or unauthorized access.

1.2 Access Control Models

Access is the ability to do something with a computer resource (e.g., use, change, or view). Access control is the means by which the ability is explicitly enabled or restricted in some way. Computer-based access controls can prescribe not only who or what process may have access to a specific system resource, but also the type of access that is permitted. These controls may be implemented in the computer system or in external devices (12).

Access can be in different forms, and can be granted different access rights or privileges. A privilege allows a user to access some data objects in a certain manner, such as retrieve, modify, create, delete, read, update, etc. It is the duty of the SA to ensure that a proper check is performed and certain means of protection are implemented to ensure the security of the data.

This control on the access of information can be performed in different ways. Many approaches had been established for the access control, among them are the discretionary, mandatory and role-based access control.

1.2.1 Discretionary Access Control

A user who creates an object such as a table or a view automatically gets all applicable privileges on that object. DAC permits the granting and revoking of access privileges to be left to the discretion of the individual users allowing them to grant or revoke access to any authority under their control without the intercession of a system administrator (4). It acts as a mean of restricting access to resources based on the identity of persons and/or groups to which they belong. DAC, is an access control mechanism that permits system users to allow or disallow other users the access to objects under their control.

1.2.2 Mandatory Access Control

MAC is a means of restricting access to data based on varying degrees of security requirements for information contained in the objects. It is based on system policies that can not be changed by the individual users. Each database object is assigned a security class, each user is assigned a clearance for a security class, and rules are imposed on reading and writing. Information is associated multi-level security requirements with labels such as TOP SECRET, SECRET, and CONFIDENTIAL. The Database Management System (DBMS) determines whether a given user can read or write a given object depending on certain rules that involves the security level of the object and the clearance of the user. These rules make sure that sensitive data can never be 'passed on' to users without the required clearance. The most common MAC is used to secure military systems. In short, MAC requires all those who create, access, and maintain information to follow rules set by administrators (26).

1.2.3 Role-Based Access Control

RBAC has been developed at the NIST (National Institute of Standards & Technology) to meet the needs of industry (26). RBAC is a form of mandatory access control but not based on multilevel security requirements. Rather, access control decisions are determined by the roles individual users take on as part of an organization. RBAC tends to be modeled after the natural structure of an organization where functions are grouped into roles and users are permitted to one or more of these roles. RBAC, in a different way than MAC, instead of labeling information, it associates roles with each individual who may be in need to access and use this information, in such a way that each role defines a specific set of information that the individual associated to this role may access (26). Roles can be granted to users and to other roles. The security policy of the organization determines role membership and the allocation of each role's capabilities. Unlike DAC, Under RBAC mechanism, users can not pass access permission(s) on to other users at their discretion, which is the fundamental difference between DAC and RBAC (5). RBAC's premise is that the organization, not the user, owns the objects being secured. In most implementations, users cannot sub-delegate access permission(s) on to other users at their discretion (4).

1.3 Problem Statement

RBAC has received considerable attention as a promising alternative to traditional discretionary and mandatory access control. RBAC is policy neutral and flexible. The policy that is enforced is a consequence of the detailed configuration of various components of RBAC. The flexibility of RBAC allows a wide range of policies to be implemented. The administration of RBAC is very important and must be carefully controlled to ensure the policy does not drift away from its original objectives.

For large systems, managing roles, users, permission(s) and their interrelationships is a task that cannot be centralized in a small team of security officers. Decentralizing the details of RBAC administration without losing central control over broad policy is a challenging goal for system designers. The administrative RBAC model (ARBAC97) defined by Sandhu et al. (25) provides a significant advance towards this goal.

Three components are identified in ARBAC97. URA97, PRA97, and RRA97 are defined. (25).

URA97 is concerned with user-role assignments. There are two consequences of assigning a user to a role. First, the user is authorized to use the permission(s) associated with that role and its juniors. Second, the user becomes eligible for assignment to other roles by appropriate administrative roles. These two aspects of role membership are tightly coupled in URA97. Consider the example of a visitor. Assignment of a visitor to a role should allow the visitor to use role's permission(s) but this membership should not be used by administrators to assign the visitor to other roles. (a visitor is considered as an immobile user.) Therefore, there is a need to decouple these two aspects. The distinction between mobile and immobile users can be very useful in practice. The concept of mobile and immobile users is needed to be formally defined and incorporated in ARBAC, and how this concept affects user-role assignments and permission-role assignments (24).

RBAC is a promising alternative to traditional discretionary and mandatory access control. It has been shown that MAC can be accommodated in RBAC. Is DAC within the purview of RBAC? In this thesis an answer to this question will be found. A simulation of DAC using roles will be used to show the flexibility of

RBAC. RBAC can accommodate both classical form of access control. This raises an important question as to whether or not it can accommodate other generalized access control models.

1.4 Organization of the Thesis

Chapter 2 gives the brief background on Role Based Access Control Models (RBAC96), In Chapter 3 the Administrative model of RBAC (ARBAC97), is discussed, in particular the User-Role Assignment (URA97), Permission-Role Assignment (PRA97), and Role-Role Assignment (RRA97). Chapter 4 describes the evolution of URA97 and PRA97 into URA99 and PRA99 respectively. Chapter 5 introduces Windows 2000 operating system, its administrative control, security and organization. Chapter 6 will explain my attempt to simulate RBAC under Windows 2000, by developing an administrative user interface that will take advantage of the RBAC features, which are group hierarchies and the decentralization of group assignments. Finally, Chapter 7 is the conclusion in which the findings and the recommendation for future research is summarized.

CHAPTER 2

RBAC96 MODEL FOR ROLE BASE ACCESS CONTROL

2.1 Role Based Access Control

Role Based Access Control (RBAC) is considered as a promising alternative to traditional discretionary and mandatory access controls (MAC and DAC). In RBAC permission(s) are associated with the roles, and users are made members of appropriate roles thereby acquiring the roles' permission(s). This greatly simplifies the management of permission(s) (17).

2.1.1 Role Definition

A **role** can be defined as the set of transactions, functions and operations that a user or a set of users can perform within the context of an organization (5). A role is a semantic construct around which access control policy is formulated. The particular collection of users and permission(s) brought together by a role is transitory.

The role is more stable because an organization's activities or functions usually change less frequently.

Within the RBAC framework, a *user* is a person, a *role* is a collection of job functions, and an *operation* represents a particular mode of access to a set of one or more protected RBAC *objects* (5). Objects are data objects or resource objects represented by data within the computer system. Permission(s) can be applied to either single and/or many objects. Ex. Read access to a particular file or as generic as read access to all files belonging to a certain department.

Groups are typically treated as a collection of users and not as a collection of permission(s).

A role is both a collection of users on one side and a collection of permission(s) on the other side. The role serves as an intermediate to bring these two collections together.

Roles are created for various job functions in an organization and users are made members of the roles based on their responsibilities and qualifications. Users can easily be reassigned from one role to another. Roles can be granted new

permission(s) as new applications or systems are incorporated. Permission(s) can be revoked from roles as needed. (17) Role-role relationships can be established to layout broad policy objectives.

Under the RBAC framework, users are granted membership into roles based on their competencies and responsibilities in the organization. The designation of persons and transactions as objects included within a role is based on objective guidelines defined by the organization rather than on subjective decisions of a security administrator. (16)

The different operations and tasks a user is allowed to perform are strictly based on the user's role. Each user is given a membership to one or more roles depending on his/her job description. This membership can be revoked easily and either replaced by a new membership or terminated completely depending on the new job assignment(s).

Whenever a new operation is introduced, a new role association can be created, and whenever an operation is terminated, its relevant role(s) association is deleted. This ease of revoking and granting access rights simplifies the management and administration of security and privileges. Updating roles can be done without the hassle of updating the privileges for every individual user independently.

With RBAC, roles are associated with a user and define profiles that represent sets of tasks that are authorized for execution. Root authority can be set apart into appropriate packages. This reduces the opportunity for persons to go beyond their realm of expertise or to unintentionally or intentionally make a change that results in a system failure. (28)

Roles are group oriented. For each role, a set of transactions allocated to the role is maintained. A transaction can be thought of as a program object associated with data. (4) Security issues are addressed by associating programming code and data into a transaction. Access control does not require any checks on the user's or the program's right to access a data item, since the accesses are built into the transaction.

A key goal for RBAC is to simplify authorization and enable better audit capabilities so that as people change responsibilities they are simply assigned new roles. The categorization of persons within roles is segregated from the specifications of the roles themselves. The role provides a clarification and concise identification of policies for given functions. This allows for the authorities of a person to be easily

documented. By contrast, in a typical access list model, one must search the entire set of authorities to develop a clear picture of a person's rights. (3)

2.1.2 Role Perspectives

Roles can be considered from different angles: (9)

Group by Organization: This is a classic view where an organization is composed of varying segregated roles to make up the composite whole.

Group by Relative relationship: This is a view that takes into account the natural interaction of members of an organization. When a person interacts by definition the role interacts. There is also relativity as it applies to the role owner. For example, a manager role differs depending on what is being managed; say projects, people or technology.

Group by convenience: This is another classic view emphasizing ease of reference where authorizations can be manipulated and assigned in a flexible manner.

Group by selection: This is a view grouping by capability. It is similar to access control and used in a workflow process.

2.1.3 Role Hierarchies

Role hierarchies are natural means for structuring roles to reflect an organization's lines of authority and responsibility. More powerful (**Senior**) roles are shown toward the top of a hierarchy diagram, while less powerful (**Junior**) roles toward the bottom.

A role hierarchy defines roles that have unique attributes and that may contain other roles; that is, one role may implicitly include the operations that are associated with another role. Operations and roles are typically subject to organizational policies or constraints. When operations overlap, hierarchies of roles can be established. Role hierarchies facilitate organizing roles to reflect authority and responsibility. Roles can have overlapping responsibilities and privileges. Users belonging to different roles may need to perform common operations. In these cases, RBAC provides an efficient

way to avoid repeatedly specifying these general operations for each role that is created. (12)

For example, in the healthcare situation, a role Specialist could contain the roles of Doctor and Intern. This means that members of the role Specialist are implicitly associated with the operations associated with the roles Doctor and Intern without the administrator having to explicitly list the Doctor and Intern operations. Moreover, the roles Cardiologist and Rheumatologist could each contain the Specialist role (See Figure 2-1). Because roles are contained by other roles through the “contains” relationship, granting membership in a role implies membership to all the roles that role “contains.” (12)(5).

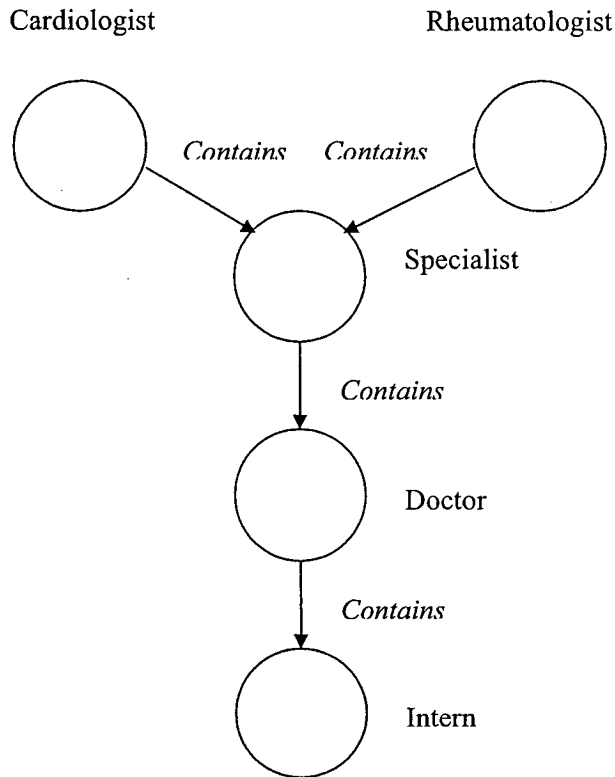


Figure 2-1 Example of a role hierarchy

2.2 Role Authorization

In order for a user to be associated with a role, or for a role to execute certain tasks, some policies should be taken into consideration:

2.2.1 Role Subsidiary

Tasks should be performed by the appropriate role that has suitable responsibility regardless of the authorities implied by the structure of the organization. As previously mentioned, role hierarchy tends to organize roles according to their shared capabilities. Role hierarchy does not fully identify organizational role complexity in terms of inheritance. Subsidiary illustrates sometime the incomplete nature of hierarchy. For example, inheritance often has exclusions due to subsidiary. For example, an IT development project manager should not change the programs codes even though he manages the project and he may even manage the programmers, but his authority does not extend completely within the hierarchy to involve changing the code of a program. This is often a classic scenario where managers who are technicians find it hard to make the transition (3).

2.2.2 Integrity

Only authorized roles should modify data and processes, and this is done only in authorized ways. The difficulty of the problem is determined by the complexity of the related transaction. A practical approach is simply for transactions to be certified and trusted. For example, bank tellers may execute a savings deposit transaction, requiring both read and write access to specific fields within savings and transaction log files. An accounting supervisor may be able to execute correction transactions, requiring exactly the same read and write access to the same files as the teller. The difference is in the process executed and the values written to the transaction log file suggesting similar but different transaction objects (3).

2.2.3 Cardinality

A user can become a new member of a role as long as the number of members allowed for the role is not exceeded. The number of users allowed for a role and the active number of users at one time are determined by the organization's policy. For example, the role of a manager can be granted to only one employee at a time.

Although an employee other than the manager may act in that role, only one person may assume the responsibilities of a manager at any given time **(21)**.

2.2.4 Least Privilege

The principle of least privilege is important for meeting integrity objectives. It requires that a user be given only the privilege necessary to perform a job. Ensuring least privilege requires identifying what the user's function is, determining the minimum set of privileges necessary, and restricting the user to a set of roles with only those privileges. By excluding users from transactions that are unnecessary for the performance of their duties, those transactions cannot be used to circumvent organizational security policy. Through the use of RBAC, enforced minimum privilege is easily achieved. **(12)**

2.2.5 Separation of Duties

Separation of duties is determined from organizational policy. RBAC facilitates splitting administration of the role-user relationship from that of the role-permission relationship **(16)**. Since many users in an organization typically perform similar functions and have similar access rights, user functions are separated by role. Separation of duties requires that for particular sets of transactions, no single role be allowed to execute all transactions within the set. As an example, there are separate transactions needed to initiate a payment and to authorize a payment. No single role should be capable of executing both transactions. Separation of duties by role is valuable in deterring fraud since fraud can occur if an opportunity exists for collaboration between various job-related capabilities.

A role can be further qualified with affiliation. A person's access could be limited to a geographic or departmental boundary. For example, a role of branch manager could be qualified by an affiliation to a particular branch thereby conferring branch manager permission only within that branch **(21)**.

Both static and dynamic forms of separation exist. Static separation means that roles which have been specified as mutually exclusive cannot be included together in a user's set of authorized roles. As an example of a Static separation of duty, or conflict of interest **(6)**, a bank Teller may not be allowed to be a member of

the role Auditor of the same bank. Dynamic separation means that, while users may be authorized for roles that are mutually exclusive, those exclusive roles cannot be active at the same time. In other words, static separation of duty enforces the mutual exclusion rule at the time of role definition while dynamic separation of duty enforces the rule at the time roles are selected for execution by a user.

2.3 Role Activation

Each subject is a mapping of a user to one or possibly many roles. A user establishes a session during which the user is associated with a subset of roles for which the user has membership. A user's role authorization (which is a consequence of role membership) is a necessary but not always sufficient condition for a user to be permitted to perform an operation. Other organizational policy considerations or constraints may need to be taken into account in relation to authorizing users to perform operations (16).

With role activation in RBAC, organizations have the means to implement their policies. A user when given a role can only use its authority after this role is activated.

Depending on the organizational policy under consideration, checks are applied in terms of the role which is being proposed for activation, the operation which is being requested for execution, and/or the object which is being accessed. That is, a role can be activated if:

- the user is authorized for the role being proposed for activation.
- the activation of the proposed role is not mutually exclusive with any other active role(s) of the user.
- the proposed operation is authorized for the role that is being proposed for activation;
- the operation being proposed is consistent within a mandatory sequence of operations. (16)

Once a role is established it can be enabled dynamically and only under certain conditions within a pre-defined scope. These constraints are separate from the

typical RBAC policy constraints of *where*, *when*, and *what*. The most common additional activation considerations include: (9)

Event-triggered: Activation occurs depending on particular circumstances and is a composite condition where other conditions or events must be enabled for the role to activate.

Qualification: This is similar to typical role criteria but reflects a dynamic attribute that suggests a person may have a different role depending on circumstance.

Delegation or transfer: This is related to *hierarchy* and also to *grant*. It speaks to whether or not the granter retains the authority as it is passed on within a hierarchy. As an example, a manager could *delegate* authority to a subordinate, as empowerment while the manager would *transfer* authority to a replacement if the manager left the organization.

2.4 Role-Based Administration

Computer scientists, government agencies, and security conferences have been examining role-based administration for about 10 years because role-based administration is considered a promising technique for simplifying security administration and more easily auditable than previous methods of administering complex security environments. The National Institute of Standards and Technology (NIST) funded research on role-based access control (RBAC). The Defense Advanced Research Projects Agency (DARPA) has also funded work on role-based access control. Conferences are now including sessions devoted to role-based administration. What makes role-based administration better than the existing security administration methods?

Role-based administration is comprised of users and groups, roles, permission(s), and resources. Users typically are assigned to groups, based on their functions within the organization. For example, financial analysts would be placed in a different group than administrative assistants because their job functions are different. Roles are also created based on the requirements of different job functions within an organization. Permission(s) are associated with roles. Roles assign permission(s) to perform a particular task, access a specific application, etc.

Permission(s) are the ability to perform specific tasks utilizing specific resources. Some examples of resources are applications, files in a directory, databases, printers, and workstations. For example, a computer operator requires access to servers and their tape drives to back up the data on the servers, while a financial analyst does not have that requirement. Decisions on the permission(s) assigned to a particular role are typically based on the organizational function performed by that role. Defining a comprehensive set of roles within an organization is critical to the success of role-based administration, if roles are to be an effective method of managing access to resources within that organization.

Using role-based administration improves security practices within an organization. Security "Best Practices" dictate that organizations write and maintain security policies that clearly outline the security roles and responsibilities for an organization. Role-based administration facilitates an organization's ability to create security policies by building security policies around the organizational roles. Role-based administration allows for separation of duties. Security administrators determine the set of permission(s) required for each role, which can be a complex process. Delegated local administrators have the much simpler task of applying the roles to individual users and groups, based on the users' or groups' job function. A security administrator converts a particular role into a list of resource permission(s) necessary for the users to perform a job function. As a user's function changes in the organization, the user can be moved to a different group and/or roles can be added or removed from the user's account. As requirements for a role change, permission(s) can be added or removed from the role to reflect the change without changing permission(s) for each and every user and group assigned to that role. By simplifying the administration tasks and eliminating the need for manually assigning and re-assigning permission(s) for individual users, role-based administration enhances security in user environments.

Once the transactions of a role are established within a system, they tend to remain relatively constant or change slowly over time. This simplifies the understanding and management of privileges. Roles can be updated without having to directly update the privileges for every user on an individual basis. (26) The administrative task consists of granting and revoking membership to the set of specified named roles within the system. When a new person enters the organization, the administrator grants membership to an existing role. User membership into roles

can be revoked easily and new memberships established as job assignments dictate. (4)

2.5 Advantages of ARBAC

A properly-administered RBAC system enables users to carry out a broad range of authorized operations, and provides great flexibility and breadth of application. System administrators can control access at a level of abstraction that is natural to the way that enterprises typically conduct business. This is achieved by statically and dynamically regulating users' actions through the establishment and definition of roles, role hierarchies, relationships, and constraints. Thus, once an RBAC framework is established for an organization, the principal administrative actions are the granting and revoking of users into and out of roles (12).

The strength of RBAC as an access control mechanism is its flexibility in that it secures at the level of operations where an operation may theoretically be anything. This is in contrast to other access control mechanisms where bits or labels are associated with information record. These bits or labels indicate relatively simple operations, such as, read or write, which can be performed on an information record. One goal of implementing RBAC is to allow operations associated with roles to be as general as possible while not adversely impacting the administrative flexibility or the behavior of applications. (2)

Furthermore, it is possible to associate the concept of an RBAC operation with the concept of "method" in Object Technology. This association leads to approaches where Object Technology can be used in applications and operating systems to implement an RBAC operation (12).

For distributed systems, RBAC administrator responsibilities can be divided among central and local protection domains; that is, central protection policies can be defined at an enterprise level while leaving protection issues that are of local concern at the organizational unit level. For example, within a distributed healthcare system, operations that are associated with healthcare providers may be centrally specified and pertain to all hospitals and clinics, but the granting and revoking of memberships into specific roles may be specified by administrators at local sites (12).

2.6 RBAC96 MODEL

A general family of RBAC models called RBAC96 was defined by Sandhu et al (16). Figure 2.1 illustrates the most general model in this family. For simplicity the term RBAC96 is used to refer to the family of models as well as its most general member.

The top half of figure 2.2 shows (regular) roles and permission(s) that regulate access to data and resources. However, the bottom half shows administrative roles and permission(s). Administrative Permission(s) AP are disjoint from regular permission(s) P. Administrative Roles AR are disjoint from regular roles R. Where Permission(s) can only be assigned to Roles and Administrative Permission(s) can only be assigned to Administrative Roles.

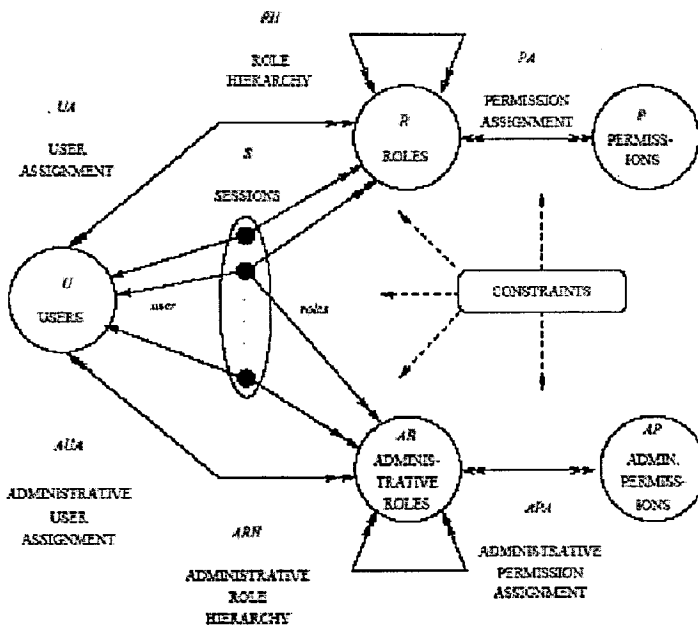


Figure 2-2: RBAC96 Model

Intuitively, a user is a human being or an intelligent autonomous agent such as robots, software agents, immobile computers, or even network of computers. A role is a job function or a job title within the organization with some associated semantics

regarding the authority and responsibility conferred on a member of the role, and permission is an approval of a particular mode of access to one or more objects in the system. Other terms as authorization, access rights and privileges are also used in order to denote permission(s). Permission(s) are always positive and confer the ability of the holder of the responsibility to perform some action(s).

Administrative permission(s) 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 permission(s) on the other hand control operations on the data and resources and do not permit administrative operations. The term role is loosely used to include both regular and administrative roles while making this distinction precise whenever appropriate. The same applies for the term permission. (15)

The user assignment (UA) and permission assignment (PA and APA) relations of Figure 2-2 are many-to-many. A user can be a member of many roles, and a role can have many users. Similarly, a role can have many permission(s), and the same permission can be assigned to many roles. There is a partially ordered role hierarchy RH, also written as $>$, where $x > y$ signifies that role x inherits the permission(s) assigned to role y , but not vice versa. In such cases, x is senior to y . By obvious extension $x \geq y$ means $x > y$ or $x = y$. 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 (7).

Inheritance of permission(s) is transitive too. Inheritance is reflexive because a role can inherit its own permission(s). Inheritance is anti-symmetry, this to rule out roles that inherit from one another and would then be redundant.

A user is allowed to establish a session with any combination of roles junior to those the user is a member of. The permission(s) in a session are those directly assigned to the active roles of the session as well as those assigned to roles junior to these.

We define a Session as the mapping of one user to possibly many roles. A user may have multiple sessions open at the same time, each in a different window. Each session may have a different combination of active roles. A session is a unit of access control, and a user may have multiple sessions with different permission(s) active at the same time. Sessions are under the direct control of individual users. A user can create a session and chose to activate some subset of the user's roles. Roles

that are active in a session can be changed at the user's discretion and the session terminates at the user's initiative.

Each session in Figure 2-2 relates one user to possibly many roles. As previously explained, a user establishes a session during which the user 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 indicates that multiple roles and administrative roles can be simultaneously activated. The permission(s) available to the user are the union of permission(s) from all 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 permission(s) active at the same time.

Finally, Figure 2-2 shows a collection of constraints. Constraints are a powerful and important mechanism for enforcing higher-level organizational policy. Once certain roles are declared to be mutually exclusive, there would be no need to much concern about assignment of individual users to roles.

Constraints can also apply to sessions, and to the user and roles functions associated with a session. 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 (15).

For centralized management of RBAC, in a single security officer, constraints are a useful convenience. While for decentralized management of RBAC, constraints become a mechanism by which senior security officers can restrict the ability of users who can exercise administrative privileges. The most frequent constraint in the context of RBAC is *mutually exclusive roles*. The same user can be assigned to at most one role in a mutually exclusive set. This supports the concept of separation of duties.

When we have a senior role and a junior role, a Role Range can be specified as the range that includes all roles between these two endpoints. The [and] brackets indicate that respectively the junior and senior end points are included in the range. The (and) brackets indicate that respectively the junior and senior end points are not included in the range.

Another concept that needs to be discussed here is the Prerequisite Roles concept. This concept is based on competency and appropriateness. A user can be assigned to a role A only if the user is already a member of role B . So the prerequisite role is junior to the new role being assumed.

On the other hand, Prerequisite Conditions, is a Boolean expression using the \wedge and \vee operators. A user can be assigned to a role A only if the user is already a member or not a member of a role B .

A user can be a member of two roles, but can not be active in both at the same time. In a role hierarchy, the constraint is that permission assigned to a junior role must also be assigned to all senior roles. Equivalently, the constraint is that a user assigned to a senior role must also be assigned to all junior roles (7).

The following definition formalizes the above discussion, and show the different components of the RBAC96 model:

- 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 permission(s) and administrative permission(s);
- $UA \subseteq U \times (R \cup AR)$, 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 , to a single user 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 set of roles and administrative roles $roles(S_i) \subseteq \{r \mid (\exists r' \geq r)[(user(S_i), r') \in UA \cup AUA]\}$ which can change with time.
Session S_i has the permission(s) $\cup_{r \in roles(S_i)} \{p \mid (\exists r'' \leq r)[(p, r'') \in PA \cup APA]\}$
- There is a collection of constraints stipulating which values of various components of the RBAC model are allowed or forbidden. (15)

2.6.1 RBAC96 Administrative Model

As mentioned previously, in large systems, the number of roles can be in the hundreds or thousands. Managing these roles and their interrelationships is a task that often is highly centralized and delegated to a small team of security administrators. Because the main advantage of RBAC is to facilitate administration of permission(s), it is natural to ask how RBAC can be used to manage RBAC itself. It is believed that the use of RBAC for managing RBAC will be an important factor in the success of RBAC.

RBAC96 makes a clear distinction between permission(s) and administrative permission(s) and likewise between roles and administrative roles. In the philosophy of RBAC, the administrative model itself is policy neutral but does facilitate formulation and articulation of administrative policy. This is an important area for research and for the future of RBAC. Effective decentralized management of permission(s) within parameters established by central authority will be required to implement enterprise-wide information systems (16)(15).

CHAPTER 3

ARBAC97 ADMINISTRATIVE MODEL FOR ROLE BASE ACCESS CONTROL

3.1 Introduction

In large organizations, the number of roles can exceed hundreds or thousands, while the number of users can reach tens or hundreds of thousands. Managing such a number of roles and users, and their relationships can be a big task that requires a professional, dedicated, centralized team of security administrators.

Decentralizing the details of RBAC administration without losing central control over broad policy is a challenging goal for system designers. A possibility is to use RBAC itself to facilitate decentralized administration of RBAC. The model subject of the discussion is called Administrative RBAC97. It consists of three components. URA97, PRA97, and RRA97 for the administration of user-role, permission-role, and role-role assignments. (23)

Before describing the three different components of ARBAC97, some definitions are needed in order to understand clearly the ARBAC97 model.

3.2 ARBAC97 Administrative Models

Consider the example of Role-Based Access Control (RBAC) where roles are created according to the job functions in the organization. Permission(s) are associated with roles and users are made members according to their qualifications and experience, thereby acquiring the associated permission(s). Role hierarchies can be created where senior roles inherit permission(s) from junior roles. The user-role, permission-role and role-role assignments are controlled by administrative roles. One such example is that of an engineering department with the regular role hierarchy shown in figure 3-1(a) and administrative role hierarchy of figure 3-1(b). There is a junior-most role E and every employee of the organization is a member of this role. Within the engineering department there is junior-most role ED and senior-most role DIR. Within the department there are two projects, project 1 and project 2. Each

project has a senior-most project lead (PL1 and PL2) and junior most engineer role (E1 and E2) (24).

In between, each project has two incomparable roles, production engineer (PE1 and PE2) and quality engineer (QE1 and QE2). This structure can be extended to dozens or hundreds of projects within the engineering department. Moreover each project could have different structure of its roles.

The administrative hierarchy (figure 3-1(b)) coexists with the regular role hierarchy (figure 3-1(a)). There is a senior security officer role (SSO). The simplicity of administration requires the decentralization of responsibilities. Therefore our interest is in the administrative roles that are junior to SSO. There are two project security officers (PSO1 and PSO2) and a department security officer (DSO) role. The relationship of these roles is shown in figure 3-1(b). PSO1 and PSO2 have partial responsibilities over project 1 and project 2 respectively. The authority of PSO1 and PSO2 over a part of the role hierarchy implies autonomy in modifying the internal role structure of that part. That includes the creation and deletion of roles as well as the alteration of role-role relationships by adding and deleting edges (25)(7).

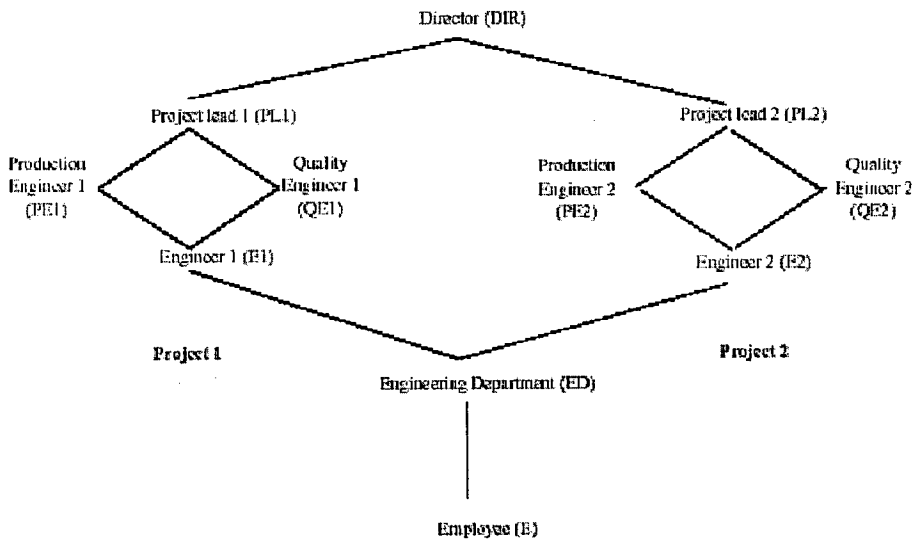


Figure 3-1 (a) Roles

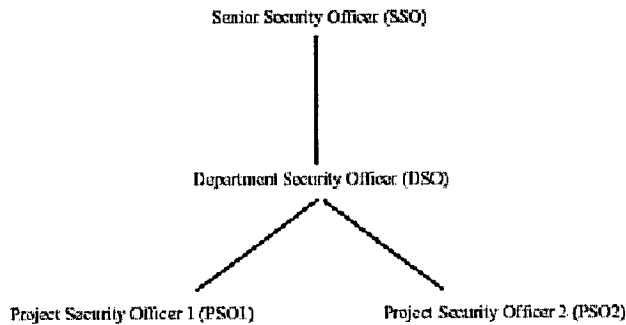


Figure 3-1 (b) Administrative Roles

If Administrative Roles are mutually exclusive with respect to regular roles: security administrator can manage RBAC but not use any of the privileges themselves.

In general, each administrative role will be mapped to some subset of the role hierarchy it is responsible for managing. More generally, there is the need to scope not only the roles an administrative role manages, but also the permission(s) and users that role manages. It is also important to control changes in the role hierarchy itself (7).

In fact, it is possible to separate the issues of assigning users to roles, assigning permission(s) to roles, and assigning roles to roles in order to define a role hierarchy.

This can be best done by different administrators or administrative roles.

Assigning Permission(s) to Roles can be done by Application Administrators. While assigning Users to Roles can be done by Personnel Management function. Therefore, it is now possible to introduce the new components of ARBAC97 which are: (25)

- 1- User-Role Assignment component URA97.
- 2- Permission-Role Assignment component PRA97.
- 3- Role-Role Assignment component RRA97.

3.2.1 URA97: User-Role Assignment

URA97 is concerned with administration of the user-assignment relation UA which relates users to roles.

URA97 is defined in two steps: granting a user membership in a role and revoking a user's membership.

Authorization to modify this relation is controlled by administrative roles. Assignment of user to administrative roles is outside the scope of URA97 and is assumed to be done by the chief security officer. 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 (25).

The notation of a *prerequisite condition* is a key part of URA97.

The Grant/Revoke Models

The URA97 model controls user-role assignment by means of the relations $can_assign(x,y,Z)$, $can_revoke(x, Y)$.

$can_assign(x,y,Z)$ means that a member of the administrative role x (or 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 in range Z .

$can_revoke(x, Y)$ means that a member of the administrative role x (or senior to x) can revoke membership of a user from any regular role y that belongs to Y . Y here defines the *range of revocation*.

It is noticed that revocation is independent of assignment. In general, it is expected that the role being assigned is *Senior* to the role previously required by the user. The revocation operation in URA97 is said to be weak because it applies only to the role that is directly revoked.

A user U is an *explicit member* of a role x if (U, x) belongs to UA .

And a User U is an *implicit member* of a role x if for $x' > x$, (U, x') belongs to UA .

For example, an explicit member of a Senior role is an implicit member of all other Junior roles in the administrative hierarchy.

It is possible for a user to simultaneously be an explicit and implicit member of a role (25).

Weak revocation applies only to explicit membership in a single role. So a user can be revoked explicitly from a junior role but continue to be an implicit member due to explicit membership in a Senior role.

Strong revocation cascades upwards in the role hierarchy. Strong revocation requires the revocation of both explicit and implicit membership.

So a user who is revoked strongly from a junior role will be revoked weakly from all other senior roles. Of course each of the weak revokes required for this purpose has to be authorized.

Strong revocation can be defined as a series of weak revocations.

can_assign involves prerequisite conditions, but can_revoke does not.

<i>Administrative Role</i>	<i>Prerequisite Condition</i>	<i>Role Range</i>
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)

Table 3.1: Example of *can_assign*

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

Table 3.2: Example of *can-revoke*

3.2.2 PRA97: Permission-Role Assignment

PRA97 is concerned with role-permission assignment and revocation. From the perspective of a role, users and permission(s) have a similar character.

The notion of prerequisite is identical to URA97, except the condition is evaluated for membership (or non-membership) of a permission in specified roles.

$can_assignp(x,y,Z)$ means that a member of the administrative role x (or 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 .

$can_revokep(x, Y)$ means that a member of the administrative role x (or senior to x) can revoke membership of a permission from any regular role y that belongs to Y . Y here defines the *range of revocation*.

<i>Administrative Role</i>	<i>Prerequisite Condition</i>	<i>Role Range</i>
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 3.3: Example of $can_assignp$

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

Table 3.4: Example of $can_revokep$

Revocation in PRA97 is weak, so permission(s) may still be inherited after revocation. Strong revocation of a permission cascades down the role hierarchy, in contrast to cascading up for revocation of user membership (25).

3.2.3 RRA97: Role-Role Assignment

For Role-Role Assignment three kinds of roles are distinguished:

- *Abilities* are roles that can only have permission(s) and other abilities as members.
- *Groups* are roles that can only have users and other groups as members.
- *UP-Roles* are roles that have no restriction on membership, i.e., their membership can include users, permission(s), groups, abilities, and other UP-roles.

The term UP-roles signifies user and permission roles. The term role is used to mean all three kinds of roles or to mean UP-roles only, as determined by context. The three kinds of roles are mutually disjoint and are identified, respectively, as *A*, *G*, and *UPR*. (23)

The main reason for distinguishing among the three kinds of roles is that different administrative models apply to establish relationships among them. The distinction was motivated in the first place by abilities. An ability is a collection of permission(s) that should be assigned as a single unit to a role. For example, the ability to open an account in a banking application will encompass many different individual permission(s). It does not make sense to assign only some of these permission(s) to a role, since the entire set is needed to do the task properly. The idea is that application developers package permission(s) into collections, called abilities, which must be assigned together as a unit to a role (23)(25).

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

Once the notion of abilities is introduced, by analogy there should be a similar concept on the user side.

A group is a collection of users who are assigned as a single unit to a role. Such a group can be viewed as a team, which is a unit even though its membership may change over time. Groups can also be organized in a hierarchy. For group-role assignment, the URA97 model is adapted to produce the GRA97 model for group-role assignment.

Assigning an ability to an UP-role is mathematically equivalent to making the UP-role an immediate senior of the ability in the role-role hierarchy.

Abilities can only have UP-roles or abilities as immediate seniors and can only have abilities as immediate juniors. In a dual manner, assigning a group to an UP-role is mathematically equivalent to making the UP-role an immediate junior of the group in the role-role hierarchy.

Groups can only have UP-roles or groups as immediate juniors and can only have groups as immediate seniors. With these constraints, the ARA97 and GRA97 models are essentially identical to the PRA97 and URA97 models, respectively. This will introduce the problem of managing relationships between UP-roles and other UP-roles, between abilities and other abilities, and between groups and other groups. The same model is used for all three cases.

The Can_Modify Relation

Decentralization of administrative authority requires that members of different administrative roles have authority over different parts of the hierarchy. Authority over a part of the role hierarchy implies autonomy in modifying the internal role structure of that part. That includes the creation and deletion of roles, as well as alternation of role-role relationships by adding or deleting the edges.

<i>Administrative Role</i>	<i>UP-Role Range</i>
DSO	(ED, DIR)
PSO1	(E1, PL1)

Table 3.5: Example of *can_modify*

In RRA97 role creation, role deletion, edge insertion, and edge deletion are all authorized by the relation `can_modify`. The meaning of `can_modify(x, Y)` is that a member of the administrative role `x` (or a member of an administrative role that is senior to `x`) can create and delete roles in the range `Y` and can modify relationships between roles in `Y`.

Creation of a new role requires the specification of its immediate parent and child in the existing hierarchy. Since creation of a role also introduces two edges in the hierarchy, it is not possible to use any two roles as the immediate parent and immediate child. Clearly this operation should not introduce a cycle in this manner.

Deletion of a role leaves relationships between the parents and children of the deleted role unchanged.

In general, some roles are referenced in various relations in URA97, PRA97, and RRA97. If these roles are actually deleted, this will lead to dangling references (23).

Roles that cannot be deleted due to this reason can be deactivated, so that they can be phased out later by adjusting the references that prevent deletion. Furthermore, when a role is deleted, an action should be taken regarding the users and permission(s) that are directly assigned to this role.

Insertion of an edge is meaningful only between incomparable nodes. Likewise, deletion of an edge is meaningful only if that edge is not transitively implied by other edges.

3.3 Conclusion

The formal definition of a new role-based model, called ARBAC97, for RBAC administration. ARBAC97 has three components: URA97 for user-role assignment, PRA97 for permission-role assignment, and RRA97 for role-role assignment.

URA97 and PRA97 are exact duals of each other, and are based on the notions of prerequisite conditions and role ranges. Both models incorporate a notion of revocation that does not seek to undo actions taken by a user when that user's administrative roles are later revoked. Also, revocation is independent of who assigned the user or permission to the role. This reflects a role-oriented style, in contrast to a discretionary style (24).

RRA97 recognizes three kinds of roles, called abilities (roles that are assigned permission(s) only), groups (roles that are assigned users only), and UP-roles (roles that are assigned permission(s) and users). ARA97 and GRA97 deal with ability to UP-role assignment and group to UP-role assignment, respectively, and are very similar to PRA97 and URA97, respectively. The component dealing with role-role assignment proper is also (at the risk of some confusion) called RRA97. It deals with modifications of relationships between the same kinds of roles: UP-roles to UP-roles, groups to groups, and UP-roles to UP-roles. The same model applies in all three cases (23).

URA97 has been implemented on several platforms (Oracle, Unix, Solaris, and the Web) (28) (1) (13) (21), while PRA97 has been implemented on Oracle (18).

Implementation of RRA97 is quite feasible on the same platforms. Thus ARBAC97 is a practically feasible model that uses commercially available products.

CHAPTER 4

ARBAC99 ADMINISTRATIVE MODEL FOR ROLE BASE ACCESS CONTROL

4.1 Reasons for enhancements

In the URA97 model described in chapter 3 the assignment of users to roles is controlled by the can-assign relation. The consequence of assigning a user to a role is that the user can use the permission(s) associated with that role and that administrative roles can use this membership to assign this user to other roles. The need is to de-couple these tightly coupled aspects of role memberships. The following examples are provided as a motivation for de-coupling these aspects of user-role membership.

1. Visitor: Assignment of a visitor to a role should allow the visitor to use the permission(s) of the role but this membership should not be used to assign the visitor to other roles.
2. Trainee: A user under training can be assigned to the ED role of the engineering department. Being a member of this role the user can participate in the engineering department while junior administrators can be prevented from assigning this user to any project. However after completion of training user's membership in ED role can be used to assign him to other roles.
3. Consultant. A consultant assigned to E2 role of hierarchy figure 3.1(a) is required to participate in project 2 and use the general resources of engineering department due to inherited membership in ED role. At the same time junior administrators have to be prevented from using this membership to assign the consultant to project 1.

In general, this model is a continuation and enhancement to the previous model ARBAC97. It consists also of three Models: (22)

- User-Role Assignment component URA99.
- Permission-Role Assignment component PRA99.
- Role-Role Assignment component RRA99.

URA99 and PRA99 differ from their original models URA97 and PRA97, while the RRA99 stays identical to its predecessor model RRA97. The important difference between URA99, PRA99 and URA97, PRA97 is the concept of mobile and immobile users and permission(s).

PRA99 is a dual of URA99 and is defined in this manner, all changes from URA97 to URA99 are reflected in the same way on PRA99.

4.1.1 Mobility and Immobility membership

What is meant by Mobility membership is to grant the user the authority to use the permission(s) of a role and to grant the user the eligibility for further assignment. What is meant by Immobility membership is to grant the user the authority to use the permission(s) of a role but does not grant the user the eligibility for further assignment.

4.2 URA99 Model

In URA99 there are two consequences of assigning a user to a role.

1. The user is authorized to use the permission(s) of that role and its juniors.
2. The user becomes eligible for further assignment to other roles by appropriate administrative roles.

These two aspects are tightly coupled in URA79, but URA99 main intend is to decouple this aspect and this is done through the mobility and immobility membership.

A user's membership in a role can be mobile or immobile.

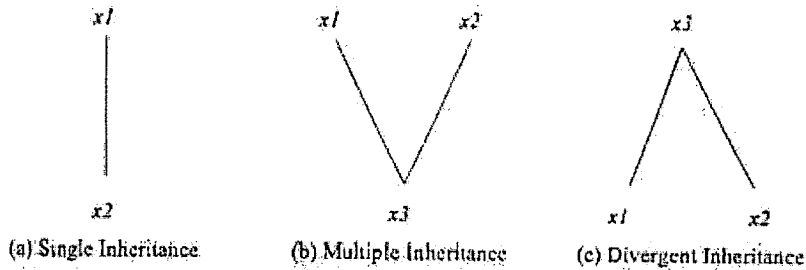
- *Mobile Membership* of user u in role x means that u can use permission(s) of role x **and** members of administrative roles **can** use this membership to put user u into other roles.
- *Immobile Membership* of user u in role x means that u can use permission(s) of role x **but** members of administrative roles **cannot** use this membership to put user u into other roles.

There are four kinds of user-role membership in URA99 for any given role x :

- *Explicit Mobile Member EMx,*
- *Explicit Immobile Member EIMx,*
- *Implicit Mobile Member ImMx,*
- *Implicit Immobile Member ImIMx,*

It is possible for a user to have all four kinds of membership in a role at the same time, but only one of those is actually in effect. Whoever, there is strict precedence amongst these four kinds of membership as follows:

$$EMx > EIMx > ImMx > ImIMx$$



URA99 can_assign

In this model two relations exist as follows:

- *can_assign_M(x,y,Z)* is that a member of administrative role x (or senior to x) can assign a user whose current membership, or non-membership, satisfies the prerequisite y to a range of roles Z as a mobile member.
- *can_assign_IM(x,y,Z)* is that a member of administrative role x (or senior to x) can assign a user whose current membership, or non-membership, satisfies the prerequisite y to a range of roles Z as an immobile member.

<i>Administrative Role</i>	<i>Prerequisite Role</i>	<i>Role Range</i>
PSO1	ED	[E1, PL1)
PSO2	ED	[E2, PL2)
DSO	ED \wedge $\overline{\text{PL2}}$	[PL1, PL1]
DSO	ED \wedge $\overline{\text{PL1}}$	[PL2, PL2]
SSO	ED	(ED, DIR]
SSO	E	[ED, ED]

Table 4.1: Example of *can_assign_M*

<i>Administrative Role</i>	<i>Prerequisite Role</i>	<i>Role Range</i>
PSO1	ED	[E1, PL1)
PSO2	ED	[E2, PL2)
DSO	ED \wedge $\overline{\text{PL2}}$	[PL1, PL1]
DSO	ED \wedge $\overline{\text{PL1}}$	[PL2, PL2]
SSO	ED	(ED, DIR]
SSO	E	[ED, ED]
DSO	E	[ED, ED]

Table 4.2: Example of *can_assign_IM*

URA99 can_revoke

In this model two relations exist as follows:

- *can_revoke_M(x,y,Z)* is that a member of administrative role *x* (or senior to *x*) can revoke a mobile membership of a user from a role or a range of roles *Z* that satisfies the prerequisite *y*.
- *can_revoke_IM(x,y,Z)* is that a member of administrative role *x* (or senior to *x*) can revoke an immobile membership of a user from a role or a range of roles *Z* that satisfies the prerequisite *y*.

<i>Administrative Role</i>	<i>Prerequisite Role</i>	<i>Role Range</i>
PSO1	E	[E1, PL1)
PSO2	E	[E2, PL2)
DSO	E	(ED, DIR)
SSO	E	[ED, DIR]
PSO1	E1	[E2, PL2)
PSO2	E2	[E1, PL1)

Table 4.3: Example of *can_revoke_M*

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

Table 4.4: Example of *can_revoke_IM*

4.3 PRA99 Model

The PRA99 model deals with assignment and revocation of permission(s) to and from the roles. Like users, permission(s) can also be assigned to roles as mobile or immobile.

There are four kinds of permission-role membership in PRA99 for any given role x :

- *Explicit Mobile Member EMx*,
- *Explicit Immobile Member EIMx*,
- *Implicit Mobile Member ImMx*,
- *Implicit Immobile Member ImIMx*,

It is possible for a user to have all four kinds of membership in a role at the same time, but only one of those is actually in effect. Whoever, there is strict precedence amongst these four kinds of membership as follows:

$$EMx > EIMx > ImMx > ImIMx$$

The PRA99 model authorizes assignment and revocation of permission(s) by using two models:

- *can_assignp_M(x,y,Z)* for mobile membership.
- *can_assignp_IM(x,y,Z)* for immobile membership
- *can_revokep_M(x,y,Z)* for mobile membership.
- *can_revokep_IM(x,y,Z)* for immobile membership. (22)

CHAPTER 5

WINDOWS 2000 Security

5.1 Introduction to Window 2000 Security

Windows 2000 is a directory-based operating system. The directory, an Active Directory is used to store user, security, application, and configuration information. The Active Directory follows a hierarchical database model. It is comprised of a forest, with one or more trees. Each tree has one or more domain, similar to the Windows NT domains. Each domain has its own instantiation of the directory database, containing objects that belong to that specific domain and possibly a subset of information about objects in other domains. A domain can be further broken out into Organizational Units (OU). The domain forms the top-level organizational unit within the domain, while all other organizational units within the domain are subordinate to the top-level domain OU. Since the Active Directory is the repository for some security information, it is one whose focus is for security administration. The Active Directory utilizes a multi-master replication model where updates can be applied to any instance of a specific directory database and are replicated to other instances of the same domain database **(11)**.

Access Control Entry (ACE) defines the permission(s) a user or group has been granted for a specific file or object. Many objects within Windows 2000 have the Security tab on their Property page, which is where access control entries are defined. A set of permission(s) for the object is assigned to a specific user or security group within Windows 2000, forming an access control entry. The set of all access control entries for a specific object form the access control list (ACL) for that object. Different objects have different available permission(s). ACEs provide the foundation for securing Windows 2000 files and other objects **(11)**.

Group Policy is a Windows 2000 administrator's primary tool for defining and controlling how programs, network resources, and the operating system behave for users and computers in an organization. Group Policy allows the administrator to specify a desired computer configuration at one time, and then to rely on the Windows 2000 environment to enforce that desired configuration on all affected client computers until the user wants to change it. After the Group Policy is set, the

system maintains the state of computers without further intervention. Group Policy Objects contain the settings that are applied to specific objects. Group Policy Objects can be applied to sites, domains, and organizational units **(10)**.

Windows 2000 comes with various tools for configuring and administering security. Security attributes can be set on each system manually via the Domain Security Policy program or through the use of a security template in the Security Templates MMC snap-in.

Windows 2000 is much more complex than Windows NT. Many different tools are required to secure different aspects of the Windows 2000 environment. While the Windows 2000 security model allows an administrator to create security groups and assign permission(s) to those groups, the tools do not facilitate auditing the access permission(s) assigned to a particular group. The Windows 2000 tools allow for delegation of administration of organizational units and domains, but the tools offer no simple view of the rights relationships within the Windows 2000 environment.

5.2 The Windows 2000 Security Model

The main task of the administrator(s) is to control the access to network resources with the components of the Windows 2000 security model. The key components that are needed are the ones used for authentication and access controls.

5.2.1 Authentication Protocols

Windows 2000 authentication is implemented as a two-part process which consists of interactive logon and network authentication.

When a user logs on to a computer, the interactive logon process authenticates the user's logon, which confirms the user's identity to the local computer and grants access to Active Directory directory service. Afterward, whenever the user accesses network resources, network authentication is used to determine whether the user has permission to do so.

Windows 2000 supports many different network authentication protocols. The key protocols are Kerberos V 5, NT LAN Manager (NTLM), and Secure Socket Layer/Transport Layer Security (SSL/TLS). A key feature of the Windows 2000 authentication model is that it supports Single Sign-On **(11)**.

5.2.2 Access Controls

The Active Directory is object-based. Users, computers, groups, shared resources, and many other entities are all defined as objects. Access controls are applied to these objects with security descriptors. Security descriptors do the following:

- List the users and groups that are granted access to objects
- Specify permission(s) the users and groups have been assigned
- Track events that should be audited for objects
- Define ownership of objects

Individual entries in the security descriptor are referred to as access control entries (ACEs). The Active Directory objects can inherit ACEs from their parent objects. This means that permission(s) for a parent object can be applied to a child object. For example, all members of the Domain Admins group inherit permission(s) granted to this group.

When working with ACEs, the following points should be kept in mind:

- Access control entries are created with inheritance enabled by default.
- Inheritance takes place immediately after the ACE is written.
- All access control entries contain information specifying whether the permission is inherited or explicitly assigned to the related object.

5.3 Understanding User and Group Accounts

Managing accounts is one of the primary tasks as a Microsoft Windows 2000 administrator. User accounts enable individual users to log on to the network and access network resources. Group accounts are used to manage resources for multiple users. The permission(s) and privileges that are assigned to user and group accounts determine which action users can perform, as well as which computer systems and resources they can access.

The administrator always has to balance the user's need for job-related resources against the need to protect sensitive resources or privileged information. For example, the management of a company does not want everyone in the company to have access to payroll data. Consequently, the management makes sure that only those who need that information have access to it.

5.3.1 Differences Between User and Group Accounts

Windows 2000 provides user accounts and group accounts (of which users can be a member). User accounts are designed for individuals. Group accounts are designed to make the administration of multiple users easier. While users can log on to user accounts, they can not log on to a group account. Group accounts are usually referred to simply as *groups* (10).

5.3.2 User Accounts

In Windows 2000 two types of user accounts are defined:

- **Domain user accounts** Users accounts defined in Active Directory are called *domain user accounts*. Through Single Sign-On, domain user accounts can access resources throughout the domain. Domain user accounts are created in Active Directory Users And Computers.
- **Local user accounts** User accounts defined on a local computer are called *local user accounts*. Local user accounts have access to the local computer only, and they must authenticate themselves before they can access network resources. The Administrator creates local user accounts with the Local Users And Groups utility.

Note Only member servers and workstations have local user and group accounts. On the initial domain controller for a domain, these accounts are moved from the local security manager to Active Directory and then become domain accounts.

All user accounts are identified with a logon name. In Windows 2000, this logon name has two parts, User name and User domain or workgroup. When working with Active Directory, an administrator may also need to specify the fully qualified domain name for a user. The fully qualified domain name for a group is the combination of the Domain Name Service (DNS) domain name, the container or organizational unit location, and the group name. User accounts can also have passwords and public certificates associated with them. Passwords are authentication strings for an account. Public certificates combine a public and private key to identify a user. Users can log on with a password interactively or with a public certificate using a smart card and a smart card reader.

Although Windows 2000 displays user names to describe privileges and permission(s), the key identifiers for accounts are security identifiers (SIDs). SIDs are unique identifiers that are generated when accounts are created. SIDs consist of the domain's security ID prefix and a unique relative ID, which was allocated by the relative ID master.

Windows 2000 uses these identifiers to track accounts independently from user names. SIDs serve many purposes. The two most important ones are to allow administrators to easily change user names and to allow them to delete accounts without worrying that someone may gain access to resources simply by re-creating an account. When administrators change a user name, they tell Windows 2000 to map a particular SID to a new name. Moreover, if an account is deleted, Windows 2000 is told that a particular SID is no longer valid. Afterward, even if an account with the same user name is created, the new account will not have the same privileges and permission(s) as the previous one. That is because the new account will have a new SID (11).

The Administrator is a predefined account that provides complete access to files, directories, services, and other facilities. This account can not be deleted or disabled. In Active Directory, the Administrator account has domain-wide access and privileges. Otherwise, the Administrator account generally has access only to the local system. Although files and directories can be protected from the Administrator account temporarily, the Administrator account can take control of these resources at any time by changing the access permission(s). In most instances the basic settings for this account do not need to be changed. However, some of the advanced settings may need to be changed, such as membership in particular groups. By default, the Administrator account for a domain is a member of these groups: Administrators, Domain Admins, Domain Users, Enterprise Admins, Schema Admins, and Group Policy Creator Owners.

5.3.3 Groups

In addition to user accounts, Windows 2000 provides groups. Groups are used to grant permission(s) to similar types of users and to simplify account administration. If a user is a member of a group that can access a resource, that particular user can access the same resource. Thus, users can have access to various work-related resources just by making them members of the correct group. Note that while a user

can log on to a computer with a user account, one can not log on to a computer with a group account.

In a Real World situation, employees in a marketing department probably need access to all marketing-related resources. Instead of granting access to these resources individually, the administrator can make the users members of a marketing group. That way, they automatically obtain the group's privileges. Later, if a user moves to a different department, this user can simply be removed from the group and all access permission(s) are revoked. Compared to having to revoke access for each individual resource, this technique is pretty easy so using groups whenever possible is advisable.

In Windows 2000, there are three types of groups:

- **Local groups:** Groups that are defined on a local computer. Local groups are used on the local computer only. Local groups are created with the Local Users And Groups utility.
- **Security groups:** Groups that can have security descriptors associated with them. Security groups are defined in domains using Active Directory Users And Computers.
- **Distribution groups:** Groups that are used as e-mail distribution lists. They can not have security descriptors associated with them. Distribution groups are defined in domains using Active Directory Users And Computers.

Groups can have different scopes—*domain local*, *built-in local*, *global*, and *universal*. That is, the groups have different areas in which they are valid.

- **Domain local groups** Groups that are used to grant permission(s) within a single domain. Members of domain local groups can include only accounts (both user and computer accounts) and groups from the domain in which they are defined.
- **Built-in local groups** Groups that have a special group scope that have domain local permission(s) and, for simplicity, are often referred to as *domain local groups*. The difference between built-in local groups and other groups is that built-in local groups can't be created or deleted. Only built-in local groups

may be modified. References to domain local groups apply to built-in local groups unless otherwise noted.

- **Global groups** Groups that are used to grant permission(s) to objects in any domain in the domain tree or forest. Members of global groups can include only accounts and groups from the domain in which they are defined.
- **Universal groups** Groups that are used to grant permission(s) on a wide scale throughout a domain tree or forest. Members of global groups include accounts and groups from any domain in the domain tree or forest (10).

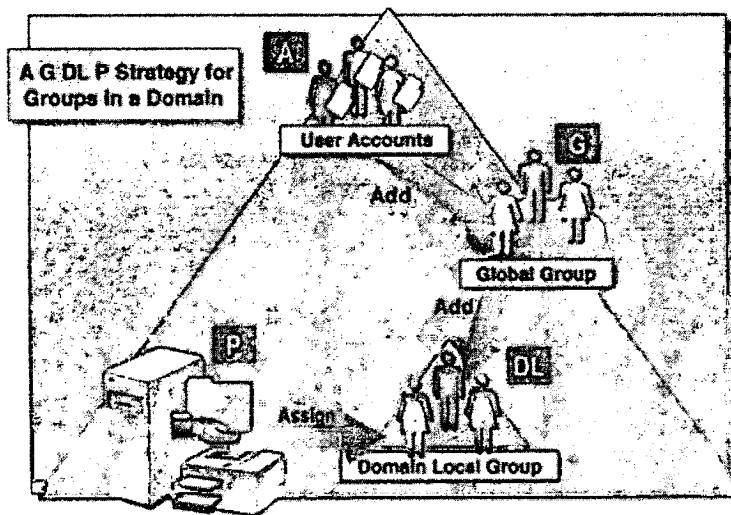


Figure 5.1 How to use Groups in a Single Domain (10)

- (A) into User Accounts
- (G) into Global group
- (DL) into Domain Local group
- (P) Grant Permission(s) to the Domain Local group.

To set up a group the following strategy is used:

1. Identify users with common responsibilities and add the user accounts to a global group.
2. Determine whether a built-in group can be selected, or a new one has to be created.
3. Make all global groups that share the same access needs for resources members of the appropriate domain local group.

4. Grant the required permission(s) to the domain local group on the domain controller.

5.4 Account Capabilities

When a user account is set up it is granted the user specific capabilities. These capabilities are generally assigned by making the user a member of one or more groups, thus giving the user the capabilities of these groups. Additional capabilities are assigned by making a user a member of the appropriate groups. Withdrawing capabilities is done by removing/revoking group membership.

In Windows 2000, various types of capabilities can be assigned to an account. These capabilities include **(10)**

- **Privileges**
- **Logon rights**
- **Built-in capabilities**
- **Access permission(s)**

5.4.1 Privileges

A privilege is a type of user right that grants permission(s) to perform a specific administrative task. These privileges are assigned through group policies, which can be applied to individual computers, organizational units, and domains. Although privileges can be assigned to both users and groups, they are usually assigned to groups. In this way, users are automatically assigned the appropriate privileges when they become members of a group. Assigning privileges to groups also makes it easier to manage user accounts.

5.4.2 Logon Rights

A logon right is a type of user right that grants logon permission(s). Logon rights are assigned to both user and group accounts. As with privileges, logon rights are assigned through group policies and are usually assigned to groups rather than to individual users.

5.4.3 Built-In Capabilities for Groups in Active Directory

A type of user right that is assigned to groups and includes the automatic capabilities of the group. Built-in capabilities are predefined and unchangeable, but they can be delegated to users with permission to manage objects, organizational units, or other containers. An example of a built-in capability is the ability to create, delete, and manage user accounts. This capability is assigned to administrators and account Operators. Thus, if a user is a member of the Administrators group, the user can create, delete, and manage user accounts.

5.4.4 Access Permission(s)

A type of user right that defines the operations can be performed on network resources. Access permission(s) can be assigned to users, computers, and groups. An example of an access permission is the ability to create a file in a directory.

5.5 Windows 2000 Active Directory

Basically the Active Directory service is a way to publish objects of interest, and in this case, the user account might be “an object of interest”, the printer that a group uses, a shared data folder, even the router just installed. “Objects of interest” could include the switch or a new Software application **(11)**.

Potentially all of these things can be accessed and managed through Active Directory to take advantage of Active Directory because its intuitive, centralized interface simplifies the organization, management, and control of network resources. With Active Directory, users log on once and have access to anything in the forest that the administrator chooses to give them. Active Directory should go a long way toward removing any reason for a single user to have more than one account.

5.5.1 Domains

Under Windows NT, a domain is a security boundary, an administrative boundary and a unit of replication. In Windows 2000, these three features haven't changed—the domain is still a security boundary, an administrative boundary, and a unit of replication. Moreover, in Windows 2000, a domain can run in one of two modes: mixed mode or native mode. Running in mixed mode is only required if some of the domain controllers are Windows NT.

5.5.2 Organizational Units

The Organizational Units (OUs) in Windows 2000 hold information about users, groups, individual accounts, policies, shares, applications, and so forth. The main value of the OU is to provide a hierarchical organization to Active Directory, as well as to allow finer, more individually, precise assignment of rights, management, and access permission(s) within a domain (10).

Although there are other reasons to create OUs, this ability is certainly one of the primary ones. There are a number of philosophies on how to create this model. It can be divided by applications, printers, locations, users, and so on. One of the best is to model the Organizational Unit structure after the administrative model rather than the business model.

There is no advantage to users to model the OU on business divisions because they do not need to know their OU in order to log on. Moreover, they do not really ever use their OU name. Since modeling the OU structure after the business could be a big disadvantage to administrative people. Instead, it is best to observe the administrative model being used now. If the domain structure is based on geography, that might be a good way to map out the OU structure.

The special nature of the solution provider business, however, may warrant a slightly different approach because having certain customers that need access to their own structure, but obviously without them being able to get to another customer's data. Therefore, solution providers may start to map their OU structure based on their customer base, or something similar.

No matter which model is chosen for the OU structure, the point is to make the system easier to *administer*.

5.5.3 Trees and Forests

As the very first domain controller is built, the root domain is built in the forest. And it is a very special domain in this forest because it anchors the entire forest structure. All the trusts, all the access to objects, are going to run through this domain. If this domain were to go away, one would have to rebuild the forest. Therefore, some clustering is probably going to be involved at this level.

5.6 Controlling Access to Active Directory Objects

An object can be put onto any resource in the forest. Once this is all set up, one needs to decide whether users can see every object in Active Directory. And for many security purposes, it is probably absolutely critical that they do not.

Not only these users should not have access, but also should not even know it exists. With Active Directory permission(s), users are not just prohibited from gaining access to an object, they do not even see the object.

5.6.1 Active Directory Permission(s)

With the concept of the Access Control List (ACL) in Windows NT, every resource has an associated list that specifies the users or groups that have permission to access it. The most permissive access to that resource is the effective one. That concept has been carried over to objects in Active Directory in something called a DACL, or Discretionally Access Control List.

DACLs assign some type of access to the object in question, which by default inherit down through that part of the Active Directory structure. The administrator can certainly change that to allow a user to see one OU, but not another, or to see certain types of objects, but not others. For example, a customer may be allowed to see users but not servers. This access is flexible in that way.

If this idea is taken a step further, users can be allowed to see some of the attributes of the objects available to them. In this case a user could, for example, find out a person's last name and phone extension but not the home address and home phone number.

With multiple permission(s), permission(s) can be allowed or denied. There are standard sets of permission(s), or as explained above, more detailed steps can be reached.

5.6.2 Using Permission(s) Inheritance

By default, permission(s) will roll down the hierarchy. However, such *permission inheritance* can be prevented, and there are two ways to do that. A user can be removed from the list, which prevents that user from getting in, or a user is denied access to the object, which means the user can never access the object.

For example, if an ACL says that the user Don has access, then he does. But if Don is not on the list and is not in any groups that are on the list, Don does not have access. If, however, Don is specifically assigned no access, it does not matter which groups he belongs to or what those groups' permission(s) are, Don does not have access. The same concept applies here.

Therefore, one needs to be careful about denying access. If a specific user has to be prevented from getting in, just leave that user off the list. And to make sure that some user or group never gets in, assign no access.

Also, existing permission(s) can be copied as inheritance is blocked, or new values can be created which will then inherit by default throughout the rest of that piece of Active Directory.

5.7 Group Policy

Group Policy is used to affect the registry, set different options for security, install and remove software, publish updates, and to handle a lot of other issues relating to centralized software distribution and management. In addition, scripts can be run for logging on, startup, shutdown, and logging off. For example, a script that cleans up a configuration during the log off process can be written.

Group Policy also enables folder redirection. For example, users' folders can always be on the server, whether they choose to put them there or not. Moreover, users' home directory can be put on a server. Users will be saving to the server, and can bring their folders back down to their machine for offline viewing. This enables users to make changes when the machine is offline. When the machine comes back online, those changes will get copied back up to the server. And all of this will be transparent to the user.

With Group Policy in Windows 2000, a Group Policy object that is usable anywhere in the forest is basically created. A Group Policy object is very much a "create once, use anywhere" type of entity, because of the ability to create a set of group objects, store them centrally, and then apply policy to users or groups from a central store, as appropriate.

CHAPTER 6

ARBAC97 APPLIED OVER WINDOWS 2000

6.1 ARBAC In a Nutshell

The introduction of administrative capabilities allowing the safe offloading of routine tasks to the most appropriate person is extremely crucial. The delegation of administrative responsibilities is vital, eliminating the need for multiple administrative accounts with broad authority and allowing the primary administrator to focus on fewer daily operations tasks. This chapter discusses a proposed administrative delegation model applied over Windows 2000 that is based on Role Based Access Control.

In my attempt, a way is suggested to simplify the management of security for using Windows 2000 by applying ARBAC over Windows 2000 and using the capabilities of the Active Directory in Windows 2000 as a product that provides role-based administration. Using role-based administration simplifies the security administration of the organization, eases auditing, eases troubleshooting security problems, simplifies implementation of the organization's security policies, and permits separation of duties. In addition, given the complexity of managing resources within Windows 2000, using a role-based administration tool is recommended to simplify security administration and to reduce the time required to maintain security on the various Windows 2000 objects.

With role-based administration, the tasks that may be delegated are combined into roles. Those roles are then assigned to a user or a group. The notion of groups in Windows 2000 is much like that in Windows NT and that in other operating systems. Rather than setting user and file rights individually for each and every user, the administrator can give rights to various groups, then place users within those groups. Grouping permission(s) into roles promotes consistency in the enterprise by ensuring that each individual is assigned a particular role. It also allows for easier and more consistent permission updates by applying any task changes made in the role to all of the appropriate assignments of that role without the laborious effort of modifying each user's unique permission(s). Moreover, Role-based administration greatly simplifies and reduces the number of steps required to later identify what permission(s) have

been assigned to a user, revoke permission(s), and ensure uniformity of permission(s) granted between users assigned to perform the same tasks.

In what follows, an attempt is described to extend the Windows 2000 group mechanism in two significant ways that are useful in managing group-based access control in large-scale systems. The goal of my attempt is to demonstrate how group hierarchies (where groups include other groups) and decentralized user-group assignment (where administrators are selectively delegated authority to assign certain users to certain groups) can be implemented by means of Microsoft Remote Procedure Call (RPC) programs. In both respects, the experimental goal is to implement previously published models (RBAC96 for group hierarchies and URA97 for decentralized user-group assignment).

6.2 Inside Windows 2000

The result indicates that Windows 2000 as Windows NT has adequate flexibility to accommodate sophisticated access control models to some extent.

Administrative control may be delegated in a Windows 2000 network by creating organizational units (OUs) in the Active Directory. Access control in Windows 2000 Active Directory is based on the concept of the Access Control List (ACL) in Windows NT

6.2.1 Administering Windows 2000 and Active Directory

In a Windows 2000 network, administrative control may be delegated in a highly granular manner to any level of the domain tree by creating organizational units (OUs) and assigning control for specific OUs to particular users or groups. An OU is a container object used to subdivide a domain and to gather related objects within that domain. In turn, Active Directory uses these OUs to provide the logical structure for assigning authority over directory and network resources.

When first implemented, the organization will need to select a model for administering their Active Directory. OU designs will reflect the departmental or geographical boundaries of an organization, or some combination of the two. This is a decision that will need to be made by the organization, as every company has a unique structure and administrative needs.

OUs are not only containers for placing objects, but they also provide the scope for delegating authority. For example, if a departmentally-based OU structure was created, delegation of administrative permission for all user and computer accounts across all locations of a single department could be collected in a single location, such as the Sales Department. However, if the design was geographically based, delegation of administrative permission for all user and computer accounts in all departments could be assigned for a single location, such as North America.

Windows 2000 identifies specific permission(s) and user rights that allow the assignment and restriction of administrative control. Using a combination of OUs, groups and permission(s), an appropriate administrative scope may be defined uniquely for each user. The administrator specifies the user or the group to which control will be given, the OUs and objects to be managed, and the particular administrative capabilities to be assigned. The result is a fine-grained control over who can do what across the network, allowing the administrator to then focus on aspects of the organization other than daily task management.

6.2.2 RBAC, Discretionary Access Control and Windows 2000

RBAC is flexible and powerful enough to simulate Discretionary Access Control (DAC). Both MAC and DAC, are two extremes of Access Control. A previous study (14) has already shown that MAC can be simulated using roles. It was assumed that RBAC is flexible enough that it can accommodate DAC as well (14).

However, in this chapter, the different variations of DAC that are considered in the study for simulation are defined. Then, the operations associated with the creation of an object in RBAC and the operations involved in destroying an object in RBAC are discussed. Afterward, a simulation of DAC variation is given to explain how RBAC can simulate DAC. Finally, revocation of access is discussed.

The central idea behind DAC is that the owner of an object, who is usually its creator, has discretionary authority over whoever can access that object. (21) So, s/he can retain the control over granting access to the objects s/he owns. Owner of an object has discretionary authority over whoever can access that object. In other words DAC policy is owner-based administration of access rights.

DAC policy has many variations, particularly concerning how the owner's discretionary power can be delegated to other users and how access is revoked. (21).

Since Windows 2000 access control in the Active Directory is based on the concept of the Access Control List (ACL) in Windows NT, as was demonstrated in the previous chapter, that every resource has an associated list that specifies the users or groups that have permission to access it. The most permissive access to that resource is the effective one. That concept has been carried over to objects in Active Directory in something called a DACL, or Discretionally Access Control List, which assigns some type of access to the object in question.

Because Windows 2000 is based on DAC and because RBAC can simulate DAC, it is thus feasible to simulate RBAC with Windows 2000.

6.2.3 Problems Facing Administrating Windows 2000

The strength of the Active Directory delegation model and its high degree of granularity is also its greatest challenge. With Active Directory, access to every object may be controlled through permission(s), such as a user account, and every property of an object can be controlled such as the user's password. This level of detail in the Active Directory delegation model introduces the potential for greatly consuming an administrator's time, and it must be managed. My attempt is to deliver the ease of management necessary for assigning and maintaining delegation through its complete role-based management solution.

6.2.3.1 Handling Hierarchy in Windows 2000

A group as previously explained is a collection of users, and it serves as a convenient unit for granting and revoking access.

Every account in Windows 2000's user database contains a group membership list indicating which groups the account belongs to. Users belonging to a group are explicitly displayed with the User Manager program. Windows 2000 is different than Windows NT which notably lacks the facility for including one group in another. In addition, Windows 2000 has the power of using Organizational Units which, as previously explained, are containers used to simulate the hierarchy in an organization. By allowing membership in a group to automatically imply membership in some other group the number of explicit access decisions that need to be made by users and administrators can be reduced. Many commercial database management systems, such as Informix, Oracle and Sybase, provide facilities for hierarchical groups (or

roles) (13). Commercial operating systems, however, provide limited facilities dealing with this purpose. On the other hand, Windows 2000 does not provide an easy way to run and handle the group hierarchy.

Another limitation of both Windows 2000 and Windows NT groups is that membership is exclusively controlled by built-in administrator groups such as Account Operators, Administrators, and Domain Admins. This is a centralized mode which does not scale gracefully to systems with large numbers of groups and users. More generally, it is possible to decentralize user-group assignment by allowing administrators to selectively delegate authority to assign certain users to certain groups.

6.2.3.2 Centralization of Administrating Windows 2000

Windows 2000 as Windows NT centralizes user-group assignment and revocation entirely in the hands of built-in administrator groups. However, this simple approach does not scale to large systems. In the case where the system is spread over a large geographical area, and the administration of the system is centralized in the hand of a system administrator, assigning new users to certain roles in the system is a timely process. Accepting the assignment of users on a temporarily basis to certain roles—as visitors—in a short time is quite difficult. The same is applicable on revoking users from roles which are centralized solely in the hands of system administrators. The ability to immediately stop a user from accessing a role without the interference of the system administrator is almost not possible.

Clearly it is desirable to decentralize user-group assignment to some degree so that expensive system administrators do not need to spend valuable time on routine tasks, and important decisions can be taken away from the direct interference of the system administrators. In particular, administrative groups can be used for this purpose.

6.3 Proposed Solution

RBAC demonstrates that administration based on roles is possible in any organization. In large organizations which have large numbers of roles and users, managing these roles and users in a centralized way is a burdensome task and is delegated to a small group of administrators. Some suitable decentralization of this

administrative task is vital. This will lead to lower administrative cost, less complexity, and fewer errors. This can be achieved through the use of the URA97 model for managing user-role assignment which can decentralize tedious details of RBAC administration without losing central control over overall system policy.

6.3.1 Group Hierarchies

The model for group hierarchies is based on the RBAC96 model for role-based access control discussed earlier in Chapter 2 of this thesis.

URA97 imposes certain restrictions on which users can be added to a role and by whom they can be added. URA97 requires a hierarchy of roles and a hierarchy of administrative roles. The set of roles and administrative roles ought to be disjoint. Senior roles are shown towards the top of the hierarchies, and juniors are to the bottom. Senior roles inherit permission from junior roles.

As shown in previous studies of how group hierarchies can be simulated in Windows NT, the same can be applied to Windows 2000. The basic idea is that when a user is added to a senior group the assigned program automatically adds the user to all junior groups. Similarly, when a user is removed from a senior group the revoke program—in the case of strong revoke—automatically removes the user from all appropriate junior roles. Moreover, a user is said to be an *explicit* member of a group if the user is explicitly designated as a member of the group. A user is an *implicit* member of a group if the user is an explicit member of some senior group. A user can be simultaneously an explicit and implicit member of the same group.

6.3.2 Decentralization of Groups

Sandhu and Bhamidipati (24) introduced the URA97 model for decentralized administration of user-role membership (URA97 stands for user-role assignment 1997). Earlier, in Chapter 3 URA97 was reviewed. In the next section the approach to implementing URA97 in Windows 2000 will be discussed. In Chapter 3 the term group rather than role was used. The description of URA97 is informal and intuitive. However, it is also emphasized that URA97 was defined in earlier work independent of any consideration of its implementation in Windows 2000.

6.3.1.1 User-Group Assignment

There are two issues that need to be addressed in the decentralized management of group membership. First, there is the need to control the groups that an administrative group has authority over. For example, if a member of an administrative role controls the membership in a group therefore it controls the membership in any group junior to that group. Second, it is also important to control which users are eligible for membership in these groups.

URA97 addresses these two issues respectively by means of a group range and a prerequisite group or more generally a prerequisite condition.

6.3.1.2 User-Group Revocation

In classical discretionary access control the source of a permission and the identity of the revoker is typically taken into account in interpreting the revoke operation. These issues are of no importance in the revocation of user-role assignment in RBAC.

URA97 defines two notions of revocation called weak and strong. Weak revocation is straightforward and has impact only on explicit membership in the group in question. Strong revocation requires revocation of both explicit and implicit membership.

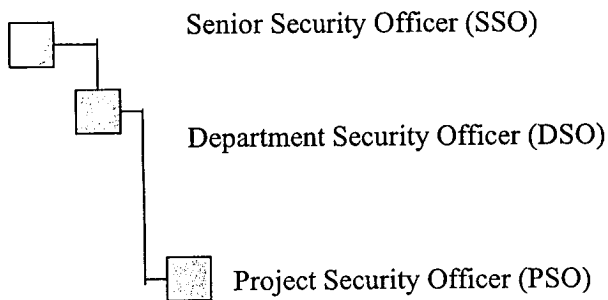
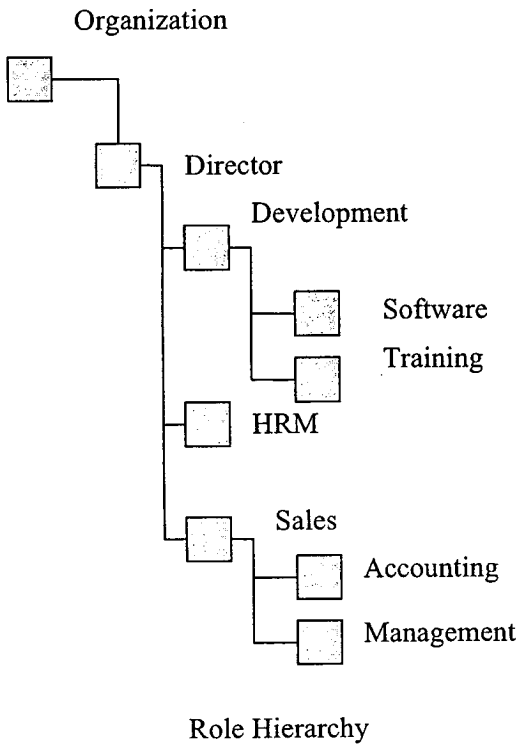
6.4 Implementation of the Proposed Solution

The model for group hierarchies is based on the RBAC96 model for role-based access control mentioned in Chapter 2, while the model for decentralized user-group assignment, called URA97, is adapted from the model previously mentioned in Chapter 3. URA97 distinguishes between regular groups and administrative groups. Neither model was designed with Microsoft RPC programs in mind.

The example used in this experiment is similar to the one previously explained and used in Chapter 3. This example is selected to reflect a kind of real life organization. However, the example used in Chapter 3 can not be adapted as it is and

thus can not be used with Windows 2000 because Windows 2000 does not allow for one child to have more than one parent.

In this thesis, the notion of a role is similar to that of a group, particularly when dealing with the issue of user-role or user-group membership. For this purpose in what follows the concepts of roles and groups are treated as essentially identical.



Administrative Role Hierarchy

Figure 6.1 Role & Administrative Role Hierarchy

6.4.1 The GUI interface

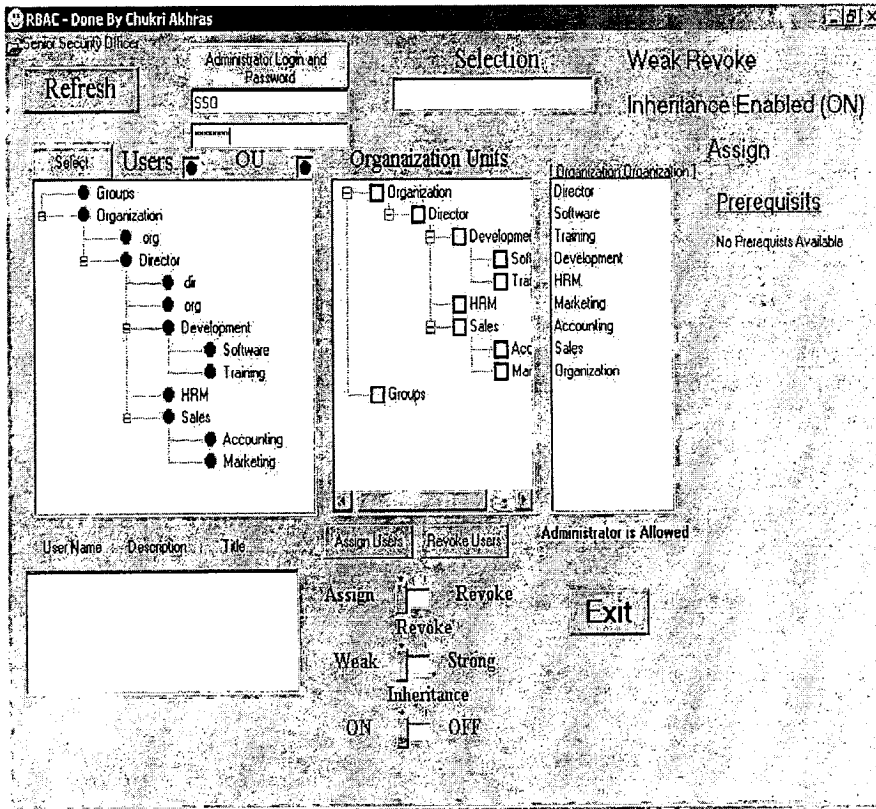


Figure 6-2 RBAC GUI.

In order to make this implementation more convenient, a graphical user interface (GUI) using Visual Basic v6 was developed (Fig 6-2). This interface is used to create a hierarchical listing of the roles and users and to initiate user-group assignment and revocation using some of the command line utilities mentioned above. Microsoft RPC is used to enforce desired behavior of URA97 with respect to different administrative groups. The RPC mechanism is the simplest way to implement client-server applications because it keeps the details of network communications out of the application code. The security of RPC is part of the operating System that uses it. Therefore, Microsoft RPC on Windows 2000 can use the Windows 2000 security built in as part of the operating system. To implement URA97 in Windows 2000 several batch and text reference files are also used.

According to the applied Windows 2000 security settings, only authorized users with appropriate administrative rights can use this application and run the

program RBAC.exe. All other users can see the batch files and text files used in the process, but only the authorized ones can modify them.

6.4.2 The RCP and Batch Files Used

Tables 6.1, 6.2 will list the set of batch files created and used during the execution of the RBAC.exe program and the associated text files containing the results obtained from executing the batch files.

Batch File	Action	Output
Constraint.bat	List all users belonging to a certain Role	Constraint.txt
Groups_Mod.bat	Add a user to a Senior Role and its Juniors (New user)	
Groups_Mod_IO.bat	Add a user to a Senior Role only	
Groups_Sel.bat	Show Groups authorized to a certain administrative role	Group_Select.TXT
LGroups2.bat	Add a user to Junior Roles	
LOUs.bat	List all OUs	LOUs.txt
LUsers.bat	List all Users	LUsers.txt
User_exist.bat	Check if a user exist	User_exist.txt
User_group.bat	List the Roles in each OU	User_group1.txt
Wrevoke.bat	Weak revoke a user from a Role	

Table 6.1 Batch Files

TXT File	Action
Constraint.txt	List all users belonging to a certain Role
Assigned.txt	Contains all administrative roles for assigning users, range of authority, prerequisites, and constraints.
Group_Assigned.txt	Contains the names of all the OUs
Group_Select.txt	Show Groups authorized to a certain administrative role
Revoked.txt	Contains all administrative roles for revoking users, range of authority, prerequisites, and constraints.
LOUs.txt	List all OUs in dns format
LUsers.txt	List all Users in dns format
User_exist.txt	Check if a user exist
User_group.txt	List the Roles in each OU
Remove.txt	Name of the user to be removed
Users.txt	Name of the users to be created

Table 6.2 Text Files

Windows 2000 has many command-line utilities to deal with the different issues related to Organizational Units, Groups and Users. These command-line

utilities are used to display, change, create or modify any Organizational Unit, Group or User. Table 6.3 lists these commands.

Command	Action
Dsget	Displays the selected attributes of a computer, group, organization unit, server, or a user in a directory.
Dsadd	This tool adds a computer, group, organization unit, or a user to a directory.
Dsmmod	This tool modifies an existing user, computer, group, or organizational units in a directory.
Dsmove	This tool moves any object from its current location in the directory to a new location and renames an object without moving it in the directory tree.
Dsqquery	This tool queries and finds a list of computers, groups, organizational units, servers, or users in the directory by using specific search criterion.
Dsrm	This tool deletes an object of a specific type or any general object from the directory.

Table 6.3 Command-Line Tools for Active Directory

6.4.3 Running The Implementation

What is needed before starting to execute the application is that the Administrator creates a two text files called Assigned.txt and Revoked.txt in which s/he includes the login names and names of the allocated administrators, the range of allocation, and flags for strong revoke, disable inheritance, and the range of prerequisites.

After selecting the kind of action required, either revoke or assign, and after a successful authentication, the role hierarchy is shown in two separate windows: the first one will show the different groups (roles) along with the users belonging to those groups; the second window will show the list of groups only with the possibility of selection of a role. A third window will show the set of groups that the logged-in administrator has the right to assign to, or revoke from depending on his/her rights.

The system will automatically find all the junior roles for the main role called Organization, and it will find all related users assigned to those roles.

Different OUs were created in a way to simulate the hierarchical setup of the example used. Within those OUs, global security groups were created with names the same as the OUs names. The root OU in this example is called Organization.

In a separate OU called Groups, different domain groups were created: Shut Down group, Log in locally group, and Change System Time group, those groups will

assign the permission(s) of shutting down the system, logging in locally and changing the system time to any of their members.

Some of the OUs were assigned to one, some or all of the permission groups. Now any user member of the assigned OUs will be a member of that role and will have the privilege to access the assigned role.

To simulate group hierarchy, information about explicit and implicit membership in account database is used. The implementation of group hierarchy by explicitly assigning a member of a senior group to be a member of all junior groups in account database may raise a scalability issue. For example, many Unix implementations limit the number of groups activated in a process to a fairly small number such as 32 or 16, so this approach does not scale for Unix. A small experiment is conducted to ascertain how many groups can be activated in a process on Windows NT. The experiment indicated that Windows NT can accommodate up to 993 groups simultaneously activated in a single process. Since Windows 2000 is a more advanced operating system than Windows NT, it thus has better capabilities. The resulting number is more than sufficient for the purpose of our experiment and so large group hierarchies can be accommodated by means of this approach (19).

6.4.4 Implementation of the URA97 Model on Windows 2000

In this section the implementation is illustrated by using examples of the role hierarchy and the administrative role hierarchy of Figure 6.1.

Figure 6.3 clearly shows decentralized administration works in the URA97. Administrative permissions are associated with administrative roles, and administrators are assigned to decentralized administrative roles.

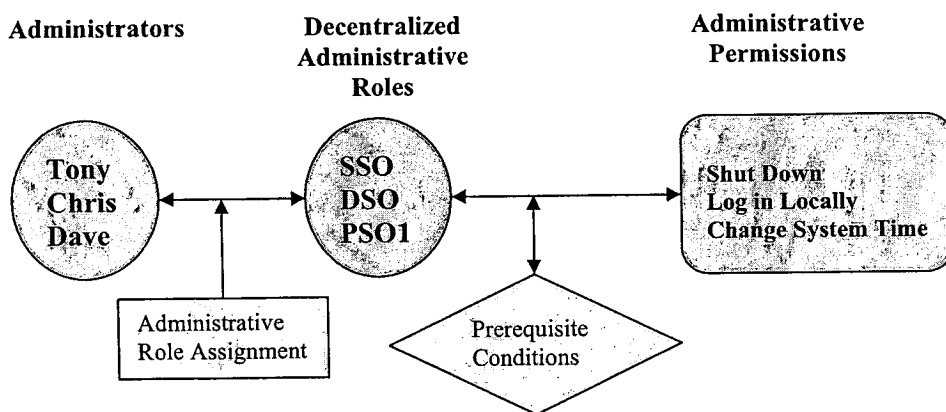


Figure 6.3: Decentralized Administration of URA97

Suppose an employee named Tony is assigned to the SSO role and wants to access the system. When he requests access, the system requires the login name and password for authentication. After the system checks his information, it shows all the roles available to him based on his assigned roles and role hierarchy of the system.

Because URA97 supports multiple administrative roles, each administrative role can have different role ranges and prerequisite conditions. Tony can have SSO, DSO and PSO as his available administrative roles. Since he is assigned to SSO, which is the highest role in the administrative role hierarchy, then, he can activate any of the junior administrative roles as a session.

6.4.4.1 *Can_Assign Relation*

Administrative Role	Prerequisite Condition	Role Range
SSO	Organization	[Org, Org]
DSO1	Software \wedge Development	[Dev, Dev]
PSO1	Software	[Tra, Sof]
MAR1	Sales \wedge Marketing	(Acc, Acc)
SAL	Sales	[Sal, Sal]
HRM	Organization	[HRM, HRM]

Table 6.4: Example of *can_assign*

URA97 has a *can_assign* relation which is stored in the file Assigned.txt. An example of the file Assigned.txt is shown in Table 6.5.

Assignment of a user to a group in URA97 means explicit assignment. Implicit assignment to junior groups happens as a consequence and as a side-effect of the explicit assignment. In other words Assigned.txt applies only to explicit membership.

The structure of the file Assigned.txt as shown in Table 6.5 is as follows:

Login code : User's Title : Range of Assignment : * : Can use Inheritance :
 Constrains and Prerequisites.

In case there is no constraints or prerequisites, a star is used, otherwise, a (+) sign indicates a prerequisite, or the user should exist in the indicated role prior to be assigned to the new role. A (-) sign indicates a constrain, where the user can not exist

in the indicated role and be assigned to the new role. As can be seen, the closed brackets [] are used to indicate inclusive membership in the assigned range, while the parenthesis () indicates exclusive membership in the assigned range.

```
SSO:Senior Security Officer:[Organization,Organization]:Y:Y:*:
DSO1:Department Security Officer:[Development,Development]:Y:Y:+Software,-Training:
DSO4:Department Security Officer:[Development,Software):Y:Y:*:
DSO3:Department Security Officer:[Organization,Development):Y:Y:*:
DSO2:Department Security Officer:[HRM,HRM]:Y:Y:*:
PSO1:Project Security Officer 1:[Training, Software]:Y:Y:*:
PSO2:Project Security Officer 2:[Development,HRM]:N:N:*:
MAR1:Marketing Security Officer 1:(Accounting,Accounting):N:Y:+Sales,-Marketing:
MAR2:Marketing Security Officer 2:[Marketing,Sales]:N:Y:*:
SAL:Sales Security Officer:[Sales,Sales]:Y:Y:*:
HRM1:Human Resource Manager:[HRM,HRM]:Y:N:*:
```

Table 6.5 the file Assigned.txt

Suppose a member of the administrative role SSO, Michel wants to assign Elie, who has only a regular role in the Accounting role currently, to Director role based on the prerequisite conditions. According to the SSO authorities, the administrative role SSO can add users in Accounting to Director.

6.4.4.2 Can-Revoke Relation

URA97 authorizes revocation by the *can_revoke* relation which is stored in the Revoked.txt file. An example is shown in table 6.6. The meaning of each row in Revoked.txt is that a member of the administrative group can revoke membership of a user from any regular group in group range. It would be acceptable to expect some correlation between the range authorized for an administrative group in Assigned.txt and in Revoked.txt, but this is not required by the model.

The structure of the file Revoked.txt as shown in Table 6.6 is as follows:

Login code : User's Title : Range of Revoke : Can use Strong Revoke : * :
 Constrains and Prerequisites.

In case there is no constraints or prerequisites, a star is used. As can be seen, the closed brackets [] are used to indicate inclusive membership in the assigned range, while the parenthesis () indicates exclusive membership in the assigned range.

SSO:Senior Security Officer:[Organization,Organization]:Y:Y:*

HRM1:Human Resource Manager:[HRM,HRM]:Y:N:*

Table 6.6 the file Revoked.txt

6.4.4.3 Prerequisite Conditions and Constraints

Another point to be noticed in Table 6.5 is that for the administrator DSO1 to assign a user to the role Development and all its junior roles, a prerequisite condition should exist which is that the user should belong first to the Software role and should not belong to the role Training. This prerequisite condition is very important in RBAC along with the concept of constraints where overlapping of permission(s) in an organization would become a real problem.

Most of role-based constraints work focuses on separation of duty constraints which is a fundamental issue in computer security for prevention of fraud and errors (8). In my example, MAR1 assigning of roles is controlled by the condition that a user can not be working at the same time in both the Sales department and the Marketing department.

6.5 Conclusion

In this thesis an experiment to provide two useful extensions to the Windows 2000 group mechanism is described. First by adding hierarchical groups by means of explicit assignment to junior groups. When a user is assigned to a senior group the system automatically adds the user to all junior groups. Similarly, when a user's membership is revoked from a group, revocation from appropriate junior groups is automatically carried out. This behavior is adapted from the RBAC96 model. Secondly, the URA97 model was adapted for decentralized user-group assignment and implemented in Windows 2000. The implementation uses Microsoft RPC programs to enforce authorization to add and remove users from groups. The

obtained results indicate that Windows 2000 has adequate flexibility to accommodate sophisticated access control models. It was also noted that Windows 2000 has a good scalability in simulating group hierarchy by explicit assignment to junior groups.

CHAPTER 7

CONCLUSION

7.1 Conclusion

From what has been discussed in this thesis and from central notion of Role-Based Access Control (RBAC) it can be noticed that users do not have discretionary access to enterprise objects (5). Instead, access permission(s) are administratively associated with roles, and users are administratively made members of appropriate roles. This idea greatly simplifies the management of authorization while providing an opportunity for greater flexibility in specifying and enforcing organization-specific protection policies. Users can be made members of roles as determined by their responsibilities and qualifications and can be easily reassigned from one role to another without modifying the underlying access structure. Roles can be granted new permission(s) as new applications and actions are incorporated, and permission(s) can be revoked from roles as needed.

Some users and vendors have recognized the importance and benefits of RBAC. Some RBAC features have been implemented in many commercial products without a frame of reference as to the functional makeup and virtues of RBAC (13).

Although RBAC is a good model, the administration of RBAC including building and maintaining access control information remains a difficult problem in large companies (27). Decentralization of administration could be a solution to ease the administration of large companies.

In this thesis a method to allow security administrators in Windows 2000 environment is suggested to manage access control by GUI model. This is done by means of Microsoft remote procedure call (RPC) programs. First by adding hierarchical groups, when a user is added to a senior group, the system adds the user automatically to all related junior groups. Similarly, when a user's membership is revoked from a group, revocation from appropriate junior groups is done automatically. Secondly, URA97 model is used for decentralized user-group assignment and implemented it in Windows 2000 (20). Microsoft RPC programs are used for this implementation to enforce authorization to add and remove users from groups. A major result of this experiment is that Windows 2000 has an adequate

flexibility to simulate RBAC and that it has a good scalability in group hierarchies. Another result is that separation of duty constraints can be determined in Windows 2000 by the assignment of individuals to groups at user-group assignment time.

7.2 Recommendation for Future Work

The positive results that were obtained from the study and the application that was build to implement RBAC in this thesis are encouraging to give future thinking about going further in applying ARBAC on Windows 2000.

The work was centered on the decentralization of assigning users to roles and on role hierarchy. Future work can look at the decentralization of assigning permissions to roles, and assigning roles to roles. In this way RBAC97 can be fully implemented on Windows 2000.

In the application, only three predefined roles were used, and users were assigned to those roles. In future work, the application can be extended to include the possibility to decentralize the process of creating roles, and the process of assigning permissions to those roles.

In the application, the consequences of assigning a user to a role applies to URA97 where the user is authorized to use the permission(s) of that role and its juniors and the user becomes eligible for further assignment to other roles by appropriate administrative roles. This will lead to the discussion of the mobility and Immobility in RBAC99 where a user's membership in a role can be mobile or immobile.

Future work can be to try to implement RBAC on new releases of Windows, such as Windows 2003, which has more flexible and powerful way in administration.

REFERENCES

1. Barkley, Kuhn, Rosenthal, Skall, and Cincotta. "Role-Based Access Control for the Web". 1998. *CALS Expo International & 21st Century Commerce 1998: Global Business Solutions for the New Millennium*. URL: <http://hissa.ncsl.nist.gov/rbac/cals-paper.html> 11/1/00.
2. Barkley. "Implementing Role Based Access Control Using Object Technology". *First ACM Workshop on Role-Based Access Control*. 1995 URL: <http://hissa.ncsl.nist.gov/rbac/rbacot/titlewkshp.html> 11/1/00
3. E. Lupu, D. Marriott, M. Sloman, and N. Yialelis. "A Policy Based Role Framework for Access Control ACM/NIST Workshop on Role-Based Access Control". 1995. URL: <http://www.doc.ic.ac.uk/~ecl1/publications/rbac95.pdf>
4. Ferraiolo, D. F. and Kuhn. "Role Based Access Controls", *15th National Computer Security Conference*, 1992. URL: <http://hissa.ncsl.nist.gov/rbac/paper/rbac1.html>, 11/1/00.
5. Ferraiolo, D. F., Cugini, J. A. and Kuhn, D. R. "Role-Based Access Control (RBAC): Features and Motivations", *11th Annual Computer Security Applications Proceedings*. 1995. URL: <http://hissa.ncsl.nist.gov/rbac/newpaper/rbac.html>
6. Ferraiolo, D.F., Sandhu, R.S., Gavrilu, S., Kuhn, D. R. and Chandramouli, R. "Proposed NIST Standard for Role-Based Access Control". *ACM Transaction on Information and System Security*, Vol. 4, No. 3, August 2001, pp. 224-274.
7. Gail, J. A. and Sandhu, R.S. "Towards Role-Based Administration in Network Information Services". *Journal of Network and Computer Applications*, Vol. 22, 1999, pp. 199-213. Online: <http://www.idealibrary.com>
8. Gail-Joon, A. and Kwangjo, K. "CONUGA: Constrained User Group Assignment", *Journal of Network and Computer Applications*, 2001, 25.
9. Goh, C. and Baldwin, A. "Towards a More Complete Model of Role". Extended Enterprise Lab, Hewlett Packard Laboratories. 1998. Bristol. United Kingdom. URL: <http://www.hpl.hp.com/techreports/98/HPL-98-92.pdf>
10. *Implementing and Administrating Microsoft ® Windows ® 2000 Directory Services*. Microsoft Official Curriculum.
11. Microsoft Windows 2000 Server, White Paper, Posted July 26, 2000. <http://www.microsoft.com/windows2000/library>
12. NIST CSL. "An Introduction to Role Based Access Control". URL: <http://csrc.ncsl.nist.gov/nistbul/csl95-12.txt> 11/1/00
13. Oracle Corporation, *ORACLE 7 Server SQL Language Reference Manual*, December 1992.

References

14. Osborn, S., Sandhu, R.S. and Qamar, M. "Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies", The University of Western Ontario and George Mason University, ACM Publications.
15. Sandhu, R.S. "Rational for the RBAC96 Family of Access Control Models". *ACM Workshop*, 1996.
16. Sandhu, R.S. "Report on the First ACM Workshop on Role-Based Access Control", Gaithersburg, Maryland. Dec, 1, 1995. George Mason University and SETA Corporation. March 1996. URL: <http://www.chacs.itd.nrl.navy.mil/ieee/cipher/old-conf-rep/conf-rep-RBAC96.html> . 11/1/00
17. Sandhu, R.S. "Role-Based Access Control", Laboratory for Information Security Technology, George Mason University, Fairfax, VA 22030, <http://www.isse.gmu.edu/faculty/sandhu>
18. Sandhu, R.S. and Bhamidipati, V. "Role-Based Administration of User-Role Assignment: The URA97 Model and its Oracle Implementation".
19. Sandhu, R.S. and Gail-Joon, A. "Group Hierarchies with Decentralized User Assignment in Windows NT", *Journal of Systems and Software*, Elsevier Science, 2000, George Mason University, ACM Publications.
20. Sandhu, R.S. and Park, S. J. "Decentralized User-Role Assignment for Web-based Intranets". *3rd ACM Workshop on role-Based Access*, ACM. 1998.
21. Sandhu, R.S. and Qamar, M. "How to do Discretionary Access Control Using Roles". *ACM Workshop*, 1998.
22. Sandhu, R.S. and Qamar, M. "The ARBAC99 Model for Administration of Roles", Laboratory for Information Security Technology, George Mason University, Fairfax, VA 22030, <http://www.list.gmu.edu>
23. Sandhu, R.S. and Qamar, M. "The RRA97 Model for Role-Based Administration of Role Hierarchies", Laboratory for Information Security Technology, George Mason University, Fairfax, VA 22030, <http://www.list.gmu.edu>
24. Sandhu, R.S., Bhamidipati, V. and Qamar, M. "The ARBAC97 Model for Role-Based Administration of Roles", *ACM Transactions on Information and System Security*, Vol. 2, No. 1, February 1999, pp. 105-135
25. Sandhu, R.S., Bhamidipati, V., Coyne, E., Ganta, S., and Yourman, C. "The ARBAC97 Model for Role-Based Administration of Roles: Preliminary Description and Outline", Laboratory for Information Security Technology, George Mason University, Fairfax, VA 22030, <http://www.list.gmu.edu>

References

26. Sandhu, R.S., Coyne, E.J., Feinstein, H., and Youman, C. "Role-Based Access Control Models". *IEEE Computer*. Vol. 29. No. 2. Feb 1996.
27. Sejong, O. and Seog, P. "An Improved Administration Method on Role-Based Access Control in the Enterprise Environment". *Journal of Information Science and Engineering*, Vol. 17, 2001, pp. 921-944.
28. Sun Microsystems Inc. "Secure Enterprise Computing with the Solaris 8 Operating System". *A Technical White Paper*. Chapter 4. URL: <http://www.sun.com/software/white-papers/wp-s8security/wp-s8security.pdf>
11/1/00