Role-Based Access Control*

Ravi S. Sandhu†

Laboratory for Information Security Technology
ISSE Department, MS 4A4
George Mason University
Fairfax, VA 22030

voice: 703-993-1659
fax: 703-993-1638
email: sandhu@isse.gmu.edu
http://www.isse.gmu.edu/faculty/sandhu

September 17, 1997

Abstract The basic concept of role-based access control (RBAC) is that permissions are associated with roles, and users are made members of appropriate roles thereby acquiring the roles' permissions. This idea has been around since the advent of multi-user computing. Until recently, however, RBAC has received little attention from the research community. This article describes the motivations, results and open issues in recent RBAC research.

The article focuses on four areas. Firstly, RBAC is a multi-dimensional concept that can range from very simple at one extreme to quite complex and sophisticated at the other. This presents problems in coming up with a definitive model of RBAC. We see how this impasse is resolved by having a family of models which can accommodate all these variations. Secondly, we discuss how RBAC can be used to manage itself. Recent models developed for this purpose are presented. Thirdly, the flexibility of RBAC can be demonstrated in many ways. Here we show how RBAC can be configured to enforce different variations of classical lattice-based mandatory access controls. Fourthly, we describe a conceptual three-tier architecture for specification and enforcement of RBAC. The article concludes with a discussion of open issues in RBAC.

---

*Portions of this article have been published earlier in [SCFY96, San96, SB97, SBC+97, SF94].

†Ravi Sandhu is also affiliated with SETA Corporation, 6862 Elm Street, McLean, VA 22101

Contents

List of Figures

1 INTRODUCTION

The concept of role-based access control (RBAC) began with multi-user and multi-application on-line systems pioneered in the early 1970s. The central notion of RBAC is that permissions are associated with roles, and users are assigned to appropriate roles. Roles are created for the various job functions in an organization and users are assigned roles based on their responsibilities and qualifications. Users can be easily reassigned from one role to another. Roles can be granted new permissions as new applications and systems are incorporated, and permissions can be revoked from roles as needed.

This basic idea has been around in one form or another for a long time. Yet, it has received surprisingly little attention from the research community until recently. This article describes the motivations, results and open issues in recent RBAC research.

This section introduces RBAC and discusses general issues related to it. In section 2 we show that RBAC is a multi-dimensional concept ranging from very simple at one end to quite sophisticated at the other. This makes it difficult to construct a single definitive model of RBAC. Instead we describe a family of models which can accommodate all these variations. In section 3 we discuss how RBAC can be used to manage itself. In section 4 we demonstrate the flexibility and power of RBAC by showing how it can be configured to enforce different variations of classical lattice-based mandatory access controls. Section 5 describes a conceptual three-tier architecture for specification and enforcement of RBAC. Section 6 concludes the article with a brief discussion of open issues in RBAC.

1.1 Motivation and Background

A recent study by the US National Institute of Standards and Technology (NIST) [FGL93] demonstrated that RBAC addresses many needs of the commercial and government sectors. In this study of 28 organizations, access control requirements were found to be driven by a variety of concerns including customer, stockholder and insurer confidence, privacy of personal information, preventing unauthorized distribution of financial assets, preventing unauthorized usage of long-distance telephone circuits, and adherence to professional standards. The study found that many organizations based access control decisions on "the roles that individual users take on as part of the organization." Many organizations preferred to centrally control and maintain access rights, not so much at the system administrator's personal discretion but more in accordance with the organization's protection guidelines. The study also found that organizations typically viewed their access control needs as unique and felt that available products lacked adequate flexibility.

Other evidence of strong interest in RBAC comes from the standards arena. Roles are being considered as part of the emerging SQL3 standard for database management systems, based on their implementation in Oracle 7. Roles have also been incorporated in the commercial security profile of the Common Criteria [Com96]. There are ongoing efforts by NIST to provide standards and guidance for RBAC.

RBAC is also well matched to prevailing technology and business trends. A number of products support some form of RBAC directly, and others support closely related concepts, such as user groups, that can be utilized to implement roles.

Many commercially successful access control systems for mainframes implement roles for security administration. For example, an operator role can access all resources but not change access permissions, a security-officer role can change permissions but have no access to resources, and an auditor role can access audit trails. This administrative use of roles is also found in modern network operating systems, e.g., Novell's NetWare and Microsoft Windows NT.

Recent resurgence of interest in RBAC has focussed on general support of RBAC at the application level. Specific applications have been, and are being, built with RBAC encoded within the application itself. Existing operating systems and environments provide little support for application-level use of RBAC. Such support is beginning to emerge in some products. The challenge is to identify application-independent facilities that are sufficiently flexible, yet simple to implement and use, to support a wide range of applications with minimal customization.

Sophisticated variations of RBAC include the capability to establish relations between roles as well as between permissions and roles and between users and roles. For example, two roles can be established as mutually exclusive, so the same user is not allowed to take on both roles. Roles can also take on inheritance relations, whereby one role inherits permissions assigned to a different role. These role-role relations can be used to enforce security policies that include separation of duties and delegation of authority. Heretofore, these relations would have to be encoded into application software; with RBAC, they can be specified once for a security domain.

With RBAC it is possible to predefine role-permission relationships, which makes it simple to assign users to the predefined roles. The NIST study cited above indicates that permissions assigned to roles tend to change relatively slowly compared to changes in user membership of roles. The study also found it desirable to allow administrators to confer and revoke membership to users in existing roles without giving these administrators authority to create new roles or change role-permission assignments. Assignment of users to roles will typically require less technical skill than assignment of permissions to roles. It can also be difficult, without RBAC, to determine what permissions have been authorized to what users.

Access control policy is embodied in various components of RBAC such as role-permission, user-role and role-role relationships. These components collectively determine whether a particular user will be allowed to access a particular resource or piece of data. RBAC components may be configured directly by the system owner or indirectly by appropriate administrative roles as delegated by the system owner. The policy enforced is the net result of the precise configuration of various RBAC components as directed by the system owner. Moreover, the access control policy will evolve incrementally over the system life cycle. The ability to modify policy to meet the changing needs of an organization is an important benefit of RBAC.

Although RBAC is policy neutral, it directly supports three well-known security principles: *least privilege*, *separation of duties* and *data abstraction*. Least privilege is supported because RBAC can be configured so only those permissions required for the tasks conducted by members of the role are assigned to the role. Separation of duties is achieved by ensuring that mutually exclusive roles must be invoked to complete a sensitive task, such as requiring an accounting clerk and account manager to participate in issuing a check. Data abstraction is supported by means of abstract permissions such as credit and debit for an account object, rather than the read, write, execute permissions typically provided by the operating system. However, RBAC cannot enforce application of these principles. The security officer can configure RBAC so it violates these principles. Also, the degree to which data abstraction is supported will be determined by the implementation details.

1.2 RBAC Limitations

As a cautionary note it is important to emphasize that RBAC is not a panacea for all access control issues. More sophisticated forms of access control are required to deal with situations where sequences of operations need to be controlled. For example, a purchase requisition requires various steps before it can lead to issuance of a purchase order. RBAC does not attempt to directly control the permissions for such a sequence of events. Other forms of access control can be layered on top of RBAC for this purpose. Mohammed and Dilts [MD94], Sandhu [San88, San91] and Thomas and Sandhu [TS94, TS97] discuss some of these issues. We view control of sequences of operations to be outside the direct scope of RBAC, although RBAC can be a foundation on which to build such controls.

1.3 What is a Role?

A role is properly viewed as a semantic construct around which access control policy is formulated. The particular collection of users and permissions brought together by a role is transitory. The role is more stable because an organization's activities or functions usually change less frequently.

There can be several distinct motivations for constructing a role, including the following. A role can represent competency to do specific tasks, such as a physician or a pharmacist. A role can embody authority and responsibility, e.g., project supervisor. Authority and responsibility are distinct from competency. Alice may be competent to head several departments, but is assigned to head one of them. Roles can reflect specific duty assignments that are rotated through multiple users, e.g., a duty physician or shift manager. RBAC models and implementations should be able to conveniently accommodate all of these manifestations of the role concept.

The concept of a role originated in organizational theory much before the advent of computerized information systems. Even in the context of modern information systems roles have significance beyond their application in security and access control. From the perspective of RBAC it is therefore important to distinguish the concept and scope of a role for access control purposes as opposed to the more general organizational context in which roles arise. Roles have bigger significance than access control but we should not be tempted to expand the access control arena as a consequence. Instead we should focus on aspects of roles that are relevant from the access control perspective. This question does impact activities such as the design of roles which may need to take the bigger picture into account even though the immediate focus is on roles for access control purposes.

1.4 Roles versus Groups

A frequently asked question is, "What is the difference between roles and groups?" Groups of users as the unit of access control are commonly provided in many access control systems. A major difference between most implementations of groups and the concept of roles is that groups are typically treated as a collection of users and not as a collection of permissions. A role is both a collection of users on one side and a collection of permissions on the other. The role serves as an intermediary to bring these two collections together.[1]

Consider the Unix operating system. Group membership in Unix is defined in two files, `/etc/passwd` and `/etc/group`. It is thus easy to determine the groups to which a particular user belongs or all the members of a specific group. Permissions are granted to groups on basis of permission bits associated with individual files and directories. To determine what permissions a particular group has will generally require a traversal of the entire filesystem tree. It is thus much easier to determine the membership of a group than to determine the permissions of the group. Moreover the assignment of permissions to groups is highly decentralized. Essentially, the owner of any sub-tree of the Unix filesystem can assign permissions for that sub-tree to a group. (The precise degree to which this can be done depends on the particular variant of Unix in question.) However, Unix groups can be used to implement roles in certain situations, even though groups are not the same as our concept of roles.

To illustrate the qualitative nature of the group versus role distinction, consider a hypothetical system in which it takes twice as long to determine group membership as to determine group permissions. Assume that group permissions and membership can only be changed by the system security officer. In this case, the group mechanism would be very close to our concept of a role.

The preceding discussion suggests two characteristics of a role, it should be approximately equally easy to determine role membership and role permissions, and control of role membership and role permissions should be relatively centralized in a few users. Many mechanisms that are claimed to be role-based fail one or both of these requirements.

Groups are also an established concept with a generally well-understood meaning much like other well-known concepts such as a directory. Although groups can be extended to provide the same features as roles, it is better to coin a new term to avoid confusion with the existing concept of a group. Moreover, groups are a useful concept without being extended to roles. It is therefore better to keep these two concepts separated.

2 THE RBAC96 MODELS

Notwithstanding the recognized usefulness of the RBAC concept, there has been little agreement on what RBAC means. As a result RBAC is an amorphous concept interpreted in different ways, ranging from simple to elaborate and sophisticated.

[1] It should be mentioned that sometimes a role is defined to be a collection of permissions [Bal90, Gui95, Not97]. We will see in section 3 that, for administrative purposes, it is actually beneficial to distinguish three kinds of roles respectively containing only users, only permissions and both users and permissions.

To understand the various dimensions of RBAC we define a family of four conceptual models. $RBAC_0$, the base model, specifies the minimum requirement for any system that fully supports RBAC. $RBAC_1$ and $RBAC_2$ both include $RBAC_0$, but add independent features to it. They are called advanced models. $RBAC_1$ adds the concept of role hierarchies (situations where roles can inherit permissions from other roles). $RBAC_2$ adds constraints (which impose restrictions on acceptable configurations of the different components of RBAC). $RBAC_1$ and $RBAC_2$ are incomparable to one another. The consolidated model, $RBAC_3$, includes $RBAC_1$ and $RBAC_2$ and, by transitivity, $RBAC_0$. We refer to this family of models as RBAC96 (for RBAC '96). The relationship between the four models of RBAC96 is shown in figure 1(a) and the consolidted model $RBAC_3$ is portrayed in figure 1(b).

These models provide a guideline for development of products and their evaluation by prospective customers. For the moment, we assume there is a single security officer who is the only one authorized to configure the various sets and relations of these models. We will introduce a more sophisticated management model in section 3.

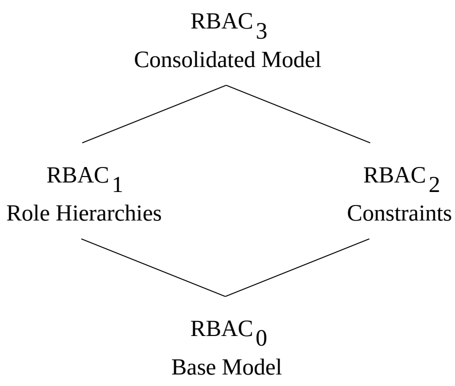
2.1 Base Model: $RBAC_0$

$RBAC_0$ consists of that part of figure 1(b) not identified with one of the three advanced models, that is, it omits the role hierarchy and constraints. There are three sets of entities called users (U), roles (R), and permissions (P). The diagram also shows a collection of sessions (S).

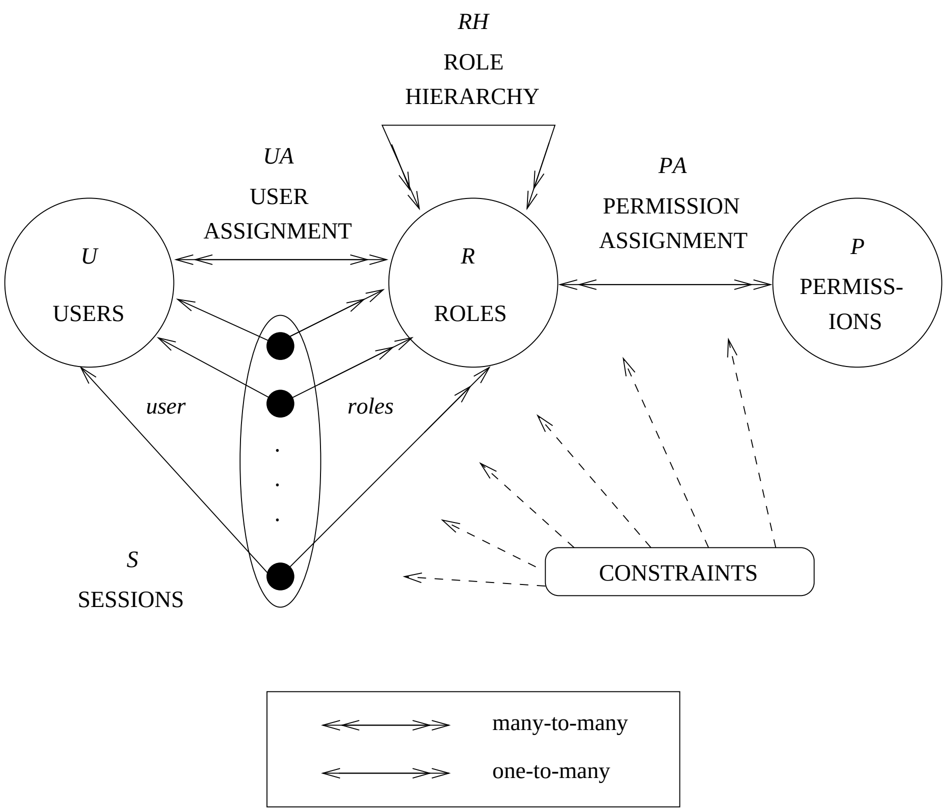
A *user* in this model is a human being. The concept of a user can be generalized to include intelligent autonomous agents such as robots, software agents, immobile computers, or even networks of computers. For simplicity, we focus on a user as a human being. A *role* is a job function or job title within the organization with some associated semantics regarding the authority and responsibility conferred on a member of the role.

A *permission* is an approval of a particular mode of access to one or more objects in the system. The terms authorization, access right and privilege are also used in the literature to denote a permission. Permissions are always positive and confer the ability to the holder of the permission to perform some action(s) in the system. Objects are data objects as well as resource objects represented by data within the computer system. Our conceptual model permits a variety of interpretations for permissions, from very coarse grain, e.g., where access is permitted to an entire subnetwork, to very fine grain, where the unit of access is a particular field of a particular record. Some access control literature talks about "negative permissions" which deny, rather than confer, access. In our framework denial of access is modeled as a constraint rather than a negative permission.

The nature of a permission depends on the implementation details of a system and the kind of system that it is. A general model for access control must therefore treat permissions as uninterpreted symbols to some extent. Each system protects objects of the abstraction it implements. Thus an operating system protects such entities as files, directories, devices, and ports, with operations such as read, write, and execute. A relational database management system protects relations, tuples, attributes, and views, with operations such as SELECT, UPDATE, DELETE, and INSERT. An accounting application



(a) Relationship among RBAC96 models



(b) The $RBAC_3$ model

Figure 1: The RBAC96 Family of Models

protects accounts and ledgers with operations such as debit, credit, transfer, create-account, and delete-account. It should be possible to assign the credit operation to a role without being compelled to also assign the debit operation to that role. Note that both operations require read and write access to the operating system file that stores the account balance.

Permissions can apply to single objects or to many. For example, a permission can be as specific as read access to a particular file or as generic as read access to all files belonging to a particular department. The manner in which individual permissions are combined into a generic permission so they can be assigned as a single unit is highly implementation dependent.

Figure 1(b) shows *user assignment* (UA) and *permission assignment* (PA) relations. Both are many-to-many relations. A user can be a member of many roles, and a role can have many users. Similarly, a role can have many permissions, and the same permission can be assigned to many roles. The key to RBAC lies in these two relations. Ultimately, it is a user who exercises permissions. The placement of a role as an intermediary to enable a user to exercise a permission provides much greater control over access configuration and review than does directly relating users to permissions.

Each *session* is a mapping of one user to possibly many roles, i.e., a user establishes a session during which the user activates some subset of roles that he or she is a member of. The double-headed arrow from the session to R in figure 1(b) indicates that multiple roles are simultaneously activated. The permissions available to the user are the union of permissions 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 in figure 1(b). 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. This feature of RBAC_0 supports the principle of least privilege. A user who is a member of several roles can invoke any subset of these that is suitable for the tasks to be accomplished in that session. Thus, a user who is a member of a powerful role can normally keep this role deactivated and explicitly activate it when needed. We defer consideration of all kinds of constraints, including constraints on role activation, to RBAC_2. So in RBAC_0 it is entirely up to the user's discretion as to which roles are activated in a given session. RBAC_0 also permits roles to be dynamically activated and deactivated during the life of a session. The concept of a session equates to the traditional notion of a *subject* in the access control literature. A subject (or session) is a unit of access control, and a user may have multiple subjects (or sessions) with different permissions active at the same time.

The following definition formalizes the above discussion.

Definition 1 The RBAC_0 model has the following components:

- U, R, P, and S (users, roles, permissions and sessions respectively),
- $PA \subseteq P \times R$, a many-to-many permission to role assignment relation,
- $UA \subseteq U \times R$, a many-to-many user to role assignment relation,

- $user : S \to U$, a function mapping each session s_i to the single user $user(s_i)$ (constant for the session's lifetime), and
- $roles : S \to 2^R$, a function mapping each session s_i to a set of roles $roles(s_i) \subseteq \{r \mid (user(s_i), r) \in UA\}$ (which can change with time) and session s_i has the permissions $\cup_{r \in roles(s_i)} \{p \mid (p, r) \in PA\}$. □

We expect each permission and user to be assigned to at least one role.

It is possible for two roles to be assigned exactly the same permissions but still be separate roles. Likewise for users. A role is properly viewed as a semantic construct around which access control policy is formulated. The particular collection of users and permissions brought together by a role is transitory. For example, when a role is created it may have no users or permissions assigned to it. At any instant there may be several such roles that have been created. This situation may persist for days. Nevertheless the roles should not be viewed as identical.

As noted earlier, RBAC_0 treats permissions as uninterpreted symbols because the precise nature of a permission is implementation and system dependent. We do require that permissions apply to data and resource objects and not to the components of RBAC itself. Permissions to modify the sets U, R, and P and relations PA and UA are called *administrative permissions*. These will be discussed later in the management model for RBAC. For now we assume that only a single security officer can change these components.

Sessions are under the control of individual users. As far the model is concerned, a user can create a session and choose to activate some subset of the user's roles. Roles active in a session can be changed at the user's discretion. The session terminates at the user's initiative. (Some systems will terminate a session if it is inactive for too long. Strictly speaking, this is a constraint and properly belongs in RBAC_2.)

RBAC_0 does not have a notion of one session creating another session. Sessions are created directly by the user. For simplicity we have omitted this aspect, but we recognize there are situations where an existing will need to create another session, possibly with different roles.

Some authors include duties [Jon93], in addition to permissions, as an attribute of roles. A duty is an obligation on a user's part to perform one or more tasks, which are generally essential for the smooth functioning of an organization. In our view duties are an advanced concept which do not belong in RBAC_0. We have also chosen not to incorporate duties in our advanced models. One approach is to treat them as similar to permissions. Other approaches could be based on new access control paradigms such as task-based authorization [San88, San91, TS94, TS97].

2.2 Role Hierarchies: RBAC_1

RBAC_1 introduces role hierarchies (RH). Role hierarchies are almost inevitably included whenever roles are discussed in the literature [FK92, HDT95, NO95, vSvdM94]. They are also commonly implemented in systems that provide roles.

Role hierarchies are a natural means for structuring roles to reflect an organization's

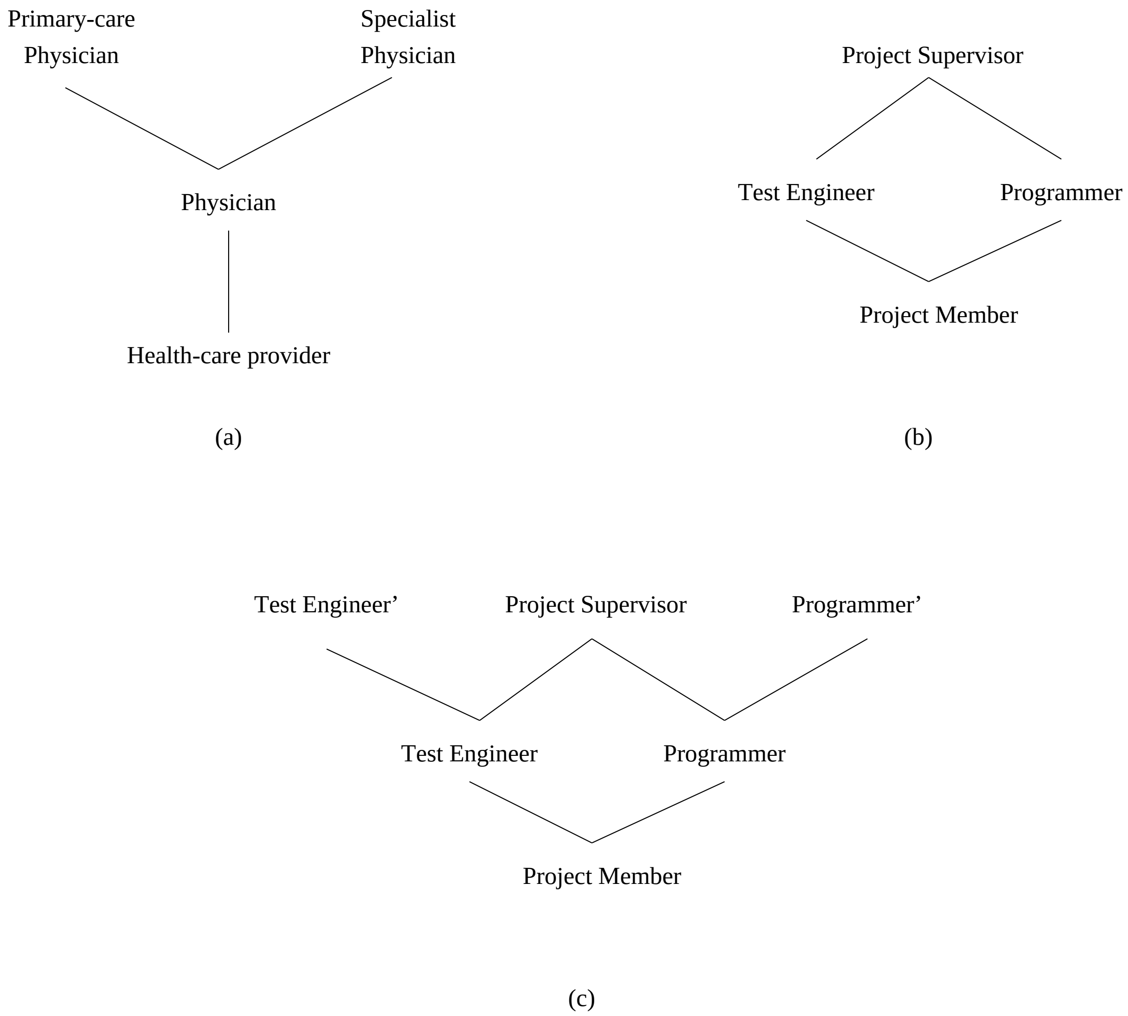


Figure 2: Examples of Role Hierarchies

lines of authority and responsibility. Examples of role hierarchies are shown in figure 2. By convention more powerful (or senior) roles are shown toward the top of these diagrams, and less powerful (or junior) roles toward the bottom. In figure 2(a) the junior-most role is health-care provider. The physician role is senior to health-care provider and thereby inherits all permissions from health-care provider. The physician role can have permissions in addition to those inherited from the health-care provider role. Inheritance of permissions is transitive so, for example, in figure 2(a), the primary-care physician role inherits permissions from the physician and health-care provider roles. Primary-care physician and specialist physician both inherit permissions from the physician role, but each one of these will have different permissions directly assigned to it. Figure 2(b) illustrates multiple inheritance of permissions, where the project supervisor role inherits from both test engineer and programmer roles.

Mathematically, these hierarchies are partial orders. A partial order is a reflexive, transitive and anti-symmetric relation. Inheritance is reflexive because a role inherits its own permissions, transitivity is a natural requirement in this context, and anti-symmetry rules out roles that inherit from one another and would therefore be redundant.

The formal definition of RBAC_1 is given below.

Definition 2 The RBAC_1 model has the following components:

- U, R, P, S, PA, UA, and $user$ are unchanged from RBAC_0,
- $RH \subseteq R \times R$ is a partial order on R called the role hierarchy or role dominance relation, also written as $\geq$ in infix notation, and
- $roles : S \rightarrow 2^R$ is modified from RBAC_0 to require $roles(s_i) \subseteq \{r \mid (\exists r' \geq r)[(user(s_i), r') \in UA]\}$ (which can change with time) and session s_i has the permissions $\cup_{r \in roles(s_i)} \{p \mid (\exists r'' \leq r)[(p, r'') \in PA]\}$. □

We also write $x > y$ to mean $x \geq y$ and $x \neq y$.

Note that a user is allowed to establish a session with any combination of roles junior to those the user is a member of. Also, the permissions in a session are those directly assigned to the active roles of the session as well as those assigned to roles junior to these.

It is sometimes useful in hierarchies to limit the scope of inheritance. Consider the hierarchy of figure 2(b) where the project supervisor role is senior to both the test engineer and programmer roles. Now suppose test engineers wish to keep some permissions private to their role and prevent their inheritance in the hierarchy to project supervisors. This situation can exist for legitimate reasons, for example, access to incomplete work in progress may not be appropriate for the senior role while RBAC can be useful for enabling such access to test engineers. This situation can be accommodated by defining a new role called test engineer′ and relating it to test engineer as shown in figure 2(c). The private permissions of test engineers are assigned to role test engineer′. Test engineers are assigned to role test engineer′ and inherit permissions from the test engineer role, which are also inherited upward in the hierarchy by the project supervisor role. Permissions of test engineer′ are, however, not inherited by the project supervisor role. We call roles such as test engineer′

as *private roles*. Figure 2(c) also shows a private role programmer′. In some systems the effect of private roles is achieved by blocking inheritance of certain permissions. In this case the hierarchy does not depict the distribution of permission accurately. It is preferable to introduce private roles and keep the meaning of the hierarchical relationship among roles intact.

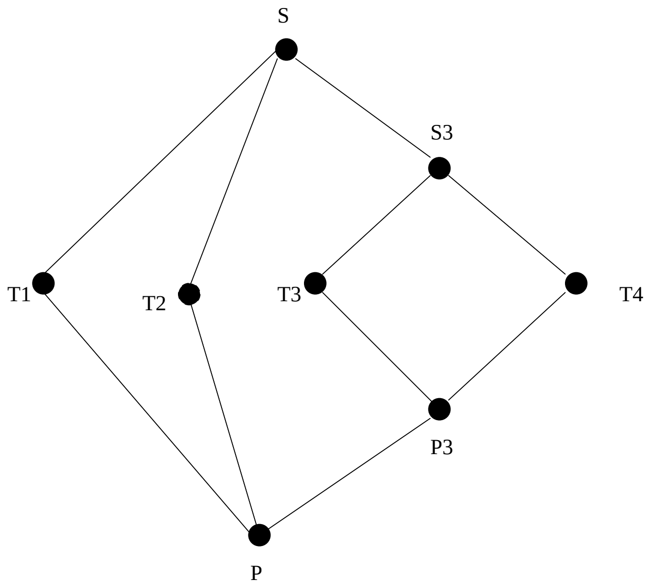
Figure 3 shows, more generally, how a private subhierarchy of roles can be constructed. The hierarchy of figure 3(a) has four task roles, $T1$, $T2$, $T3$ and $T4$, all of which inherit permissions from the common project-wide role P. Role S at the top of the hierarchy is intended for project supervisors. Tasks $T3$ and $T4$ are a subproject with $P3$ as the subproject-wide role, and $S3$ as the subproject supervisory role. Role $T1'$ in figure 3(c) is a private role for members of task $T1$. Suppose the subproject of figure 3(a) comprising roles $S3$, $T3$, $T4$, and $P3$, requires a private subhierarchy within which private permissions of the project can be shared without inheritance by S. The entire subhierarchy is replicated in the manner shown in figure 3(c). The permissions inheritable by S can be assigned to $S3$, $T3$, $T4$, and $P3$, as appropriate whereas the privates ones can be assigned to $S3'$, $T3'$, $T4'$, and $P3'$, allowing their inheritance within the subproject only. As before members of the subproject team are directly assigned to $S3'$, $T3'$, $T4'$, or $P3'$. Figure 3(c) makes it clear as to which private roles exist in the system and assists in access review to determine what the nature of the private permissions is.

2.3 Constraints: RBAC_2

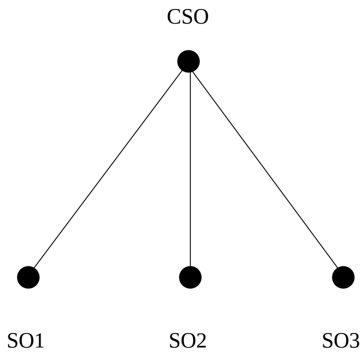
RBAC_2 introduces the concept of constraints. Constraints apply to all aspects of RBAC as indicated in figure 1(b). Although we have called our models RBAC_1 and RBAC_2, there isn't really an implied progression. Either constraints or role hierarchies can be introduced first. Hence the incomparable relation between RBAC_1 and RBAC_2 in figure 1(a).

Constraints are an important aspect of RBAC and are sometimes argued to be the principal motivation for RBAC. A common example is that of mutually disjoint roles, such as purchasing manager and accounts payable manager. In most organizations the same individual will not be permitted to be a member of both roles, because this creates a possibility for committing fraud. This is a well-known and time-honored principle called separation of duties.

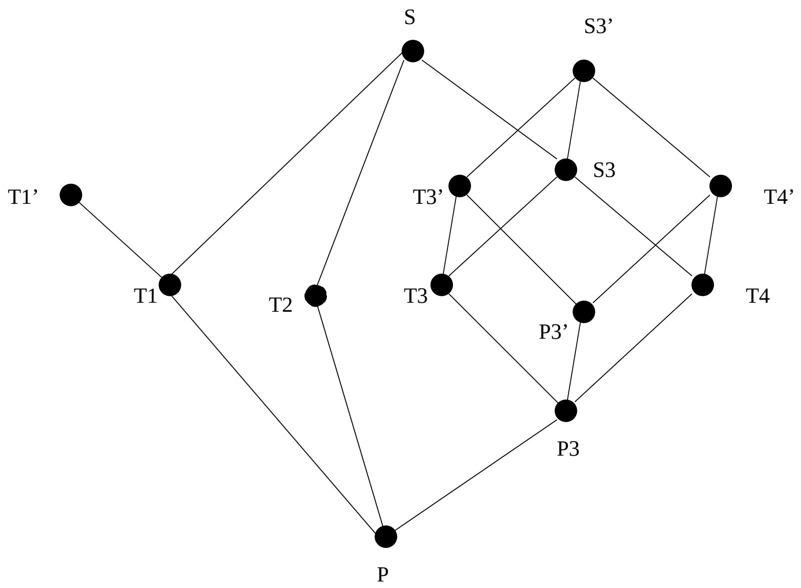
Constraints are a powerful mechanism for enforcing higher-level organizational policy. Once certain roles are declared to be mutually exclusive, there need not be so much concern about the assignment of individual users to roles. The latter activity can be delegated and decentralized without fear of compromising overall policy objectives of the organization. So long as the management of RBAC is entirely centralized in a single security officer, constraints are a useful convenience; but the same effect can largely be achieved by judicious care on the part of the security officer. However, if management of RBAC is decentralized (as will be discussed later), constraints become a mechanism by which senior security officers can restrict the ability of users who can exercise administrative privileges. This enables the chief security officer to lay out the broad scope of what is acceptable and impose this as a mandatory requirement on other security officers and users who participate in RBAC management.



(a) Role Hierarchy



(b) Administrative Role Hierarchy



(c) Private and Scoped Roles

Figure 3: Role Hierarchies for a Project

With respect to RBAC_0 constraints can apply to the UA and PA relations and the *user* and *roles* functions for various sessions. Constraints are predicates which, applied to these relations and functions, return a value of "acceptable" or "not acceptable." Constraints can also be viewed as sentences in some appropriate formal language. Intuitively, constraints are better viewed according to their kind and nature. We discuss constraints informally rather than stating them in a formal notation. Hence, the following definition.

Definition 3 RBAC_2 is unchanged from RBAC_0 except for requiring that there be a collection of constraints that determine whether or not values of various components of RBAC_0 are acceptable. Only acceptable values will be permitted. □

Implementation considerations generally call for simple constraints that can be efficiently checked and enforced. Fortunately, in RBAC simple constraints can go a long way. We now discuss some constraints that we feel are reasonable to implement. Most, if not all, constraints applied to the user assignment relation have a counterpart that applies to the permission assignment relation. We therefore discuss constraints on these two components in parallel.

The most frequently mentioned 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 separation of duties. Provision of this constraint requires little motivation. The dual constraint on permission assignment receives hardly any mention in the literature. Actually, a mutual exclusion constraint on permission assignment can provide additional assurance for separation of duties. This dual constraint requires that the same permission can be assigned to at most one role in a mutually exclusive set. Consider two mutually exclusive roles, accounts-manager and purchasing-manager. Mutual exclusion in terms of UA specifies that one individual cannot be a member of both roles. Mutual exclusion in terms of PA specifies that the same permission cannot be assigned to both roles. For example, the permission to issue checks should not be assigned to both roles. Normally such a permission would be assigned to the accounts-manager role. The mutual exclusion constraint on PA would prevent the permission from being inadvertently, or maliciously, assigned to the purchasing-manager role. More directly, exclusion constraints on PA are a useful means of limiting the distribution of powerful permissions. For example, it may not matter whether role A or role B gets signature authority for a particular account, but we may require that only one of the two roles gets this permission.

More generally membership by users in various combinations of roles can be deemed to be acceptable or not. Thus it may be acceptable for a user to be a member of a programmer role and a tester role in different projects, but unacceptable to take on both roles within the same project. Similarly for permission assignment.

Another example of a user assignment constraint is that a role can have a maximum number of members. For instance, there is only one person in the role of chairman of a department. Similarly, the number of roles to which an individual user can belong could also be limited. We call these *cardinality constraints*. Correspondingly, the number of roles to which a permission can be assigned can have cardinality constraints to control the distribution of powerful permissions. It should be noted that minimum cardinality constraints may be difficult to implement. For example if there is a minimum number of occupants of a role, what can the system do if one of them disappears? How will the system know this has happened?

The concept of *prerequisite roles* is based on competency and appropriateness, whereby a user can be assigned to role A only if the user is already a member of role B. For example, only those users who are already members of the project role can be assigned to the testing task role within that project. In this example the prerequisite role is junior to the new role being assumed. Prerequisites between incomparable roles can also occur in practice. A similar constraint on permission assignment can arise in the following way. It could be useful, for consistency, to require that permission p can be assigned to a role only if that role already possesses permission q. For instance, in many systems permission to read a file requires permission to read the directory in which the file is located. Assigning the former permission without the latter would be incomplete. More generally we can have *prerequisite conditions* whereby a user can be assigned to role A only if the user is already a member or not a member of specified roles, and similarly for permissions. This idea is used in the administrative models of section 3.

User assignment constraints are effective only if suitable external discipline is maintained in assigning user identifiers to human beings. If the same individual is assigned two or more user identifiers, separation and cardinality controls break down. There must be a one-to-one correspondence between user identifiers and human beings. A similar argument applies to permission constraints. If the same operation is sanctioned by two different permissions, the RBAC system cannot effectively enforce cardinality and separation constraints.

Constraints can also apply to sessions, and the *user* and *roles* functions associated with a session. It may be acceptable for a user to be a member of two roles but the user cannot be active in both roles at the same time. Other constraints on sessions can limit the number of sessions that a user can have active at the same time. Correspondingly, the number of sessions to which a permission is assigned can be limited.

A role hierarchy can be considered as a constraint. The constraint is that a permission assigned to a junior role must also be assigned to all senior roles. Or equivalently, the constraint is that a user assigned to a senior role must also be assigned to all junior roles. So in some sense, RBAC_1 is redundant and is subsumed by RBAC_2. However, we feel it is appropriate to recognize the existence of role hierarchies in their own right. They are reduced to constraints only by introducing redundancy of permission assignment or user assignment. It is preferable to support hierarchies directly rather than indirectly by means of redundant assignment.

2.4 Consolidated Model: RBAC_3

RBAC_3 combines RBAC_1 and RBAC_2 to provide both role hierarchies and constraints. There are several issues that arise by bringing these two concepts together.

Constraints can apply to the role hierarchy itself. The role hierarchy is required to be a partial order. This constraint is intrinsic to the model. Additional constraints can limit the number of senior (or junior) roles that a given role may have. Two or more roles can also be constrained to have no common senior (or junior) role. These kinds of constraints are useful in situations where the authority to change the role hierarchy has been decentralized, but the chief security officer desires to restrict the overall manner in which such changes can be made.

Subtle interactions arise between constraints and hierarchies. Suppose that test engineer and programmer roles are declared to be mutually exclusive in the context of figure 2(b). The project supervisor role violates this mutual exclusion. In some cases such a violation of a mutual exclusion constraint by a senior role may be acceptable, while in other cases it may not. We feel that the model should not rule out one or the other possibility. A similar situation arises with cardinality constraints. Suppose that a user can be assigned to at most one role. Does an assignment to the test engineer role in figure 2(b) violate this constraint? In other words, do cardinality constraints apply only to direct membership, or do they also carry on to inherited membership?

The hierarchy of figure 2(c) illustrates how constraints are useful in the presence of private roles. In this case the test engineer′, programmer′, and project supervisor roles can be declared to be mutually exclusive. Because these have no common senior for these roles, there is no conflict. In general private roles will not have common seniors with any other roles because they are maximal elements in the hierarchy. So mutual exclusion of private roles can be specified without conflict. The shared counterpart of the private roles can be declared to have a maximum cardinality constraint of zero members. In this way test engineers must be assigned to the test engineer′ role. The test engineer role serves as a means for sharing permissions with the project supervisor role.

2.5 Discussion

We have presented the RBAC96 family of models that systematically spans the spectrum from simple to complex. RBAC96 provides a common frame of reference. $RBAC_0$ is simple and free of built in constraints so as to provide a foundation for the more advanced models. $RBAC_2$ allows us to accommodate existing systems and models that have built in constraints such as only one role can be used at any time. $RBAC_1$ models the commonly occurring case of hierarchical roles, and $RBAC_3$ consolidates all of these. We conclude this section by discussing some salient aspects of RBAC96.

Users and Sessions

The distinction between a user and a session is a fundamental aspect of RBAC and consequently arises in $RBAC_0$. A user is a human being, or other intelligent agent, capable of autonomous activity in the system. To support the principle of least privilege a user should be allowed to login to a system with only those roles appropriate for a given occasion.

Many systems will turn on all permissions of a user irrespective of what the user wishes to accomplish in a particular session. Thus, a user who has powerful permissions (or roles) that are used only rarely when needed finds that these permissions are turned on all the time. It is possible to set up separate accounts, one in which the usual permissions are turned on and another in which the powerful permissions are turned on. Assigning multiple accounts to the same user introduces problems with respect to auditing, accountability and constraints such as separation of duties. It is not a desirable general-purpose solution but can be employed in the short term to simulate RBAC on existing platforms.

In $RBAC_0$ the distinction between users and sessions is useful only if users exercise discipline regarding the roles they normally invoke. With constraints it may not be possible for a user to activate all their roles simultaneously. Consider a constraint that stipulates two roles which can be assigned to the same user but cannot be simultaneously activated in a session. For instance, a user may be qualified to be a pilot and a navigator but at any time can activate at most one of these roles. In presence of such constraints a user cannot establish a single session with all the user's roles activated.

An important property of a session is that the user associated with a session cannot change. In many applications there are long-lived sessions where one user hands over to another without a logout and login. This preserves the integrity of the computing activity being performed in the session. We feel this problem is an artifact of existing system architectures. Continuity of activity across multiple security sessions should be possible in properly engineered systems. Also our models are conceptual models seeking to capture what needs to be achieved. In implementations on specific platforms we will need to simulate the requirements with the mechanisms available.

The RBAC96 models do not address the issues of idle session termination and lockout. In practice this is an important issue. In our conceptual framework termination and lockout is most easily modeled as a constraint and belongs in $RBAC_1$. As a practical matter it would be hard to effectively do $RBAC_0$ without bringing in at least a small number of constraints of this nature.

Changing the roles activated in a session is a security sensitive act and should be acknowledged to the security system via a so-called trusted path which guarantees that the user is making the request rather than some program acting on the user's behalf. Such changes can be regulated by constraints in $RBAC_1$. For instance, certain roles may not be dynamically added but can only be acquired when a session is created. $RBAC_0$ allows dynamic changing of roles in a session because of two reasons. From a conceptual viewpoint constraints belong in $RBAC_1$ and higher, and should not be present in $RBAC_0$. We could still define $RBAC_0$ to disallow all changes in a session's roles. We felt this is impractical and too restrictive for a base model.

RBAC96 does not address how one session might create another session. This issue is discussed in [Tho91]. RBAC96 takes the view that a user creates a session and in absence of other constraints can change the roles of this session as the user pleases. However, it is possible for one session to create another session with different roles under program control rather than under direct user control. Thomsen uses a domain transition table to authorize this. This issue needs further work in RBAC96.

Permissions

It is difficult to identify the nature of permissions precisely in an abstract general purpose model such as RBAC96. Permissions tend to be implementation dependent. In lattice-based access control models [San93] it is possible to abstract the essential operations into read and write. This is because these models are focussed on one-directional information flow in a lattice of security labels.

RBAC models are policy neutral. Hence the nature of permissions has to be open ended. In applying RBAC to a particular system the interpretation of permissions is among the most important steps to be performed.

We deliberately decided to exclude so-called negative permissions from RBAC96. Negative permissions deny rather than confer access. They are used in some discretionary access control models to disallow a user from obtaining a permission from some alternate source. The use of constraints in RBAC is a much more useful mechanism to achieve the same result. The literature on negative permissions is fraught with problems concerning their interaction and relative strength with respect to positive permissions. In the presence of role hierarchies this could become very complicated and arcane. We would be very reluctant to add negative permissions into a complex model such as RBAC96.

The scope of RBAC is also consciously limited to classical permissions. Sequencing or temporal dependencies between permissions are important in emerging applications such as workflow [San88, San91, TS94, TS97]. We decided to limit the scope of RBAC to exclude these for two reasons. Firstly, these are not yet well understood and much further basic research is required for this purpose. Secondly, RBAC must have a well-delineated scope otherwise it will be an amorphous concept which can be taken to include all kinds of security and authorization issues.

Model Conformance

What does it mean for a system to conform to RBAC96? RBAC96 is best viewed as a family of reference models which play a dual role. On one hand RBAC96 provides a framework for analyzing the capabilities of existing systems to assess how well and how extensively they can support RBAC. RBAC96 also provides guidance to vendors and developers regarding access controls to be implemented in future systems.

It is not necessary for a system to completely conform to RBAC_0 before it includes features of RBAC_1 or RBAC_2. Many existing systems do not distinguish between users and sessions. We would say these systems have aspect of RBAC_0, RBAC_1 and RBAC_2, but are also missing other aspects of RBAC_0. These systems essentially have built-in constraints that all roles must be turned on in a session and none can be dropped. Other systems have hard-wired constraints, such as a session can only have one role at a time. Such systems cannot accommodate RBAC_0, because they do too much without any choice in the matter. But we can still place them within the RBAC96 family.

3 THE ARBAC97 ADMINISTRATIVE MODELS

So far we have assumed that all components of RBAC are under direct control of a single security officer. In large systems the number of roles can be in the hundreds or thousands. Managing these roles and their interrelationships is a formidable 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 permissions, it is natural to ask how RBAC can be used to manage RBAC itself. We believe that the use of RBAC for managing

RBAC will be an important factor in the success of RBAC. Decentralizing the details of RBAC administration without loosing central control over broad policy is a challenging goal for system designers and architects.

We mention some approaches to access control management that have been discussed in the literature. ISO has developed a number of security management related standards and documents. These can be approached via the top-level System Management Overview document [ISO]. The ISO model is object-oriented and includes a hierarchy based on containment (a directory contains files and a file contains records). Roles could be integrated into the ISO approach.

There is a long tradition of models for propagation of access rights, where the right to propagate rights is controlled by special control rights. Among the most recent and most developed of these is Sandhu's typed access matrix model [San92]. While it is often difficult to analyze the consequences of even fairly simple rules for propagation of rights, these models indicate that simple primitives can be composed to yield very flexible and expressive systems.

One example of work on managing RBAC is by Moffet and Sloman [MS91] who define a fairly elaborate model based on role domains, owners, managers, and security administrators. In their work authority is not controlled or delegated from a single central point, but rather is negotiated between independent managers who have only a limited trust in each other.

Our management model for RBAC is illustrated in figure 4. The top half of this figure is essentially the same as figure 1(b). The constraints in figure 4 apply to all components. The bottom half of figure 4 is a mirror image of the top half for administrative roles and administrative permissions. It is intended that administrative roles AR and administrative permissions AP be respectively disjoint from the regular roles R and permissions P. The model shows that permissions can only be assigned to roles and administrative permissions can only be assigned to administrative roles. This is a built-in constraint, stated formally as follows.

Definition 4 Administrative permissions AP authorize changes to the various components that comprise RBAC_0, RBAC_1, RBAC_2 or RBAC_3 whereas regular permissions P do not. Administrative permissions are disjoint from regular permissions, i.e., $AP \cap P = \emptyset$. Administrative permissions and regular permissions can only be assigned to administrative roles AR and regular roles R respectively. Administrative roles are disjoint from regular roles, i.e., $AR \cap R = \emptyset$. □

The top half of figure 4 can range in sophistication across RBAC_0, RBAC_1, RBAC_2, and RBAC_3. The bottom half can similarly range in sophistication across ARBAC_0, ARBAC_1, ARBAC_2, and ARBAC_3, where the A denotes administrative. In general we would expect the administrative model to be simpler than the RBAC model itself. Thus ARBAC_0 can be used to manage RBAC_3, but there seems to be no point in using ARBAC_3 to manage RBAC_0.

It is also important to recognize that constraints can cut across both top and bottom halves of figure 4. We have already asserted a built-in constraint that permissions can only be assigned to roles and administrative permissions can only be assigned to administrative

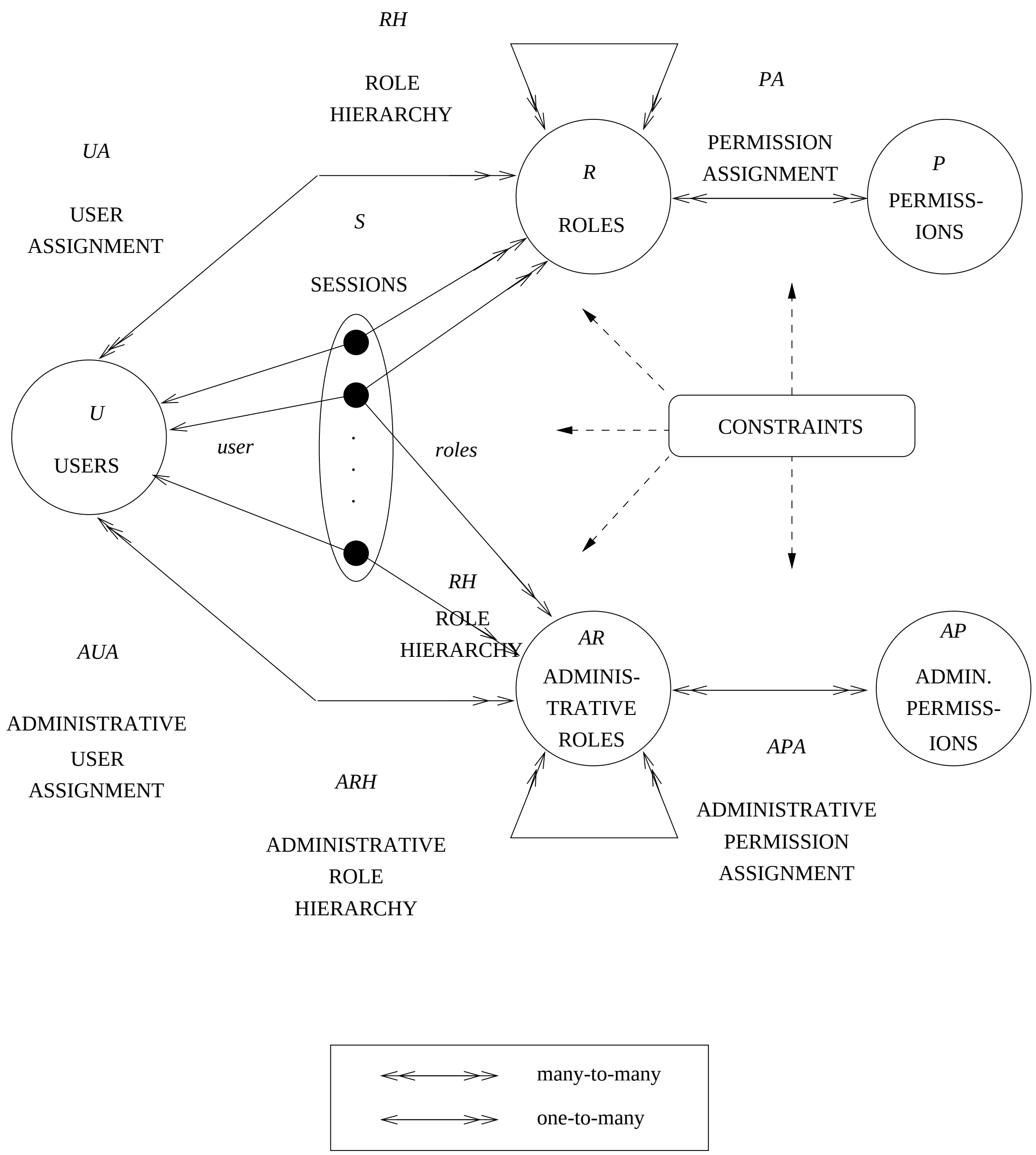


Figure 4: RBAC Administrative Model

roles. If administrative roles are mutually exclusive with respect to regular roles, we will have a situation where security administrators can manage RBAC but not use any of the privileges themselves.

How about management of the administrative hierarchy? In principle one could construct a second level administrative hierarchy to manage the first level one and so on. We feel that even a second level of administrative hierarchy is unnecessary. Hence the administration of the administrative hierarchy is left to a single chief security officer. This is reasonable for a single organization or a single administrative unit within an organization. The issue of how these units interact is not directly addressed in our model. More generally administrative roles could themselves manage administrative roles.

One of the main issues in a management model is how to scope the administrative authority vested in administrative roles. To illustrate this consider the hierarchies shown in figure 3(a). The administrative hierarchy of figure 3(b) shows a single chief security officer role (CSO), which is senior to the three security officer roles SO1, SO2, and SO3. The scoping issue concerns which roles of figure 3(a) can be managed by which roles of figure 3(b). Let us say the CSO role can manage all roles of figure 3(a). Suppose SO1 manages task T1. In general we do not want SO1 to automatically inherit the ability to manage the junior role P also. So the scope of SO1 can be limited exclusively to T1. Similarly, the scope of SO2 can be limited to T2. Assume SO3 can manage the entire subproject consisting of S3, T3, T4, and P3. The scope of SO3 is then bounded by S3 at the top and P3 at the bottom.

In general, each administrative role will be mapped to some subset of the role hierarchy it is responsible for managing. There are other aspects of management that need to be scoped. For example, SO1 may only be able to add users to the T1 role but their removal requires the CSO to act. More generally, we need to scope not only the roles an administrative role manages, but also the permissions and users that role manages. It is also important to control changes in the role hierarchy itself. For example, because SO3 manages the subhierarchy between S3 and P3, SO3 could be authorized to add additional tasks to that subproject.

As we have seen there are many components to RBAC. RBAC administration is therefore multi-faceted. In particular we can separate the issues of assigning users to roles, assigning permissions to roles, and assigning roles to roles to define a role hierarchy. These activities are all required to bring users and permissions together. However, in many cases, they are best done by different administrators or administrative roles. Assigning permissions to roles is typically the province of application administrators. Thus a banking application can be implemented so credit and debit operations are assigned to a teller role, whereas approval of a loan is assigned to a managerial role. Assignment of actual individuals to the teller and managerial roles is a personnel management function. Assigning roles to roles has aspects of user-role assignment and role-permission assignment. Role-role relationships establish broad policy. Control of these relationships would typically be relatively centralized in the hands of a few security administrators.

In this section we describe a model for role-based administration of RBAC. Our model is called ARBAC97 (administrative RBAC '97). It has three components as follows.

1. The **user-role assignment** component of ARBAC97 is called URA97 (user-role assignment '97).

2. The **permission-role assignment** component of ARBAC97 is a dual[2] of URA97 and is called PRA97 (permission-role assignment '97).

3. The **role-role assignment** component of ARBAC97 itself has several components which are determined by the kind of roles that are involved. We defer discussion of the role-role assignment model till section 3.3.

3.1 URA97 for User-Role Assignment

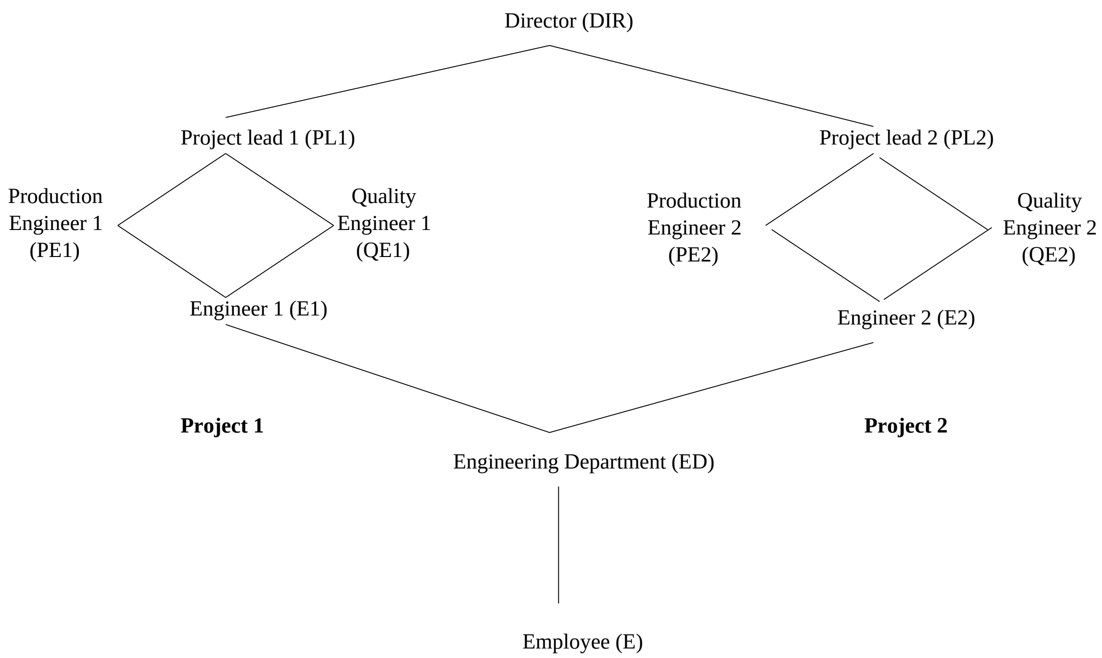
The URA97 model was originally defined by Sandhu and Bhamidipati [SB97], who also developed an Oracle implementation of this model. We use the hierarchies of figures 5(a) and 5(b) in our running example through this section. Figure 5(a) shows the regular roles in an engineering department. There is a junior-most role E to which all employees belong. The engineering department has a junior-most role ED and senior-most role DIR. In between there are roles for two projects within the department, project 1 on the left and project 2 on the right. Each project has a senior-most project lead role (PL1 and PL2), a junior-most engineer role (E1 and E2), and in between two incomparable roles, production engineer (PE1 and PE2) and quality engineer (QE1 and QE2). Figure 5(b) shows the administrative role hierarchy with the senior security officer (SSO) role at the top, and two project security officer roles (PSO1 and PSO2) and a department security officer (DSO) role.

URA97 is concerned with administration of the user-assignment relation UA which relates users to roles. Authorization to modify this relation is controlled by administrative roles. Thus members of the administrative roles in figure 5(b) are authorized to modify membership in the roles of figure 5(a). Assignment of user to administrative roles is outside the scope of URA97 and is assumed to be done by the chief security officer.

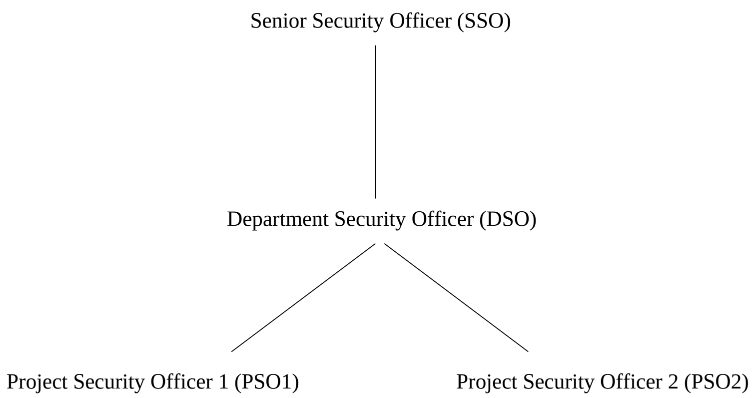
There are two aspects to decentralization of user-role assignment. We need to specify the roles whose membership can be modified by an administrative role. We also need to specify a population of users eligible for membership. For example, URA97 will let us specify that the administrative role PSO1 can assign users to the roles PE1, QE1 and E1, but these users must previously be members of the role ED. The idea is that PSO1 has freedom to assign users to roles in project 1 (excepting the senior-most role PL1) but these users must already be members of the engineering department. This is an example of a prerequisite role. More generally URA97 allows for a prerequisite condition as follows.

Definition 5 A prerequisite condition is a boolean expression using the usual $\wedge$ and $\vee$ operators on terms of the form x and $\overline{x}$ where x is a regular role (i.e., $x \in R$). For a given set of roles R let CR denotes all possible prerequisite conditions that can be formed using the roles in R. A prerequisite condition is evaluated for a user u by interpreting x to be true if $(\exists x' \geq x)(u, x') \in UA$ and $\overline{x}$ to be true if $(\forall x' \geq x)(u, x') \notin UA$. $\Box$

[2]In our work we have often observed a duality between user-role and permission-role relationships. For example, every constraint on user-role relationships has a dual counterpart with respect to permission-role relationships, and vice versa. We see this duality exhibited here too.



(a) Roles



(b) Administrative Roles

Figure 5: Example Role and Administrative Role Hierarchies

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

(a) can-assign

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

(b) can-revoke

Figure 6: Example of can-assign and can-revoke

Definition 6 User-role assignment and revocation are respectively authorized in URA97 by the following relations, *can-assign* $\subseteq AR \times CR \times 2^R$ and *can-revoke* $\subseteq AR \times 2^R$. □

The meaning of *can-assign*(x, y, Z) is that a member of the administrative role x (or a member of an administrative role that is senior to x) can assign a 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.[3] The meaning of *can-revoke*(x, Y) is that a member of the administrative role x (or a member of an administrative role that is senior to x) can revoke membership of a user from any regular role $y \in Y$.

Figure 6 illustrates these relations. Role sets are specified in URA97 by the following *range notation*.

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

By figure 6(a) PSO1 can assign users in ED to the roles E1, PE1 and QE1, and similarly for PSO2 with respect to E2, PE2 and QE2. DSO can assign a user in ED to PL1 provided that user is not already in PL2, and similarly for PL2 with respect to PL1.

A notable aspect of revocation in URA97 is that revocation is independent of assignment. If Alice, by means of some administrative role, can revoke Bob's membership in a regular role the revocation takes effect independent of how Bob came to be a member of that regular role. This is consistent with RBAC philosophy where granting and revoking of membership is done for organizational reasons and not merely at the discretion of individual administrators.

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

Weak and Strong Revocation

The revocation operation in URA97 is said to be weak because it applies only to the role that is directly revoked. Suppose Bob is a member of PE1 and E1. If Alice revokes Bob's membership from E1, he continues to be a member of the senior role PE1 and therefore can use the permissions of E1. Various forms of strong revocation can be considered as embellishments to URA97. Strong revocation cascades upwards in the role hierarchy. If Alice has administrative role PSO1 and she strongly revokes Bob's membership from E1 as per figure 6, his membership in PE1 is also revoked. However, if Charles is a member of E1 and PL1, and Alice strongly revokes Charles' membership in E1 the cascaded revoke is outside of Alice's range and is disallowed. The question remains whether or not Charles' membership in E1 and PE1 should be revoked even though the cascaded revoke from PL1 failed? It seems appropriate to allow both options depending upon Alice's choice. In general URA97 treats strong revocation as a series of weak revocations each of which must be individually authorized by can-revoke. In this way we keep the basic URA97 model simple while allowing for more complex revocation operations to be defined in terms of weak revocation. At the same time we feel it is important to support strong revocation.

3.2 PRA97 for Permission-Role Assignment

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

Definition 7 Permission-role assignment and revocation are respectively authorized by the following relations, $\mathit{can\text{-}assignp} \subseteq AR \times CR \times 2^R$ and $\mathit{can\text{-}revokep} \subseteq AR \times 2^R$. □

The meaning of $\mathit{can\text{-}assignp}(x, y, Z)$ is that a member of the administrative role x (or a member of an administrative role that is senior to x) can assign a permission whose current membership, or non-membership, in regular roles satisfies the prerequisite condition y to regular roles in range Z.[4] The meaning of $\mathit{can\text{-}revokep}(x, Y)$ is that a member of the administrative role x (or a member of an administrative role that is senior to x) can revoke membership of a permission from any regular role $y \in Y$.

Figure 7 shows examples of these relations. The DSO is authorized to take any permission assigned to the DIR role and make it available to PL1 or PL2. Thus a permission can be delegated downward in the hierarchy. PSO1 can assign permissions from PL1 either PE1 or QE1, but not to both. The remaining rows in figure 7(a) are similarly interpreted.

Figure 7(b) authorizes DSO to revoke permissions from any role between ED and DIR. PSO1 can revoke permissions from PE1 and QE2, and similarly for PSO2.

Revocation in PRA97 is weak so permissions may still be inherited after revocation. Strong revocation can be defined in terms of weak revocation as in URA97. Strong revo-

[4] Permission-role assignment may be subject to additional constraints. In other words $\mathit{can\text{-}assignp}(x, y, Z)$ is a necessary but not sufficient condition.

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

(a) can-assignp

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

(b) can-revokep

Figure 7: Example of can-assignp and can-revokep

cation of a permissions cascades down the role hierarchy, in contrast to cascading up of revocation of user membership.

3.3 RRA97 for Role-Role Assignment

Finally, we consider the issue of role-role assignment. Our treatment is informal and preliminary at this point because the model is still evolving. Our focus is on the general direction and intuition.

For role-role assignment we distinguish three kinds of roles, roughly speaking as follows.

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

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

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

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

The main reason to distinguish these three kinds of roles is that different administrative models apply to establishing relationships between them. The distinction was motivated in the first place by abilities. An ability is a collection of permissions that should be assigned as a single unit to a role. For example the ability to open an account in a banking application will encompass many different individual permissions. It does not make sense to assign only some of these permissions to a role because the entire set is needed to do the task properly. The idea is that application developers package permissions into collections called abilities which must be assigned together as a unit to a role. The function of an ability is to collect permissions together so that administrators can treat these as a single unit. Assigning abilities to roles is therefore very much like assigning permissions to roles. For convenience it is useful to organize abilities into a hierarchy (i.e., partial order). Hence the PRA97 model can be adapted to produce the very similar ARA97 model for ability-role assignment.

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

This leads to the following models.

Definition 8 Ability-role assignment and revocation are respectively authorized in ARA97 by $\textit{can-assigna} \subseteq AR \times CR \times 2^{UPR}$ and $\textit{can-revokea} \subseteq AR \times 2^{UPR}$. □

Definition 9 Group-role assignment and revocation are respectively authorized in GRA97 by $\textit{can-assigng} \subseteq AR \times CR \times 2^{UPR}$ and $\textit{can-revokeg} \subseteq AR \times 2^{UPR}$. □

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

Assigning an ability to an UP-role is mathematically equivalent to making the UP-role an immediate senior of the ability in the role-role hierarchy. Abilities can only have UP-roles or abilities as immediate seniors and can only have abilities as immediate juniors. In a dual manner, assigning a group to an UP-role is mathematically equivalent to making the UP-role an immediate junior of the group in the role-role hierarchy. Groups can only have UP-roles or groups as immediate juniors and can only have groups as immediate seniors. With these constraints the ARA97 and GRA97 models are essentially identical to the PRA97 and URA97 models respectively.

This leaves us with the problem of managing relationships between UP-roles.[5] Consider figure 5(a) again. We would like the DSO to configure and change the hierarchy between DIR and ED. Similarly, we would like PSO1 to manage the hierarchy between PL1 and E1, and likewise for PSO2 with respect to PL2 and E2. The idea is that each department and each project has autonomy in constructing its internal role structure.

[5]Strictly speaking we also have to deal with administration of group-group and ability-ability relationships. These can be handled in the same way as relationships between UP-roles to analogously give us the G-RRA97 and A-RRA97 models.

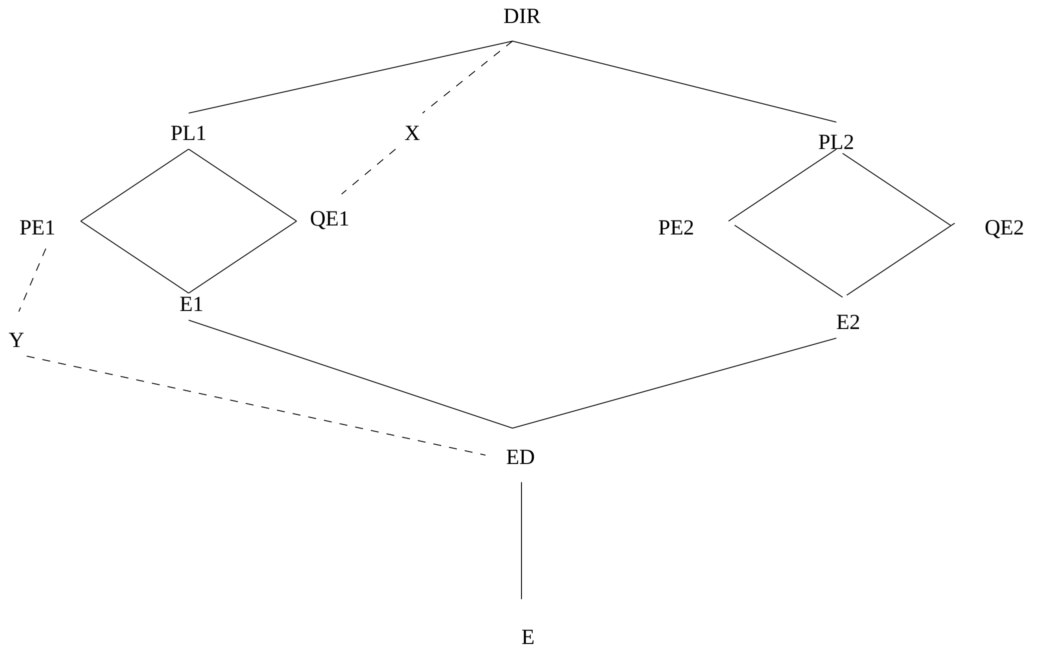


Figure 8: Out of Range Impact

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

Figure 9: Example of can-modify

Definition 10 Role creation, role deletion, role-role edge insertion, role-role edge deletion are all authorized in UP-RRA97 by *can-modify* : $AR \rightarrow 2^{UPR}$. □

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, except for the endpoints of Y, and can modify relationships between roles in the range Y. This authority is, however, tempered by constraints that we discuss below.

Figure 9 illustrates an example of can-modify relative to the hierarchies of figure 5. By convention the UP-role ranges are shown as open intervals since the endpoints are not included. DSO can create, delete and alter relationships between all roles in the engineering department (except the endpoints ED and DIR). When a DSO creates a new role it will be senior to ED and junior to DIR, and will remain so (unless some more senior administrator changes this relationship). PSO1 has similar authority with respect to roles in project 1.

Restrictions on can-modify

The authority conferred by can-modify is constrained by global consistency requirements. It is not possible to change pieces of the role hierarchy in arbitrary ways without impacting larger relationships. Here we identify two conflicts that arise and explain how RRA97 deals with them.

Suppose DSO is given the authority to create and delete edges and roles in the hierarchy between DIR and ED. If PL1 gets deleted figures 6 and 7 will be left with dangling references to a non-existent role. To avoid this situation we require that roles that are referenced in any can-assign or can-revoke relation cannot be deleted. In this way the DSO's power to delete roles is restricted to maintain global consistency of the authorizations.

The second problem arises if the DSO introduces roles X and Y as shown in figure 8. Now suppose PSO1 has authority to create and delete edges and roles in the hierarchy between PL1 and E1. If PSO1 makes PE1 junior to QE1 the effect is to indirectly make Y junior to X. Now PSO1 was given authority in the range (PL1,E1) but has effectively introduced a relationship between X and Y. There are several approaches to resolving this issue. We can prevent the DSO from introducing X and Y as shown, because this violates the range integrity of (PL1,E1) with respect to PSO1. We can allow figure 8 to happen and prevent PSO1 from later making PE1 junior to QE1. Or we can tolerate the possibility of PSO1 affecting UP-role to UP-role relationships that are outside the authorized range of (E1, PL1). RRA97 allows all three possibilities.

There may be other issues that will arise as we evolve this model. Our general principle for decentralized administration of role-role relationships is a sound one. We wish to give administrative roles autonomy within a range but only so far as the global consequences of the resulting actions are acceptable. To do so we need to disallow some operations authorized by the range, thereby tempering the administrative role's authority.

3.4 Discussion

In this section we have described the motivation, intuition and outline of a new model for RBAC administration called ARBAC97 (administrative RBAC '97). ARBAC97 has three components: URA97 (user-role assignment '97), PRA97 (permission-role assignment '97) and RRA97 (role-role assignment '97). URA97 was recently defined by Sandhu and Bhamidipati [SB97]. ARBAC97 incorporates URA97, builds upon it to define PRA97 and some components of RRA97, and introduces additional concepts in developing RRA97.

RRA97 itself consists of three components. ARA97 and GRA97 deal with ability-role assignment and group-role assignment respectively, and are very similar to PRA97 and URA97 respectively. The component dealing with role-role assignment is still evolving but we have identified the basic intuition and important issues that need to be dealt with.

An interesting property of ARBAC97 is that it allows multiple user membership in a chain of senior to junior roles. Thus Alice can be a member of PE1, ED and E simultaneously in figure 5(a), that is, (Alice,PE1)$\in$UA, (Alice,ED)$\in$UA and (Alice,E)$\in$UA. There are models in the literature that prohibit this, so we could only have (Alice,PE1)$\in$UA. The net effect regarding the permissions of Alice is identical in both cases. However, there is substantial difference when we look at the administrative model. In ARBAC97 Alice's membership in PE1, ED and E could be achieved by different administrators and could be revoked independently on one another. In fact prior membership in a prerequisite role could lead to later membership in more senior roles. We feel this is a more flexible model. Insisting that Alice can only be in a single role in a senior to junior chain of roles has dramatic impact on revocation and decentralized user-role assignment. Similar comments apply to permission-role assignment.

ARBAC97 does not address all issues of RBAC administration. For example, it does not talk about creation and deletion of users and permissions. It also does not address the management of constraints. Another issue omitted in ARBAC97 is delegation of roles, whereby one user can authorize another to represent the former in the capacity of one or more roles. Delegation of roles to automated agents acting on behalf of a user is also outside the scope of ARBAC97. Clearly there are many interesting issues in management of RBAC that remain to be investigated.

4 ROLES AND LATTICES

An important characteristic of RBAC is that it is policy neutral. RBAC provides a means for articulating policy rather than embodying a particular security policy. The policy enforced in a particular system is the net result of the precise configuration and interactions of various RBAC components as directed by the system owner. Moreover, the access control policy can evolve incrementally over the system life cycle. In large systems it is almost certain to do so. The ability to modify policy incrementally to meet the changing needs of an organization is an important benefit of RBAC.

Classic lattice-based access control (LBAC) models [San93] on the other hand are specifically constructed to incorporate the policy of one-directional information flow in a lattice. There is nonetheless strong similarity between the concept of a security label and a role. In particular, the same user cleared to say Secret can on different occasions login to a system at Secret and Unclassified levels. In a sense the user determines what role (Secret or Unclassified) should be activated in a particular session.

This leads us naturally to ask whether or not LBAC can be simulated using RBAC. If RBAC is policy neutral and has adequate generality it should indeed be able to do so. Particularly, because the notion of a role and the level of a login session are so similar. This question is theoretically significant because a positive answer would establish that LBAC is just one instance of RBAC thereby relating two distinct access control models that have been developed with different motivations. A positive answer is also practically significant, because it implies that the same system can be configured to enforce RBAC in general and LBAC in particular. This addresses the long held desire of multi-level security practitioners that technology which meets needs of the larger commercial marketplace be applicable to LBAC. The classical approach to fulfilling this desire has been to argue that LBAC has applications in the commercial sector. So far this argument has not been terribly productive. RBAC, on the other hand, is specifically motivated by needs of the commercial sector. Its customization to LBAC might be a more productive approach to dual-use technology.

In this section we show how several variations of LBAC are easily accommodated in RBAC by configuring a few RBAC components.[6] Our constructions show that the concepts of role hierarchies and constraints are critical to achieving this result. Changes in the role hierarchy and constraints lead to different variations of LBAC.

4.1 Lattice-Based Access Controls

Lattice based access control (LBAC) enforces one directional information flow in a lattice of security labels. LBAC is also known as mandatory access control (MAC) or multilevel security.[7] LBAC can be applied for confidentiality, integrity, confidentiality and integrity together, or for aggregation policies such as Chinese Walls [San93]. The mandatory access control policy is expressed in terms of security labels attached to subjects and objects. A label on an object is called a *security classification*, while a label on a user is called a *security clearance*. It is important to understand that a Secret user may run the same program, such as a text editor, as a Secret subject or as an Unclassified subject. Even though both subjects run the same program on behalf of the same user, they obtain different privileges due to their security labels. It is usually assumed that the security labels on subjects and objects, once assigned do not change.

An example of a security lattice is shown in figure 10. Information is only permitted to flow upward in the lattice. In this example, H and L respectively denote high and low, and M1 and M2 are two incomparable labels intermediate to H and L. This is a typical confidentiality lattice where information can flow from low to high but not vice versa. The labels in the lattice are partially ordered by the dominance relation written $\geq$, for example, H$\geq$L in our example. Lattices also have a least upper bound operator. Our constructions apply to partially ordered security labels in general so this operator is not relevant.

The specific mandatory access rules usually specified for a lattice are as follows, where λ signifies the security label of the indicated subject or object.

- **(Simple Security)** Subject s can read object o only if $\lambda(s) \geq \lambda(o)$
- **(Liberal $\star$-property)** Subject s can write object o only if $\lambda(s) \leq \lambda(o)$

The $\star$-property is pronounced as the star-property. For integrity reasons sometimes a stricter form of the $\star$-property is stipulated. The liberal $\star$-property allows a low subject to write a high object. This means that high data may be maliciously destroyed or damaged by low subjects. To avoid this possibility we can employ the strict $\star$-property given below.

- **(Strict $\star$-property)** Subject s can write object o only if $\lambda(s) = \lambda(o)$

[6] It should be noted that RBAC will only prevent overt flows of information. This is true of any access control model, including LBAC. Information flow contrary to the one-directional requirement in a lattice by means of so-called covert channels is outside the purview of access control per se. Neither LBAC nor RBAC addresses the covert channel issue directly. Techniques used to deal with covert channels in LBAC can be used for the same purpose in RBAC.

[7] LBAC is typically applied in addition to classical discretionary access controls (DAC) [SS94] but for our purpose we will focus only on the MAC component.

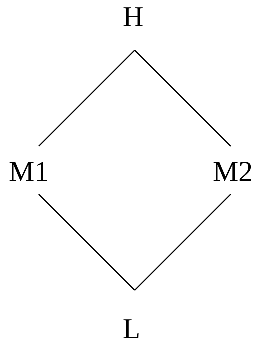


Figure 10: A Partially Ordered Lattice

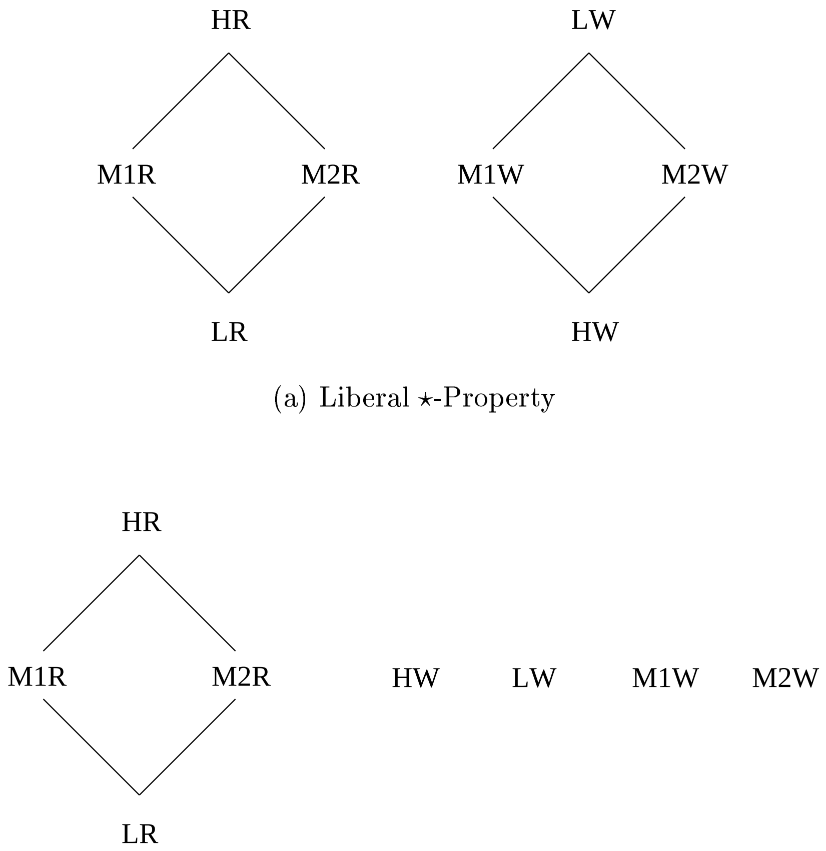
The liberal $\star$-property is also referred to as write-up and the strict $\star$-property as non-write-up or write-equal. There are also variations of LBAC where the one-directional information flow is partly relaxed to achieve selective downgrading of information or for integrity applications [Bel87, Lee88, Sch88].

We now show how these two variations of LBAC can be simulated in RBAC. It turns out that we can achieve this by suitably changing the role hierarchy and defining appropriate constraints. This confirms that role hierarchies and constraints are central to defining policy in RBAC.

4.2 Basic Lattices

Consider the example lattice of figure 10 with the liberal $\star$-property. Subjects with labels higher up in the lattice have more power with respect to read operations but have less power with respect to write operations. Thus this lattice has a dual character. In role hierarchies subjects (sessions) with roles higher in the hierarchy always have more power than those with roles lower in the hierarchy. To accommodate the dual character of a lattice for LBAC we will use two dual hierarchies in RBAC, one for read and one for write. These two role hierarchies for the lattice of figure 10 are shown in figure 11(a). Each lattice label x is modeled as two roles xR and xW for read and write at label x respectively. The relationship among the four read roles and the four write roles is respectively shown on the left and right hand sides of Figure 11(a). The duality between the left and right lattices is obvious from the diagrams.

To complete the construction we need to enforce appropriate constraints to reflect the labels on subjects in LBAC. Each user in LBAC has a unique security clearance. This is enforced by requiring that each user in RBAC is assigned to exactly one role xR determined by the user's clearance x. Each user is also assigned to *all* the maximal write roles. In this case there is one maximal write role LW. An LBAC user can login at any label dominated by the user's clearance. This requirement is captured in RBAC by requiring that each session has exactly two matching roles yR and yW. The condition that x $\geq$ y, that is the user's



(a) Liberal ⋆-Property

(b) Strict ⋆-Property

Figure 11: Role Hierarchies for the Lattice of Figure 10

clearance dominates the label of any login session established by the user, is not explicitly required because it is directly imposed by the RBAC model anyway.

LBAC is enforced in terms of read and write operations. In RBAC this means our permissions are read and writes on individual objects written as (o,r) and (o,w) respectively. An LBAC object has a single sensitivity label associated with it. This is expressed in RBAC by requiring that each pair of permissions (o,r) and (o,w) be assigned to exactly one matching pair of xR and xW roles respectively. By assigning permissions (o,r) and (o,w) to roles xR and xW respectively, we are implicitly setting the sensitivity label of object o to x.

The above construction is formalized below.

Example 1 *(Liberal ⋆-Property)*

- R = {HR, M1R, M2R, LR, HW, M1W, M2W, LW}
- RH as shown in figure 11(a)
- P = {(o,r), (o,w) | o is an object in the system}
- Constraint on UA: Each user is assigned to exactly one role xR and to all maximal write roles (in this case being the single role LW)
- Constraint on sessions: Each session has exactly two roles yR and yW
- Constraints on PA:
  - (o,r) is assigned to xR iff (o,w) is assigned to xW
  - (o,r) is assigned to exactly one role xR □

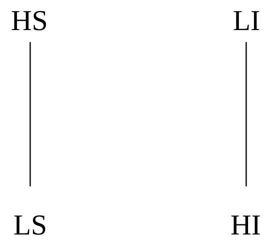
The set of permissions P remains the same in all our examples so we will omit its explicit definition in subsequent examples.

Variations in LBAC can be accommodated by modifying this basic construction in different ways. In particular, the strict ⋆-property retains the hierarchy on read roles but treats write roles as incomparable to each other as shown in figure 11(b).

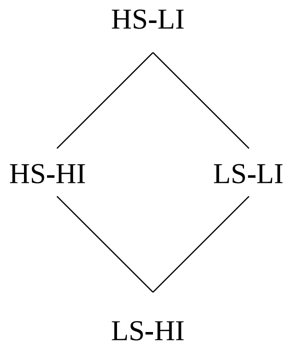
Example 2 *(Strict ⋆-Property)* Identical to example 1 except

- RH is as shown in figure 11(b)
- each user is assigned in UA to all roles of form yW (since all of these are now maximal roles) □

Now the permission (o,w) is no longer inherited by other roles as is the case in example 1. Extensions of this construction to lattices with trusted write range are given in [San96].



(a) Two Independent Lattices



(b) One Composite Lattice

Figure 12: Confidentiality and Integrity Lattices

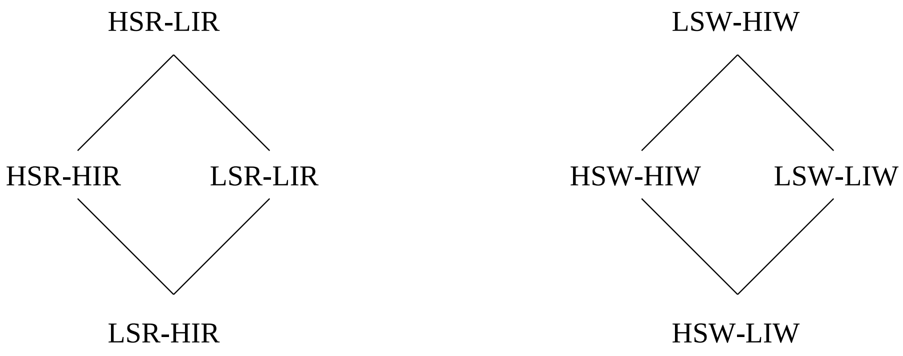
4.3 Composite Confidentiality and Integrity Roles

LBAC was first formulated for confidentiality purposes. It was subsequently observed that if high integrity is at the top of the lattice and low integrity at the bottom then information flow should be downward rather than upward (as in confidentiality lattices). In [San93] it is argued that it is simpler to fix the direction of information flow and put high integrity at the bottom and low integrity at the top in integrity lattices. Because the confidentiality models were developed earlier we might as well stay with lattices in which information flow is always upwards.

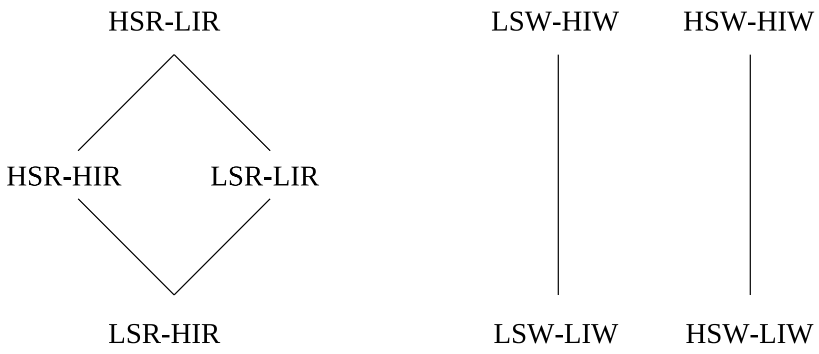
Figure 12(a) shows two independent lattices. The one on the left has HS (high secrecy) on the top and LS (low secrecy) on the bottom. The one on the right has LI (low integrity) on the top and HI (high integrity) on the bottom. In both lattices information flow is upward. The two lattices can be combined into the single composite lattice shown in figure 12(b).[8]

One complication in combining confidentiality and integrity lattices (or multiple lattices in general) is that these lattices may be using different versions of the $\star$-property. We have discussed earlier that the strict $\star$-property is often used in confidentiality lattices due

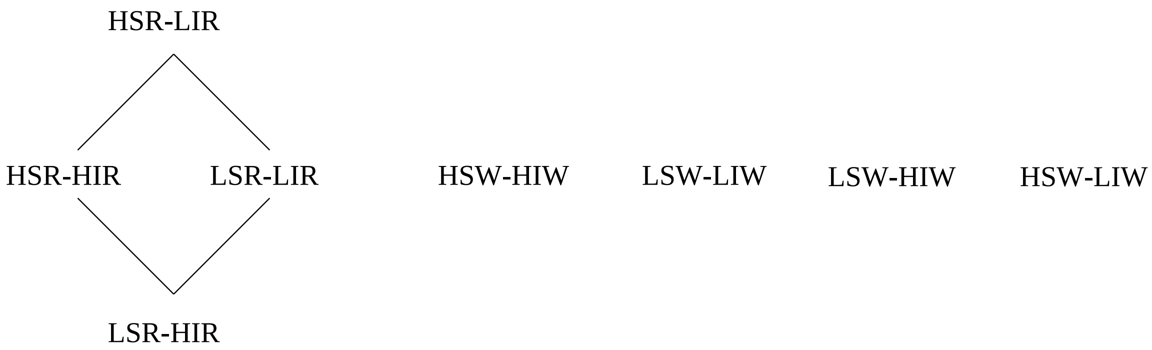
[8]It is always possible to mathematically combine multiple lattices into a single lattice.



(a) Liberal Confidentiality and Liberal Integrity



(b) Strict Confidentiality and Liberal Integrity



(c) Strict Confidentiality and Strict Integrity

Figure 13: Composite Confidentiality and Integrity Roles

to integrity considerations. In integrity lattices there is no similar need to use the strict $\star$-property, and one would expect to see the liberal $\star$-property instead.

Consider the composite lattice of figure 12(b).[9] The RBAC realization of three combinations of liberal or strict $\star$-properties are shown in figure 13.[10] Since the simple security property does not change we have a similar role hierarchy for the read roles shown on the left hand side of the three role hierarchies of figures 13(a), (b) and (c). In each case the hierarchy for the write roles needs to be adjusted as shown on the right hand side of each of these figures. The constructions are formally described below.

Example 3 *(Liberal Confidentiality and Liberal Integrity $\star$-Property)*

- $R =$ {HSR-LIR, HSR-HIR, LSR-LIR, LSR-HIR, HSW-LIW, HSW-HIW, LSW-LIW, LSW-HIW}
- RH as shown in figure 13(a)
- Constraint on UA: Each user is assigned to exactly one role xSR-yIR and all maximal write roles.
- Constraint on sessions: Each session has exactly two roles uSR-vIR and uSW-vIW
- Constraints on PA:
  - (o,r) is assigned to xSR-yIR iff (o,w) is assigned to xSW-yIW
  - (o,r) is assigned to exactly one role xSR-yIR □

Example 4 *(Strict Confidentiality and Liberal Integrity $\star$-Property)* Identical to example 3 except that RH is as shown in figure 13(b). □

Example 5 *(Strict Confidentiality and Strict Integrity $\star$-Property)* Identical to example 3 except that RH is as shown in figure 13(c). □

The constructions indicate how a single pair of roles can accommodate lattices with different variations of the $\star$-property. The construction can clearly be generalized to more than two lattices.

4.4 Discussion

In this section we have shown how different variations of lattice based access controls (LBAC) can be simulated in role-based access control (RBAC). RBAC is itself policy neutral but can be easily configured to specify a variety of policies as we have shown. The main components of RBAC that need to be adjusted for different LBAC variations are the role hierarchy and constraints. This attests to the flexibility and power of RBAC.

[9] Similar constructions for the distinct lattices of figure 12(a) are given in [San96].

[10] The fourth combination of liberal confidentiality and strict integrity could be easily constructed but is rather unlikely to be used in practice so is omitted.

A practical consequence of our results is that it might be better to develop systems that support general RBAC and specialize these to LBAC. RBAC has much broader applicability than LBAC, especially in the commercial sector. LBAC can be realized as a particular instance of RBAC. This approach provides the added benefit of greater flexibility for LBAC, for which we have seen there are a number of variations of practical interest. In LBAC systems these variations so far require the rules to be adjusted in the implementation. RBAC provides for adjustment by configuration of role hierarchies and constraints instead.

5 THREE-TIER ARCHITECTURE

In this section we present a conceptual framework, or reference architecture, for specifying and enforcing RBAC. Our framework has three tiers in loose analogy to the well-known ANSI/SPARC architecture for database systems [Te78], illustrated in figure 14. Although we take our inspiration from the database domain, we emphasize that our proposed RBAC architecture is germane to applications and systems in general and is not limited to databases per se.

Our reference architecture is motivated by two main considerations. Firstly, a number of RBAC features have been incorporated in commercial products, and more such products can be expected to appear in future. Vendors tend to integrate RBAC facilities in products in different ways, because of the economics of integrating such features into existing product lines. Over time the emergence of standards may impose some order in this arena, but the near term is likely to display a divergence of approaches. Even as standards emerge, we can expect a diversity of support for RBAC due to the longevity of legacy systems.

Secondly, in large organizations there will be a large number of roles and complex relationships between the roles and permissions authorized by them. In most contexts it would be appropriate to take a simplified view appropriate for the task at hand. For example, in some situations all members of a particular department can be treated as belonging to a single role; whereas in other situations more refined roles such as managers, technical staff and administrative staff need to be distinguished.

The central tier of our architecture resides in a single community view of RBAC as it applies to the entire organization in question. This community view will typically be large and complex reflecting the reality of modern organizations. The specialized context-specific views of RBAC tailored to particular applications and situations are accommodated in multiple user views that reside above the central tier. The views of RBAC embodied in different products are embodied in multiple implementation views residing below the implementation tier. Figure 15 illustrates these three tiers. The central tier serves as the focal point for mapping the external user views to the internal implementation views.

5.1 The Three Tiers

The ANSI/SPARC report [Te78] described a three-tier architecture for a database, consisting of:

1. the external or user view which is concerned with the way data is viewed by end users,

2. the conceptual or community view which amalgamates diverse external views into a consistent and unified composite, and
3. the internal or implementation view which is concerned with the way that data is actually stored.

This database architecture is shown in figure 14.

Note that there are multiple external views, but only a single conceptual and a single internal view. This three-tier approach to database systems has stood the test of time, and is remarkably independent of the particular data model being used.

We believe a similar approach is suitable for developing a common framework or reference architecture for RBAC. RBAC is concerned with the meaning and control of access control data (i.e., data used to control access to the actual data of the organization). In other words we are concerned with a special purpose database system. It is therefore sensible to adapt the approach used for general-purpose database systems. However, there is one significant difference. In database systems, it is intended that the implementation will eventually be on a particular database management platform. Consequently, the internal or implementation view is closely tied to the particular platform that is selected. With RBAC we do not have the luxury of assuming a homogeneous implementation environment. Instead we must confront the reality of heterogeneous implementations up front. This leads us to modify the three-tier ANSI/SPARC architecture by introducing multiple internal views, corresponding to different platforms on which the implementation is done. This RBAC reference architecture is shown in figure 15.

Our three-tiered approach to RBAC therefore consists of multiple external views, a single conceptual view, and multiple implementation views.

Next, let us consider the appropriate model for each of these tiers. We again turn to the ANSI/SPARC architecture for inspiration. There is a conspicuous difference between the models used at the implementation and conceptual tiers. We expect a similar difference in our RBAC reference architecture. Why is this so? We expect the model used at the conceptual level to have richer constructs and primitives, because it is intended to express a composite system-wide view of RBAC. Practical considerations will inevitably dictate that not all these features can be directly supported in an implementation. Hence the implementation models will be simpler and less user-friendly. Moreover, we expect a range of sophistication from rather primitive mechanisms (say on a vanilla UNIX platform) at one end to very elaborate ones (say on an object-oriented database management system) at the other. Note that this viewpoint lets us accommodate legacy systems co-existing with newer ones. It should also be clear that the effort required to translate a conceptual view will be less or greater depending upon the sophistication of the implementation platform being targeted. In some cases, a translation may not even be feasible (or practical) without enhancement of the target platform.

The difference between the conceptual and external tiers is less marked. Whether or not there should be any difference is open to debate. For relational databases, both tiers are often identical and directly based on the relational data model. However, sometimes a richer model such as the entity-relationship model is used for the external view while a relational model is used at the conceptual view. We anticipate a similar situation in the

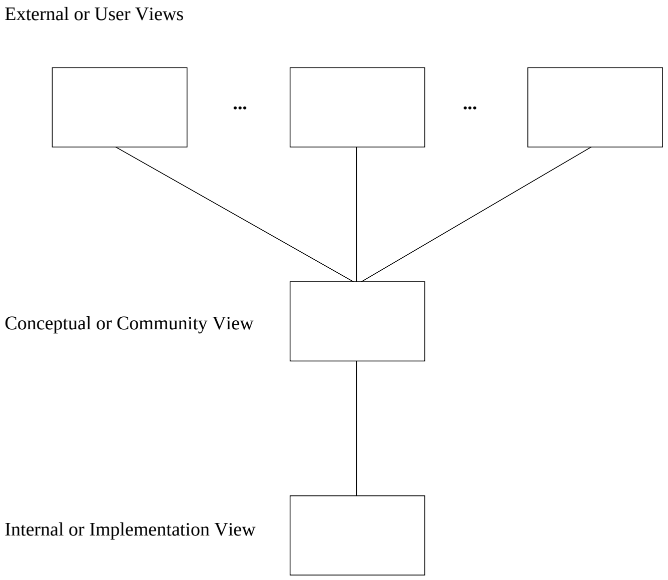


Figure 14: ANSI/SPARC Database Architecture

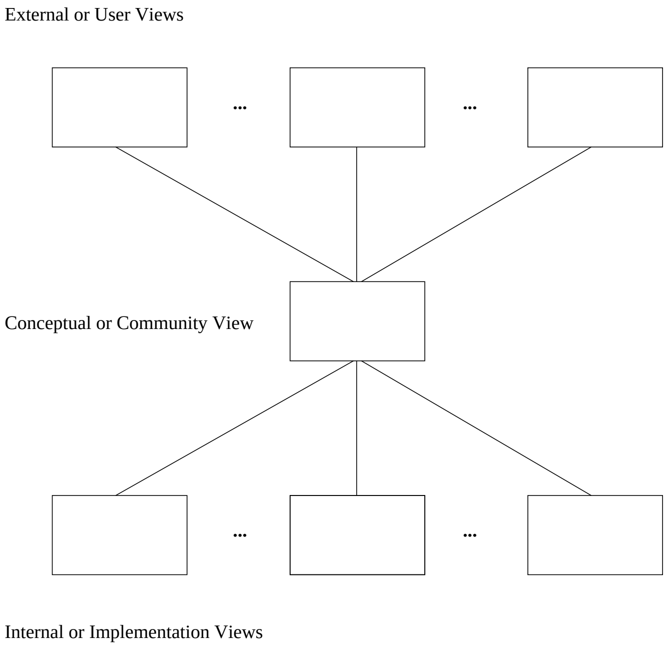


Figure 15: A Three Tier Architecture for RBAC

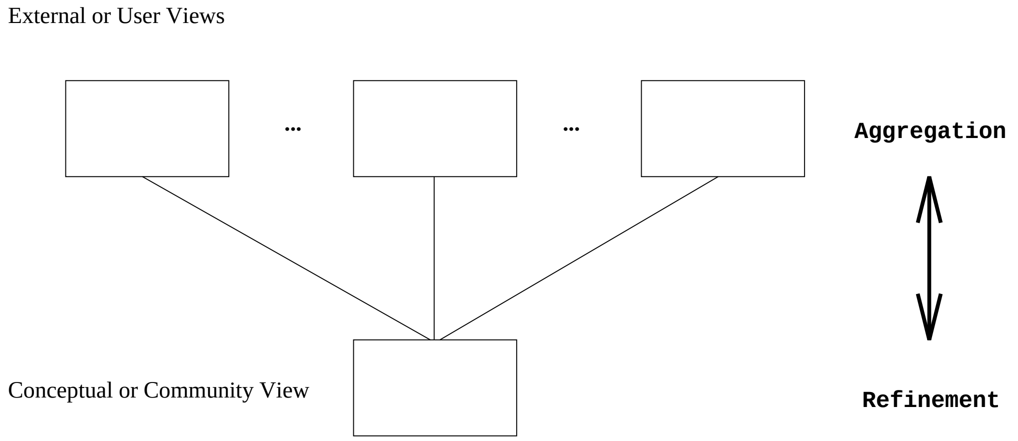


Figure 16: Harmonizing the Top Two Tiers

RBAC reference architecture. Based on the historical experience with the ANSI/SPARC architecture, it might well happen that initially the same RBAC model is used at both tiers, but over time richer models are developed for the external view.

5.2 The Central Tier

The central tier of our reference architecture consists of a single community view of RBAC applicable to the entire information system and its myriad applications. This community view is the essential conceptual vehicle for effective deployment of enterprise-wide RBAC. The RBAC96 family of models provides us a flexible and general framework for this tier.

5.3 Harmonizing the Top Two Tiers

Let us now consider the relationship between the top two tiers of the reference architecture, reproduced in figure 16. Each external view gives one perspective on the common community view, relevant to the particular context at hand. The relationship between the the top two tiers is one of aggregation and refinement as indicated in the figure.

Aggregation is a process by which several distinct roles are combined into a single role, because the distinction is not relevant in the given context. For example, the community view might have distinct roles for, say, Undergraduate Students, Master's Students and Doctoral Students. In an application where are all students are treated alike, these roles could be collapsed (i.e., aggregated) into a single Student role. In other applications, which confer different privileges to the various student roles, this distinction is significant. Refinement is simply the opposite operation to aggregation.

Different external views will aggregate different collections of roles from the community view. Some external views may aggregate the student roles into a single one. Others may keep the distinction between student roles but aggregate distinct faculty roles into one.

Still others may aggregate both or none of the student and faculty roles. Our expectation is that a relatively small portion of the overall role set from the community view will be needed more or less intact in a particular external view. Most of the roles will, however, be aggregated. In other words each external view will see only a small part of the roles set in all its detail.

So long as entire roles are being aggregated or refined, the mapping between the top two tiers is relatively simple. There may be situations where the role relevant to the external view does not come about so cleanly by aggregation. For example, suppose the community view has roles A and B, whereas the external view requires a role which has some (but not all) members of A and some (but not all) members of B. We identify below some techniques for accommodating such an external view.

- One could modify the community view to create a new role C and explicitly assign those members of A and B who should belong to this role. This treats A, B and C as unrelated roles.
- One could modify the community view to partition A into A_1 and A_2 (with $A_1 \cap A_2 = \emptyset$), and B into B_1 and B_2 (with $B_1 \cap B_2 = \emptyset$) so that the aggregate role $C = A_1 \cup B_1$ can be defined in the desired external view. This would require external views which use A to now treat A as an aggregate of A_1 and A_2, instead of being a role form the community view. Similarly, for external views which use role B.
- We could allow aggregation which can select the appropriate subsets of A and B, based on some condition for identifying members who should belong to the aggregated role C. This will complicate the aggregation operation and might dilute the central role of the conceptual view.

This list is not intended to be exhaustive. The point is that various alternatives are available as the community and external views adapt to the ever changing demands of the applications. One needs a systematic methodology for dealing with such changes.

5.4 Harmonizing the Bottom Two Tiers

Now consider harmonization of the bottom two tiers, shown in figure 17. Each of the implementation views will aggregate roles from the community view. The aggregation done here will constrain which external views can be hosted on which implementation views. An implementation view that aggregates distinct student roles into a single role obviously cannot support an external view that requires this distinction to be maintained. In an ideal situation the implementation view may do no aggregation, in which case it could support all the external views. In practice, however, one would expect considerable aggregation to occur; if only because of legacy systems which have directly built in the external view without consideration of the common community view. Performance considerations may also require such aggregation to occur. Note that in both figures 16 and 17 aggregation is in the direction away from the central community view, and refinement is directed towards this view.

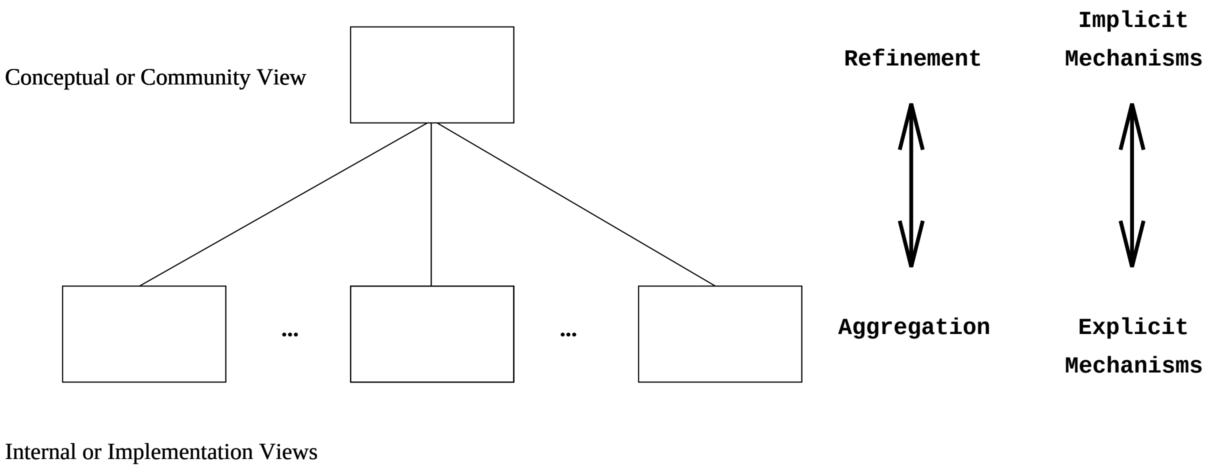


Figure 17: Harmonizing the Bottom Two Tiers

The second mapping shown in figure 17 is between implicit and explicit mechanisms. This mapping recognizes that the implementation platform may not support all the features of RBAC in the community view. For example, role hierarchies may not be supported. Suppose there are two roles Faculty and Staff such that every member of the Faculty role is automatically a member of the Staff role (but not vice versa). Thus a new faculty member need only be enrolled in the Faculty role, and will automatically be enrolled in the Staff role. Support for such role inheritance in the community view is highly desirable, but such support will not be available on every implementation platform. To continue our example, at the community view it suffices to enroll a new faculty member into the Faculty role. However, in the implementation view the new faculty member will need to be enrolled in both Faculty and Staff roles. Similarly, a departing faculty member needs to be removed from the Faculty role in the community view; but in the implementation view requires removal from both Faculty and Staff roles.

5.5 Discussion

In this section we have described a three-tiered reference architecture for role-based access control (RBAC), and have identified some of the issues that need to be addressed in making this framework a reality. We note that the appeal of RBAC is in the simplification of the management of authorizations. For example, maintaining cognizance of the permission set of an individual and the consequence of assigning particular role sets to a user is vital. It is also important for a security administrator to know exactly what authorization is implied by a role. This is particularly so when roles can be composed of other roles. Moreover, as new roles and transactions are introduced the security administrator needs tools to assist in their integration into the existing system. Future work in RBAC should identify useful tools for security administration and point the way toward designing these. We feel the central role of the community view in our reference architecture will greatly assist in this objective.

6 CONCLUSION

This article has described the motivations, results and open issues in recent RBAC research focusing on the RBAC96 family of models, the ARBAC97 administrative models, the flexibility and power of RBAC in simulating variations of classical lattice-based control, and a three tier conceptual architecture for enforcing RBAC in large heterogeneous environments. Although the basic ideas of role-based access control are very simple, we hope to have convinced the reader that there are a number of interesting technical challenges and much further research to be done.

In conclusion we would like to note two major areas that need considerable work to fully realize the potential of RBAC. The first area is that of *role engineering*, that is the discipline and methodology for configuring RBAC in an organization, and most importantly designing the role hierarchy. The second area is *role transition* meaning the means for moving towards RBAC in coexistence with legacy modes of access control.

Acknowledgment

This work was funded in part by contracts 50-DKNA-4-00122 and 50-DKNB-5-00188 from the National Institute of Standards and Technology through SETA Corporation, and by grant CCR-9503560 from the National Science Foundation through George Mason University. The author acknowledges the assistance of the following people in conducting this research: Venkata Bhamidipati, Edward Coyne, Hal Feinstein, Srinivas Ganta and Charles Youman. The author also acknowledge the following people for useful discussions on these topics: John Barkley, David Ferraiolo, Serban Gavrila and Roshan Thomas.

References

[Bal90] Robert W. Baldwin. Naming and grouping privileges to simplify security management in large database. In *Proceedings of IEEE Symposium on Research in Security and Privacy*, pages 61–70, Oakland, CA, April 1990.

[Bel87] D.E. Bell. Secure computer systems: A network interpretation. In *Proceedings of 3rd Annual Computer Security Application Conference*, pages 32–39, 1987.

[Com96] Common Criteria Editorial Board. *Common Criteria for Information Technology Security*, January 1996. Version 1.0.

[FGL93] David F. Ferraiolo, Dennis M. Gilbert, and Nickilyn Lynch. An examination of federal and commercial access control policy needs. In *Proceedings of NIST-NCSC National Computer Security Conference*, pages 107–116, Baltimore, MD, September 20-23 1993.

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

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

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

[ISO] ISO/IEC 10040. *Information Technology – Open Systems Interconnection – Systems Management Overview.*

[Jon93] Dirk Jonscher. Extending access controls with duties—realized by active mechanisms. In B. Thuraisingham and C.E. Landwehr, editors, *Database Security VI: Status and Prospects*, pages 91–111. North-Holland, 1993.

[Lee88] T.M.P. Lee. Using mandatory integrity to enforce "commercial" security. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 140–146, Oakland, CA, May 1988.

[MD94] Imtiaz Mohammed and David M. Dilts. Design for dynamic user-role-based security. *Computers & Security*, 13(8):661–671, 1994.

[MS91] Jonathan D. Moffett and Morris S. Sloman. Delegation of authority. In I. Krishnan and W. Zimmer, editors, *Integrated Network Management II*, pages 595–606. Elsevier Science Publishers B.V. (North-Holland), 1991.

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

[Not97] L. Notargiacomo. Role-based access control in ORACLE7 and Trusted ORACLE7. In *Proceedings of the 1st ACM Workshop on Role-Based Access Control*. ACM, 1997.

[San88] Ravi S. Sandhu. Transaction control expressions for separation of duties. In *Proceedings of 4th Annual Computer Security Application Conference*, pages 282–286, Orlando, FL, December 1988.

[San91] Ravi S. Sandhu. Separation of duties in computerized information systems. In S. Jajodia and C.E. Landwehr, editors, *Database Security IV: Status and Prospects*, pages 179–189. North-Holland, 1991.

[San92] Ravi S. Sandhu. The typed access matrix model. In *Proceedings of IEEE Symposium on Research in Security and Privacy*, pages 122–136, Oakland, CA, May 1992.

[San93] Ravi S. Sandhu. Lattice-based access control models. *IEEE Computer*, 26(11):9–19, November 1993.

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

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

[SBC+97] Ravi Sandhu, Venkata Bhamidipati, Edward Coyne, Srinivas Ganta, and Charles Youman. The arbac97 model for role-based administration of roles: Preliminary description and outline. In *Proceedings of the 2nd ACM Workshop on Role-Based Access Control*. ACM, 1997.

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

[Sch88] W.R. Schockley. Implementing the Clark/Wilson integrity policy using current technology. In *Proceedings of NIST-NCSC National Computer Security Conference*, pages 29–37, 1988.

[SF94] Ravi S. Sandhu and Hal L. Feinstein. A three tier architecture for role-based access control. In *Proceedings of 17th NIST-NCSC National Computer Security Conference*, pages 34–46, Baltimore, MD, October 11-14 1994.

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

[Te78] D.C. Tsichritizis and A. Klug (editors). The ANSI/X3/SPARC DBMS framework: Report of the study group on data base management system. *Information Systems*, 3, 1978.

[Tho91] D.J. Thomsen. Role-based application design and enforcement. In S. Jajodia and C.E. Landwehr, editors, *Database Security IV: Status and Prospects*, pages 151–168. North-Holland, 1991.

[TS94] Roshan Thomas and Ravi S. Sandhu. Conceptual foundations for a model of task-based authorizations. In *Proceedings of IEEE Computer Security Foundations Workshop 7*, pages 66–79, Franconia, NH, June 1994.

[TS97] Roshan Thomas and Ravi Sandhu. Task-based authorization controls (tbac): Models for active and enterprise-oriented authorization management. In T. Y. Lin and Xiaolei Qian, editors, *Database Security XI: Status and Prospects*. North-Holland, 1997.

[vSvdM94] S. H. von Solms and Isak van der Merwe. The management of computer security profiles using a role-oriented approach. *Computers & Security*, 13(8):673–680, 1994.