Role-Based Access Control Models[*†‡]

Ravi S. Sandhu[¶], *Edward J. Coyne*[‖], *Hal L. Feinstein*[‖] *and Charles E. Youman*[‖]

Revised October 26, 1995

Abstract This article introduces a family of reference models for role-based access control (RBAC) in which permissions are associated with roles, and users are made members of appropriate roles. This greatly simplifies management of permissions. Roles are closely related to the concept of user groups in access control. However, a role brings together a set of users on one side and a set of permissions on the other, whereas user groups are typically defined as a set of users only.

The basic concepts of RBAC originated with early multi-user computer systems. The resurgence of interest in RBAC has been driven by the need for general-purpose customizable facilities for RBAC and the need to manage the administration of RBAC itself. As a consequence RBAC facilities range from simple to complex. This article describes a novel framework of reference models to systematically address the diverse components of RBAC, and their interactions.

Keywords: security, access control, roles, models

---

[*]This paper has been accepted for publication in *IEEE Computer*.

[†]All correspondence should be addressed to Prof. Ravi Sandhu, ISSE Department MS 4A4, George Mason University, Fairfax, Va 22030. Phone: 703-993-1659, fax: 703-993-1638, email: `sandhu@isse.gmu.edu`.

[‡]This work is funded in part by contracts 50-DKNA-4-00122 and 50-DKNB-5-00188 from the National Institute of Standards and Technology. The work of Ravi Sandhu is also supported by grant CCR-9503560 from the National Science Foundation.

[¶]SETA Corporation and George Mason University

[‖]SETA Corporation

1 INTRODUCTION

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

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.

Several distinct motivations for constructing a role are discussed below. 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. Jane Doe 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.

A recent study by NIST [1] demonstrates 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 draft Common Criteria [2]. 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.

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

This paper describes a novel framework of four reference models developed by the authors to provide a systematic approach to understanding RBAC, and to categorizing its implementation in different systems. Our framework also separates the administration of RBAC from its use for controlling access to data and other resources.

2 BACKGROUND AND MOTIVATION

A major purpose of RBAC is to facilitate security administration and review. Many commercially successful access control systems for mainframes implement roles for security administration. For example, an operator role could access all resources but not change access permissions, a security-officer role could change permissions but have no access to resources, and an auditor role could 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. In the past, and today, specific applications have been 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 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 [1] 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 piece of data in the system. RBAC components may be configured directly by the system owner or indirectly by appropriate roles as delegated by the system owner. The policy enforced in a particular system is the net result of the precise configuration of various RBAC components as directed by the system owner. Moreover, the access control policy can evolve incrementally over the system life cycle, and in large systems it is almost certain to do so. 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 could configure RBAC so it violates these principles. Also, the degree to which data abstraction is supported will be determined by the implementation details.

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 [3] and Thomas and Sandhu [4] have discussed some of these issues. We view control of sequences of operations to be outside the scope of RBAC, although RBAC can be a foundation on which to build such controls.

3 ROLES AND RELATED CONCEPTS

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.

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.

A question is often asked concerning the relationship of roles to compartments. Compartments are a part of the security label structure as used in the classified defense and government sectors [5]. Compartments are based on the notion of need-to-know, which has a semantic connotation regarding the information available under a compartment label analogous to the semantic connotation of role. However, the use of compartments is for the specific policy of one-directional information flow in a lattice of labels. Roles do not presume a particular policy of this kind.

A long-standing distinction exists between discretionary and mandatory access controls, respectively known as DAC and MAC. This distinction emerged from security research in the defense sector. MAC enforces access controls on the basis of security labels attached to users (more precisely, to subjects) and objects [5]. DAC enforces access control to an object on the basis of permissions or denials or both configured by an individual user, typically the object's owner. RBAC can be viewed as an independent component of access control, coexisting with MAC and DAC when appropriate. In such a case access is allowed if and only if permitted by RBAC, MAC, and DAC. We also expect that RBAC in many cases will exist by itself.

As a related issue, is RBAC itself a discretionary or a mandatory mechanism? The answer depends on the precise definition of discretionary and mandatory as well as on the precise nature and configuration of permissions, roles, and users in an RBAC system. We understand mandatory to mean that individual users do not have any choice regarding which permissions or users are assigned to a role, whereas discretionary signifies that individual users make these decisions. As we said earlier, RBAC is policy-neutral by itself. Particular configurations of RBAC can have a strong mandatory flavor, while others can have a strong discretionary flavor.

4 A FAMILY OF REFERENCE MODELS

To understand the various dimensions of RBAC we define a family of four conceptual models. The relationship between these four models is shown in Figure 1(a) and their essential characteristics portrayed in Figure 1(b). $RBAC_0$, the base model, is at the bottom, indicating that it is the minimum requirement for any system that professes to support 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, $RBAC_0$.

These models are intended to be reference points for comparison with systems and models used by other researchers and developers. They can also serve as 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. Later we will introduce a more sophisticated management model.

4.1 BASE MODEL

The base model $RBAC_0$ consists of that part of Figure 1(b) not identified with one of the three advanced models. 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, 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 greatly 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 things 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 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.

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 joined 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,
- $user : S \rightarrow U$, a function mapping each session s_i to the single user $user(s_i)$ (constant for the session's lifetime), and
- $roles : S \rightarrow 2^R$, 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 role to be assigned at least one permission and each user to be assigned to at least one role. The model, however, does not require this.

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 $RBAC_2$.)

Some authors [6] include duties, 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. We feel that incorporation of concepts such as duties in access control models requires further research. 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 [4].

4.2 ROLE HIERARCHIES

The model $RBAC_1$ introduces role hierarchies (RH), as indicated in Figure 1. Role hierarchies are almost inevitably included whenever roles are discussed in the literature [7, 8, 9, 10]. They are also commonly implemented in systems that provide roles.

Role hierarchies are a natural means for structuring roles to reflect an organization's 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 $\geq$, 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 $\cup_{r \in roles(s_i)}\{p \mid (\exists r'' \leq r)[(p, r'') \in PA]\}$. $\square$

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 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 test engineer′ and relating it to test engineer as shown in Figure 2(c). The private permissions of test engineers can be 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 upward 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.

4.3 CONSTRAINTS

Model $RBAC_2$ introduces the concept of constraints as shown 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. This is indicated by 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 (except the very smallest) 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 laying out 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 then 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.

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 are less likely to occur in practice. The dual constraint on permission assignment applies more at the role end of the PA relation. 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.

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.

4.4 CONSOLIDATED MODEL

$RBAC_3$ 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, as indicated by the dashed arrow to RH in Figure 1(b). 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 always be specified without raising any 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.

5 MANAGEMENT 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. Here we can only touch on some of the major issues.

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 [11]. 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 [12]. 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 [13] 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.

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 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.

Administrative authority in RBAC can be viewed as the ability to modify the user assignment, permission assignment and role hierarchy relations. In a management model the permissions that authorize these administrative operations must be explicitly defined. The precise nature of these permissions is implementation specific, but their general nature is much the same.

One of the main issues in the 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 entirely 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.

6 CONCLUSION

We have presented a family of RBAC models that systematically spans the spectrum from simple to complex. These models provide a common frame of reference for other research and development in this area. We have also presented a management model whereby RBAC can be used to control itself. This supports our position that RBAC is policy-neutral, rather than a model of a specific security policy.

Much remains to be done to realize the promise of RBAC. One of the outstanding research problems in this area is to develop a systematic approach to the design and analysis of RBAC configurations. Some recent research on the design and analysis of role hierarchies has been reported [8, 9, 14]. As mentioned earlier, there is little discussion in the literature about constraints in the context of RBAC. A categorization and taxonomy of constraints would be useful. A formal notation for stating and enforcing constraints, along with some measure of difficulty of enforcement, should be developed. The ability to reason about constraints and analyze the net effect of an RBAC configuration in terms of higher-level policy objectives is an important open research area. The management aspects of RBAC need further work. Development of a systematic methodology that deals with the design and analysis of role hierarchies, constraints, and RBAC management in a unified framework is a challenging research goal. Many of these open issues and problems are intertwined and will require an integrated approach for their resolution.

Acknowledgement The authors are grateful to David Ferraiolo and Janet Cugini of NIST for useful comments while this work was in progress. The authors also thank the anonymous reviewers whose comments and suggestions have significantly improved the paper.

References

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

[2] Common Criteria Editorial Board. *Common Criteria for Information Technology Security Evaluation*, December 1994. Version 0.9, Preliminary Draft.

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

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

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

[6] 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.

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

[8] 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 et al, editors, *Database Security VIII: Status and Prospects*. North-Holland, 1995.

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

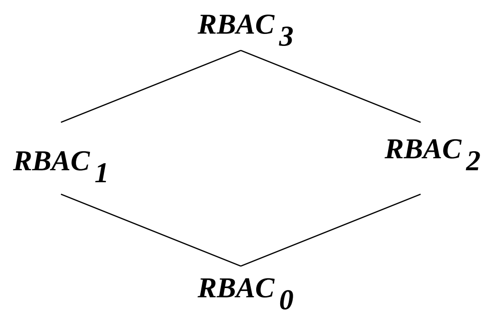
[10] S. H. von Solms and I. van der Merwe. The management of computer security profiles using a role-oriented approach. *Computers & Security*, 13(8):673–680, 1994.

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

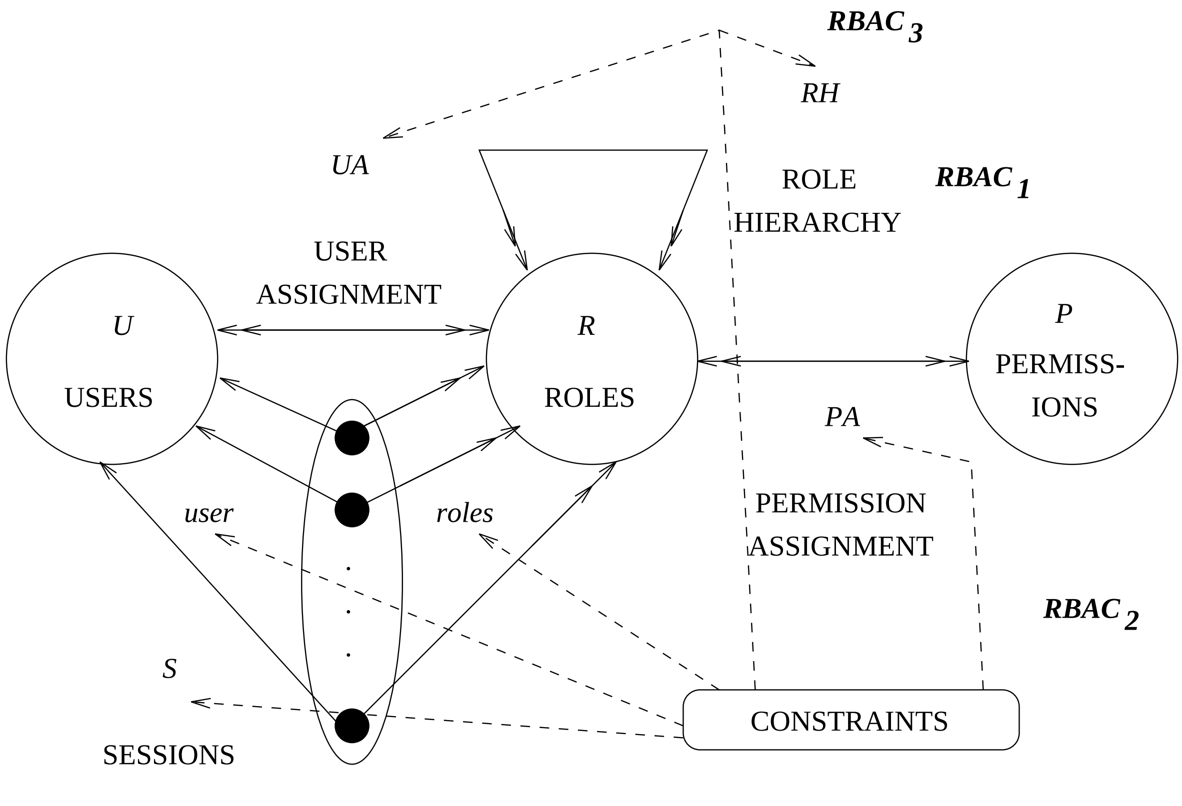
[12] Ravi S. Sandhu. The typed access matrix model. In *Proceedings IEEE Computer Society Symposium on Research in Security and Privacy*, pages 122–136, Oakland, CA, May 1992.

[13] 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.

[14] Eduardo B. Fernandez, Jie Wu, and Minjie H. Fernandez. User group structures in object-oriented database authorization. In J. Biskup et al, editors, *Database Security VIII: Status and Prospects*. North-Holland, 1995.



(a) Relationship among RBAC models



(b) RBAC models

Figure 1: A Family of RBAC Models

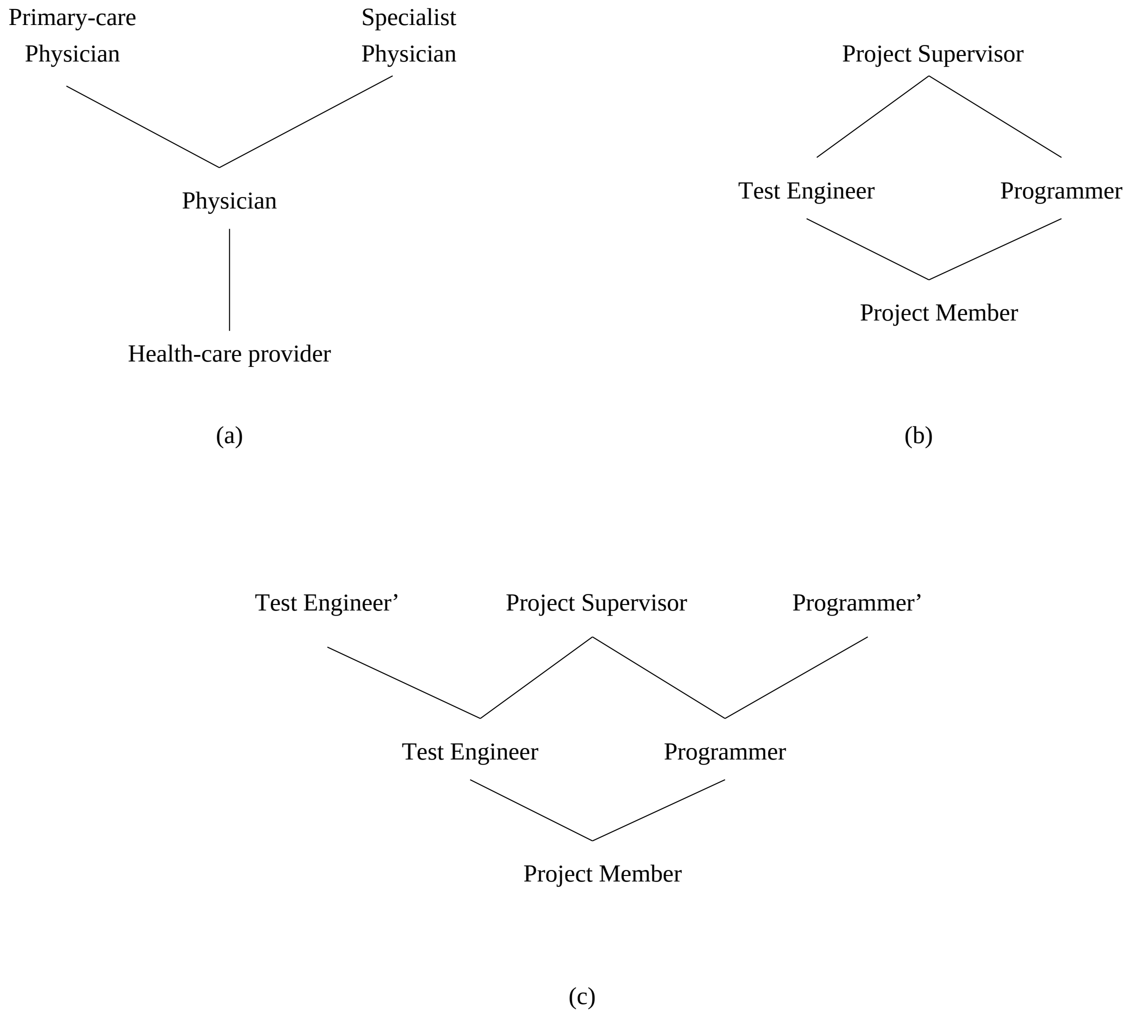
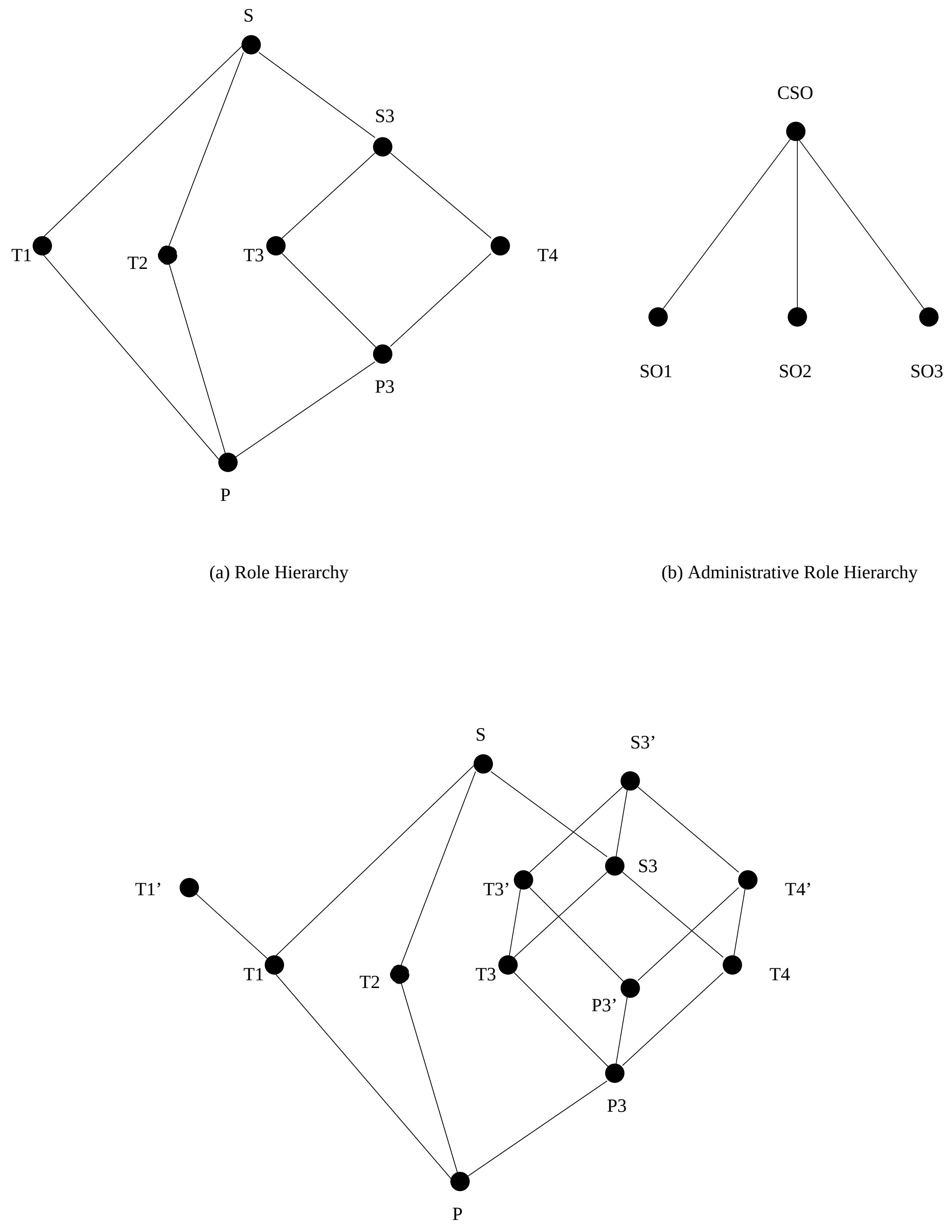


Figure 2: Examples of Role Hierarchies



(a) Role Hierarchy

(b) Administrative Role Hierarchy

(c) Private and Scoped Roles

Figure 3: Role Hierarchies for a Project

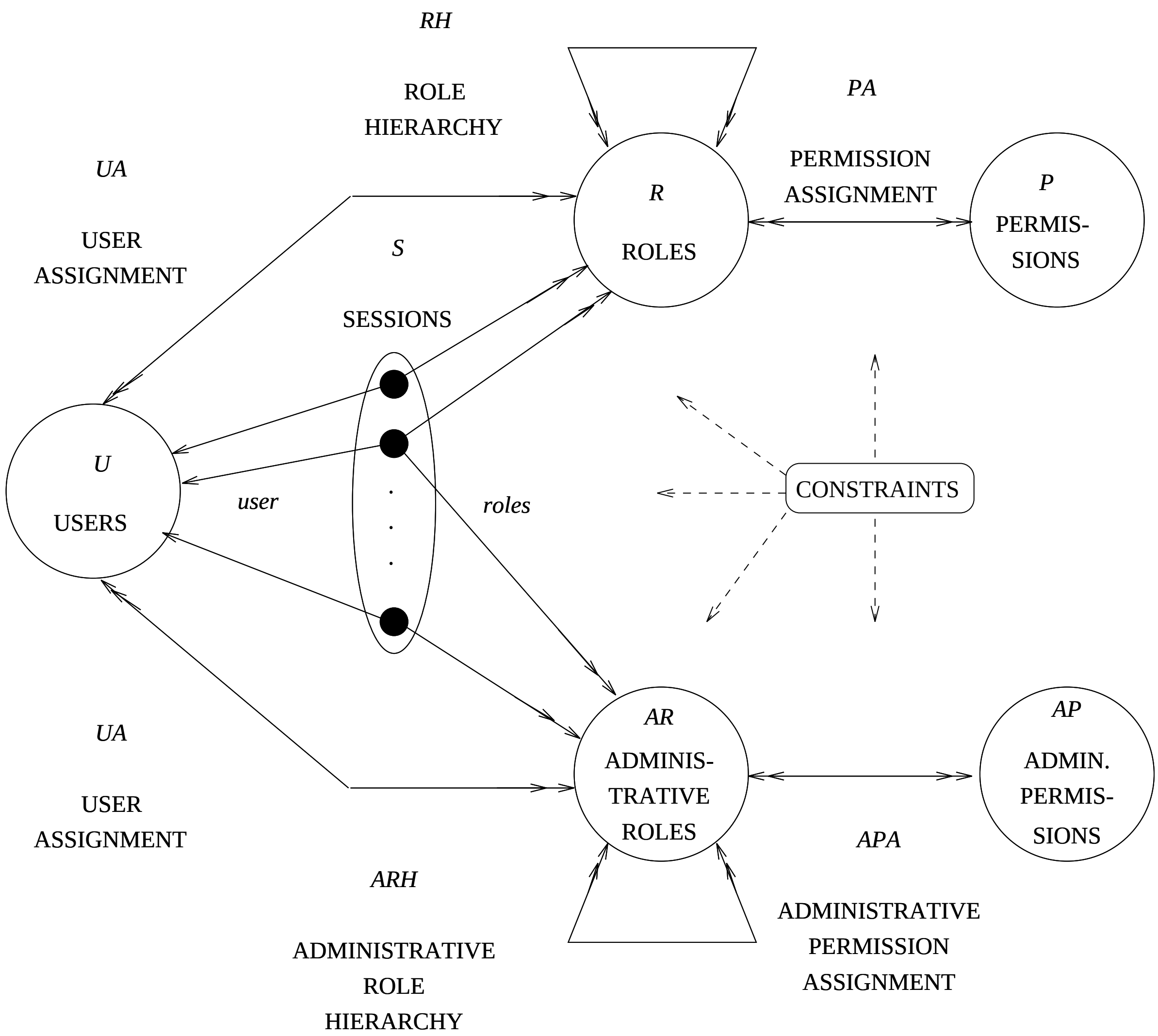


Figure 4: RBAC Administrative Model