A Role Based Access Control Model and Reference Implementation within a Corporate Intranet

David F. Ferraiolo, John F. Barkley, and D. Richard Kuhn
National Institute of Standards and Technology
Gaithersburg, Maryland 20899-8970 USA

This paper describes NIST's enhanced RBAC model and our approach to designing and implementing RBAC features for networked Web servers. The RBAC model formalized in this paper is based on the properties that were first described in [1], and [2], with adjustments that have resulted through experiences gained by prototype implementations, market analysis, and observations made by Jansen [3], and Hoffman [4]. The implementation of RBAC for the Web (RBAC/Web) provides an alternative to the conventional means of administering and enforcing authorization policy on a server-by-server basis. RBAC/Web provides administrators with a means of managing authorization data at the enterprise level, and in a manner that is consistent with the current set of laws, regulations, and practices that face businesses today.

Categories and Subject Descriptors: C.2.4 [**Computer Systems Organization**]: Computer-communication Networks —*Distributed Systems*; C.2.5 [**Computer Systems Organization**]: Computer-communication Networks —*Local and Wide-area Networks*; C.5.5 [**Computer Systems Organization**]: Computer System Implementation—*Servers*; D.4.6 [**Software**]: Security and Protection—*Access controls*; D.4.7 [**Software**]: Organization and Design—*Distributed systems*

General Terms: Security, Standardization

Additional Key Words and Phrases: RBAC, role based access, world wide web, web servers

1. INTRODUCTION

In recent years vendors have begun implementing role based access control (RBAC) features into their databases, system management, and operating system products, without any general agreement as to what actually constitutes an appropriate set of RBAC features. Several RBAC models have been proposed [1], [2], [5], [6], without any attempt at standardizing salient RBAC features. To identify RBAC features that both exhibit true enterprise value and are practical to implement, the National Institute of Standards and Technology (NIST) has conducted and sponsored market analysis [7], developed prototype implementations, sponsored external research [8], and published formal RBAC models [1], [2]. As a result of these and other efforts, much has been learned about RBAC, and its practical implementation. The purpose of this paper is to rigorously define NIST's enhanced RBAC model, and to describe its reference implementation within a world wide web (WWW) intranet application (RBAC/Web). An intranet application was chosen to show RBAC's application to a network and to demonstrate its ability to solve common authorization management and policy problems.

1.1 The Existing Problem

One of the greatest obstacles in the growth of intranets as a means of enterprise computing is the inability to effectively manage authorization data. Today, authorization management is costly and prone to error. Web Server administrators usually control user access to enterprise published documents through the creation and maintenance of access control lists (ACLs) on a server-by-server basis. ACLs specify, for each protected resource, a list of named individuals, or groups composed of individual users, with their respective modes of access to that object.

This use of ACLs is problematic for a variety of reasons. ACLs are tied to particular objects. As such, they are appropriate for discretionary need-to-know policies, where ownership of objects resides with the end user. In many enterprises within industry and civilian government, end users do not "own" the information for which they are allowed access [7], [1]. For these organizations, the corporation or agency is the actual "owner" of system objects and discretionary control on the part of the users may not be appropriate. Although enforcing a need-to-know policy is important where classified information is of concern, there exists a general need to support subject-based security policies, such as access based on competency, the enforcement of conflict-of-interest rules, or permitting access based on a strict concept of least privilege. To support such policies requires the ability to restrict access based on a user function or role within the enterprise. Here the relevant question is: "What are the current operational capabilities for this user?"

ACLs further complicate matters by allowing the direct association of users with permissions. A large number of users each with many permissions implies a very large number of user/permission associations which have to be managed. Thus, when a user takes on different responsibilities within the enterprise, reflecting these changes entails a thorough review resulting in the selective addition or deletion of user/permission associations on all servers. The larger the number of user/permission associations to be managed, the greater the risk of maintaining residual and inappropriate user access rights.

Because of the potential problems associated with this lack of operational security assurance, organizations have resisted publishing sensitive information on their Web servers. This limits their utility, depriving the organization of potential productivity gains.

1.2 Solution: Role-Based Access Control

Although it can support discretionary access control (DAC) policies [9], RBAC is primarily a non-discretionary access control model. RBAC does not permit users to be directly associated with permissions.

With RBAC, permissions are authorized for roles and roles are authorized for users. Thus, when administering RBAC two different types of associations must be managed, i.e., associations between users and roles, and associations between roles and permissions. When a user's job position changes, only the user/role associations change. If the job position is represented by a single role, then when a user's job position changes, there are only two user/role associations to change: remove the association between the user and the user's current role and add an association between the user and the user's new role.

1.2.1 *Reduced Administrative Cost Complexity.* There is usually a direct relationship between the cost of administration and the number of associations which must be managed in order to administer an access control policy: the larger the number of associations, the costlier and more error-prone access control administration. In most organizations, the use of RBAC reduces the number of associations which must be managed.

Job positions typically are occupied by more than one individual and most positions require more than one permission in order for an individual in a job position to carry out the responsibilities of that position. One can describe the associations authorizing permissions to individuals who perform the responsibilities of a job position as an ordered pair consisting of a set of individuals and a set of permissions, i.e.,

(U, P) where: $U =$ the set of individuals in a job position
$P =$ the set of permissions required to perform that job position

The number of associations required to directly relate the individuals to those permissions is:

$|U| \cdot |P|$ where: $|U| =$ the number of individuals in the set U
$|P| =$ the number of permissions in the set P

i.e., for each individual in U, there is an association for each permission in P.

A role can be described as a set of permissions. Thus, the set P can refer to a role which is the job position whose user/role and role/permission associations are represented by the ordered pair (U, P). The number of user/role and role/permission associations required to authorize each user in the set U for each of the permissions in the set P where P represents a role is:

$$|U| + |P|$$

i.e., an association with the role P for each individual in U and an association with the role P for each permission in P.

For a job position, if $|U| + |P| < |U| \cdot |P|$, then the administrative advantage of RBAC over relating users directly with permissions is realized for that job position. A sufficient condition for $|U| + |P| < |U| \cdot |P|$ is $|U|, |P| > 2$ which is typically the case for most job positions in most organizations.

If n_{jp} is the number of job positions within an organization, then the administrative advantage of RBAC is realized organization-wide when:

$$\Sigma_i^{n_{jp}}(|U_i| + |P_i|) < \Sigma_i^{n_{jp}}(|U_i| \cdot |P_i|)$$

1.2.2 *Policy Support.* The limiting factor in effectively enforcing security policy is not the capability of the ACL mechanism but the administrative interface. RBAC provides administrators with a context for the specification and enforcement of complex security policies, that are often impractical to enforce through the direct administration of lower level access control mechanisms, such as ACLs.

Unlike ACLs which only support the specification of user/permission and group/permission relationships, the RBAC model supports the specification of user/role and role/role relationships. In particular, the RBAC Model supports the specification of:

—User/role associations, i.e., user authorizations to perform roles

—Role hierarchies, e.g., the role bank teller inherits all of the permissions of the role bank employee

—Separation of Duty constraints (role/role associations indicating conflict of interest):

  —Static (SSD): user cannot be authorized for both roles, e.g., bank teller and auditor

  —Dynamic (DSD): user can be authorized for both roles but cannot act simultaneously in both roles, e.g., a bank teller who has an account in the bank where employed cannot act in the role *teller* and *customer* simultaneously;

—Limits on the number of users which can be authorized for a role (role cardinality), e.g., a branch office of a bank has only one branch manager

For Web server applications, RBAC provides administrative conveniences by composing the seemingly unrelated and incomprehensible authorization data of the lower level access control mechanisms, and other RBAC relational data, into a single RBAC authorization database. In doing so, RBAC/Web, the implementation of the NIST RBAC Model for intranets, organizes this authorization data and presents it to the intranet administrator(s) in a relational format and at a level of abstraction that is natural to the way enterprises are normally structured and conduct business. From an administrator's perspective, RBAC/Web serves as a visualization and maintenance tool [1] of the enterprise's intranet access control policies in terms of its users, roles, role hierarchies, operational constraints, and permissions.

Even with the enhanced administrative interface there exist opportunities for inconsistencies that could result in undesirable security consequences. The complexity of dealing with consistency issues is not delegated to the administrator, but instead is handled by the RBAC/Web administrative software through the implementation of a series of integrity checks that are derived from the RBAC model [10]. Based on RBAC relationships, an interface can be built to solve virtually any authorization problem - a proposition demonstrated by the advanced security policies described in [11] and [5]. Still other real world policies can be expressed by imposing new constraints on time and location. The scope of the RBAC model formalized in this paper has been limited to those properties that were first described in [1] with adjustments that have resulted through observations made by [3] and [4].

The remainder of this paper describes NIST's enhanced RBAC model and our approach to RBAC/Web. Section 2 provides a detailed description of the RBAC Model with exemplary real-world security policies that are supported by the model. Section 3 provides an overview of the administrative approach assumed by the RBAC model and a description of our approach for implementing the RBAC model presented in Section 2. This includes RBAC/Web constituent process components and services, description of the RBAC/Web distributed authorization database,

[1] The RBAC/Web implementation for Windows NT provides a tool, RGP-Admin, to manage role-permission relationships (See Sect. 3.4), but in the UNIX implementation the relationships are managed using the tools of the web server.

and a comprehensive scenario of use, from a client request for URL access (at the browser), through user authorization and role activation, to the result provided by the Web server.

2. THE RBAC MODEL AND SUPPORTED POLICIES

The RBAC security model is both abstract and general. It is abstract because properties not relevant to security are not included; and it is general because many designs could be valid interpretations of the model. Thus, the model allows design decisions to be postponed, and is usable as a basis for the design of a variety of IT systems. A goal in creating the model was to provide as concise and usable a notation as possible so that the security-relevant properties of the model are not obscured by excessive notational detail.

The RBAC model described below is sufficient to support a variety of security policies. In particular, an argument is made for least privilege and separation of duty. Least privilege is the time honored administrative practice of selectively assigning privileges to users such that the user is given no more privilege than is necessary to perform his/her job function. The principle of least privilege avoids the problem of an individual having the ability to perform unnecessary and potentially harmful actions merely as a side-effect of granting the ability to perform desired functions. Permissions (or privileges) are rights granted to an individual, or subject acting on behalf of a user, that enable the holder of those rights to act in the system within the bounds of those rights. The question then becomes how to assign the set of system privileges to the aggregates of functions or duties that correspond to a role of a user or subject acting on behalf of the user. Least privilege provides a rationale for where to install the separation boundaries that are to be provided by RBAC protection mechanisms. Ensuring adherence to the principle of least privilege is largely an administrative challenge, that requires identification of job functions, the specification of the set of privileges required to perform each function, and restricting the user to a domain with those privileges and nothing more.

Separation of duty refers to the partitioning of tasks and associated privileges among different roles associated with a single user to prevent users from colluding with one another. These separation concepts include multiplexing of shared resources, naming distinctive sets of permissions to include functional decomposition, categorical classification, and hierarchical decomposition. Within an RBAC system these separation concepts are supported by the establishment of the principle of least privilege.

The RBAC security model has two components: MC_0, and MC_1. Model component MC_0, called the RBAC Authorization Database model, defines the RBAC security properties for the authorization into static roles. Model component MC_1, called the RBAC Activation model, defines the RBAC security properties for dynamic activation of roles. Component MC_1 requires MC_0. Each model component is defined by the following sub-components:

—a set of *types*, which define a basic set of elements, together with functions on these elements. There are two types of functions: mapping functions, which represent relationships between elements, and constraint functions, which represent restrictions on relationships between elements.

—a set of *rules*, which represent assumptions about the model.

—a set of *properties*, which are implications of the rules

2.1 MC_0: RBAC Authorization Database

The MC_0 model is defined in terms of mapping functions and static properties. Static properties refer to properties of the model that do not involve either Subject or mappings from Subject to other basic elements (e.g., subject-user and active-roles). Static properties represent the most fundamental and the strongest constraints and relationships expressed in the model. They include *role hierarchy, inheritance, cardinality*, and *static separation of duty.*

2.2 Users, Roles and Permissions

RBAC is described in terms of individual users being associated with roles as well as roles being associated with permissions. As such, a role is a means for naming many-to-many relationships among individual users and unique permissions. A user in this model is 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 the user authorized for the role. We will discuss role association with permissions below, but first we will formally model user association with a role by the following types and mapping function:

—type *user* of individual users

—type *role* of role identifiers

—$RM(r : role) \rightarrow 2^{user}$, the role/members mapping, which gives the set of users authorized for the role, r.

A role is called *empty* if and only if $RM[r] = \emptyset$.

A permission or privilege is an approval of a particular operation to be performed on one or more objects. The relationship between roles and permissions is depicted in Figure 1. As shown in Figure 1, arrows indicate a many-to-many relationship (i.e., a permission can be associated with one or more roles, and a role can be associated with one or more permissions). We formally describe permission and the association of permission with a role by the following type and state independent mapping function:

—type $permission = 2^{(operation \times object)}$

—$RP(r : role) \rightarrow 2^{permissions}$, the role/permissions mapping, which gives the set of permissions authorized for the role r.

An operation is an executable image of a program, which upon invocation causes information to flow from, or to, one or more RBAC protected objects or cause the consumption of an exhaustible system resource. The type of operations and the objects that RBAC controls are dependent on the type of system in which it will be implemented. For example: within an operating system, operations might include read, write, and execute; within a database management system, operations might include insert, delete, select, append, and update. Operations are defined using the following type:

—type *operation* of executable program.

The purpose of any access control mechanism is to protect information and other resources. However, in applying RBAC to a computer system, we speak of protected objects. For a system that implements RBAC, the objects can represent information containers (e.g., files, directories, in an operating system, or columns, rows, tables, and views within a database management system) or objects can represent exhaustible system resources, such as printers, disk space, and CPU cycles. The set of objects covered by RBAC includes all of the objects included in the permissions that are associated with roles. For convenience we introduce the following supertype, of which Information-Container, and Exhaustible-Resource are disjoint subtypes:

—type $object = InformationContainer \cup ExhaustibleResource$

The following state independent mapping functions are defined for Permissions:

$POp(p : permission) \rightarrow \{operation\}$, the permission to operation mapping, which gives the set of operations associated with permission p.

$POb(p : permission) \rightarrow \{object\}$, the permission to object mapping, which gives the set of objects associated with permission p.

Figure 1 gives a relational representation of the set of RBAC elements that was originally formalized in [2]. The arrows are used to represent many-to-many relationships (e.g., user-role, and role-operations, role-object). The lines represent relationships that are derived from the model component, MC_0, and as such, are representative of the static relationships among the types as they pertain to the RBAC authorization database.

For notational and conceptual purposes, the ternary relationship between Role, Operation, and Object has been dropped and is reformulated as a pair of binary relations as illustrated in Figure 2: one between Operation and Object, referred to as Permission; the other between Role and Permission. Permission is a construct often associated with information systems, used to control user actions [1]. Permissions are also referred to as privileges in some contexts.

This arrangement provides great flexibility and granularity of assignment of privileges to roles and users to roles. Prior to providing these conveniences it was often the case that a user was granted more access to resources than was needed because of the limited control over the type of access that can be associated with users and resources. Users may need to list directories and modify existing files, for example, without creating new files, or they may need to append records to a file without modifying existing records. Any increase in the flexibility of controlling access to resources also strengthens the application of the principle of least privilege. We now describe relationships between roles.

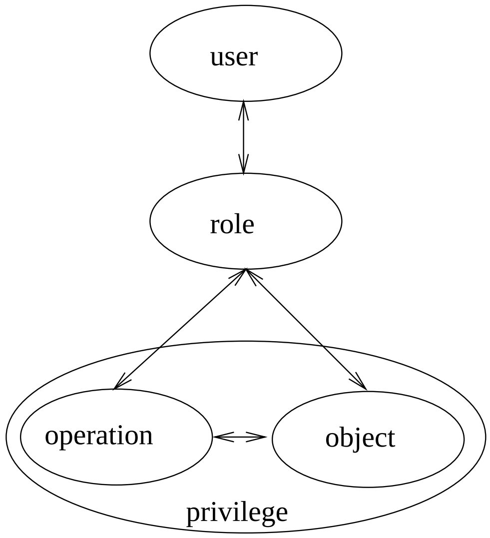

Fig. 1. Role-Operation-Object Relationship

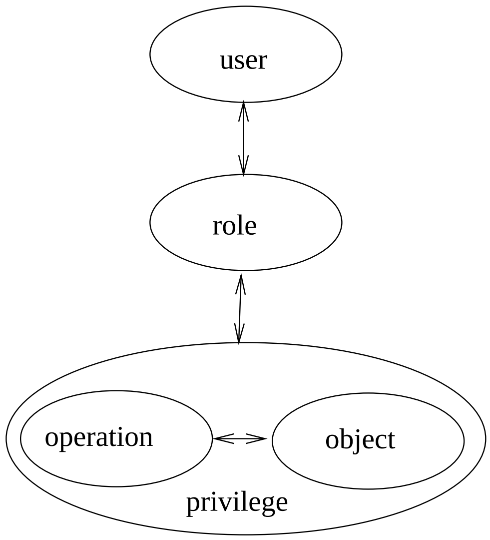

Fig. 2. Model Element Relationships

2.3 Role Membership Hierarchies

An instance of a role represents a many-to-many relationship between individual user members and individual permissions. To address policy and administrative authorization issues, RBAC includes the concept of containment. Containment is similar to inheritance in object-oriented systems, whereby the properties and constraints of a containing role are an extension of the properties and constraints of any contained role. Containment is also recursive; one role can contain other roles, which contain others, and so on. Besides facilitating role administration, containment permits the substitution of role instances. That is, if role A contains role B, written $A \succeq B$, then instances of role A are treated as instances of role B for the purpose of access control. Note that this rule defines inheritance properties for role membership, but not necessarily for role activation (see sec. 2.6). Containment is defined by the following property:

Role Membership Inheritance A containing role is effectively an instance of its contained roles:

$$(\forall i, j : \mathit{role})(\forall u : \mathit{user}) i \succeq j \wedge u \in RM[i] \Rightarrow u \in RM[j] \tag{1}$$

The complete set of containment relationships among all roles is referred to as a role hierarchy. The containment relation is represented as a set of ordered pairs $i \succeq j$, where i is the parent or containing role, and j the child or contained role. We may also say that role i *inherits* role j, or that i is a *senior* and j is a *junior* role. The symbol "$\succeq$" is used to denote that the contains relationship forms a partial ordering of the roles (i.e., a reflexive, transitive, and antisymmetric relation). Thus, $i \succeq j$ means i contains j, or alternatively, j is contained by i.

RBAC relations consist of an *explicit* set of role relationships (e.g., user-role, role-permission) and constraints (e.g., Static Separation of Duty), and an *implicit* set of inherited relationships and constraints. The explicit relationships and constraints for a role are those relationships and constraints that are directly specified through an administrator role. The implicit relationships and constraints of a role are those relationships and constraints that are inherited from other roles within a hierarchy through containment. As such, inheritance has been described in some models in terms of permissions [5], [6]. For example, in [6] a consistent notion of containment is defined in terms of an is-junior relationship, where role, r_i is-junior to r_j, if the privileges (permissions) of r_i are a subset of the privileges of r_j. Implementing a permission inheritance scheme assumes a means of identifying and naming individual permissions, and directly and indirectly associating these permissions with each system role. For a self contained RBAC system, such as a relational database management system, permissions can be centrally identified for any role. For systems that are implemented as a distributed RBAC authorization database, such as that of the RBAC/Web, the associated permissions for each role are not centrally available. For these systems containment can be expressed in terms of inheritance of user-role relationships. The users authorized for a role include all users authorized for all roles contained by the role.

The inherited user members of a role include all user members of all roles that are contained by the role. For example, if u_1 is explicitly authorized for role A and role B is contained by role A, then u_1 is implicitly authorized for role B and can execute the permissions associated with role B. It is in this sense that the permissions of role B are inherited by role A. As such, RBAC/Web provides services for the management of the authorization of users to roles in terms of user-role relationships, role-role relationships and constraints on user-role relationships, while role-permission relationships are maintained on the back-end web servers. Because the NIST model places no restrictions on assignment of permissions to roles, such assignments with respect to sensitive objects must be under strict administrator control. This is similar to the non-discretionary practice of assigning security levels to classified information in a DoD multi-level secure (MLS) environment.

As illustrated in Figure 3, role hierarchies can be combined to form an directed acyclic graph. A directed acyclic graph (DAG) role representation is a natural way of organizing roles to reflect authority, responsibility, and competency. The more powerful roles (i.e., those role that contain the greatest number of permissions and that are authorized for the fewest users) are found at the top of the graph, and the more general roles of the graph are found at the bottom. Consider the Staff role; the permissions of the Staff role are implicitly associated with all other roles in the enterprise, and the user members of all other roles are implicitly members of the Staff role. As shown in Figure 3, not all roles have to be related. For example, the roles AR Supervisor and Cashier Supervisor are not hierarchically related, and as such are not comparable under the contains relationship, but they may contain some or all of the same roles. It should also be noted that not all roles are intended to have explicit user members, and may only exist for the sole purpose of associating permissions.

Role hierarchies provide a powerful and convenient means of administratively specifying and ensuring adherence to the principle of Least Privilege [5], [8]. Since many of the responsibilities overlap between job categories, maximum privilege for each job category could cause unlawful access. To address least privilege issues, roles representing organizational functions can be associated with only those privileges that need to perform that function, and users can be made members of the roles that are known to be authorized to perform that function. In the cases where privileges completely overlap among roles, hierarchies of roles can be established. For example, it may seem sufficient to allow physicians to have access to all data within a patient record if their access is monitored sufficiently. However, this would entail much more auditing and monitoring than would be necessary with a better defined access control mechanism. With RBAC, constraints can be placed on physician access so that, for example, only those fields that are associated with a particular type of physician can be accessed. Note that multiple inheritance in role hierarchies is allowed, but not required, by the model.

2.4 Static Constraints

RBAC provides administrators with the capability of imposing constraints on user-role and role-permission authorization. From a policy perspective, constraints provide a powerful means of enforcing conflict of interest and cardinality rules for roles as they uniquely apply to an enterprise. The association of a user with a role can be subject to the following constraints:

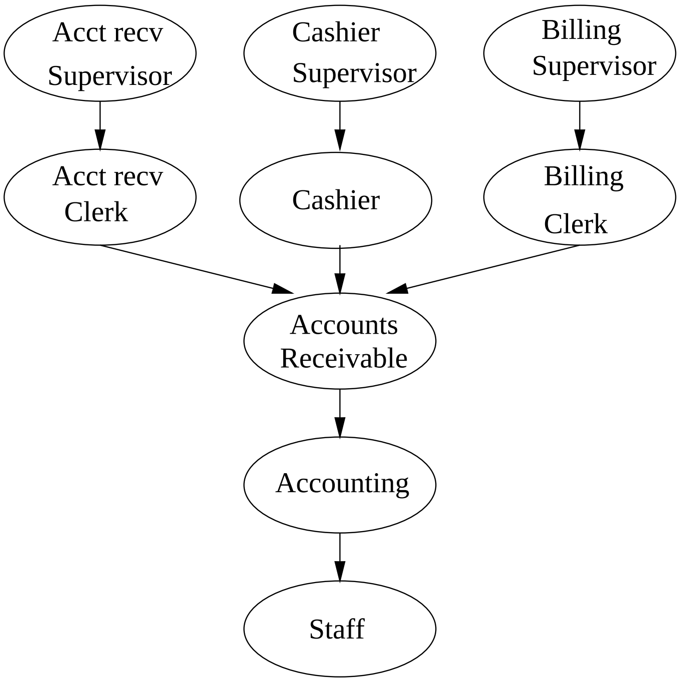

Fig. 3. Example role hierarchy

—the role for which the user is authorized is not mutually exclusive with another role for which the user already possesses membership; and

—the numerical limitation that exists for the number of users authorized for a role cannot be exceeded.

2.4.1 *Static Separation of Duty.* The first constraint, called static separation of duty (SSD), can be used to enforce conflict of interest policies that may arise as a result of concerns for a user gaining authorization for permissions associated with conflicting roles. This means that by virtue of a user being authorized as a member of one role, the user is prohibited from being a member of a second role. For example, a user who is authorized for the role Cashier in the application depicted in Figure 3 may not be authorized for the role Accounts Receivable Clerk. That is, the roles Cashier and Accounts Receivable Clerk are mutually exclusive. The policy of SSD can be centrally specified and can then be uniformly imposed on specific roles. The constraint function for mutually exclusive roles and the Static Separation of Duty property is specified as follows:

$Ea : role \times roles$, the exclusive authorization set, which gives the pairs of roles that are mutually exclusive with each other for role membership.

Static Separation of Duty A user is authorized for a role only if that role is not mutually exclusive with any of the other roles for which the user is already authorized:

$$(\forall u : user)(\forall i, j : roles) u \in RM[i] \wedge u \in RM[j] \Rightarrow (i, j) \notin Ea \tag{2}$$

As described earlier, constraints are inherited within a role hierarchy. For example, in Figure 3, since roles Accounts Receivable Clerk and Billing Clerk have an SSD relationship, then AR Supervisor also has an SSD relationship with Billing Clerk. Another way of thinking of this is that any instance of AR Supervisor can be treated as an instance of Accounts Receivable Clerk. Therefore, the SSD constraint Billing Clerk has with Accounts Receivable Clerk must also apply to AR Supervisor. Rather than including constraint inheritance as a formal property of role hierarchy, constraints are better treated as a side effects of imposing constraints on roles within a hierarchy.

Because a containing role is effectively an instance of its contained roles, no SSD relationship can exist between them. In the previous example, it would not make sense to have an SSD relationship between AR Supervisor and AR Clerk, since by definition there cannot be any conflict of interest. Otherwise, a containment relationship should not have been used to inherit implicit properties that conflict with explicit properties being defined.

2.4.2 *Role Cardinality.* Another type of constraint imposed on the RBAC authorization database is the cardinality of a role. Some roles in an organization may only be occupied by a certain number of employees at any give time. For example consider the role of a department head. Although over time a number of individuals may assume this role, only one individual may assume the responsibilities of the department head at a given point in time. Cardinality constraints could also be used as a means of enforcing licensing agreements. We formally define cardinality by the following rule:

$ML(r : role) \rightarrow \mathcal{Z}$, the number of users (0 or more) that may be authorized for role r.

Cardinality The number of users authorized for a role cannot exceed the role's cardinality:

$$(\forall r : role) \# RM[r] \leq ML[r] \tag{3}$$

Note that cardinality applies to all authorizations, i.e., both explicitly assigned and those resulting from inheritance (see sec. 3.1.1.).

2.5 Static Properties

The model constraints described in previous sections have a number of important implications for properties of the RBAC model [12]. The practical significance of the next result is that a role cannot inherit another role that has been designated as mutually exclusive with it. This is clearly a desirable property, and this result shows that the rules are sufficient to ensure it.

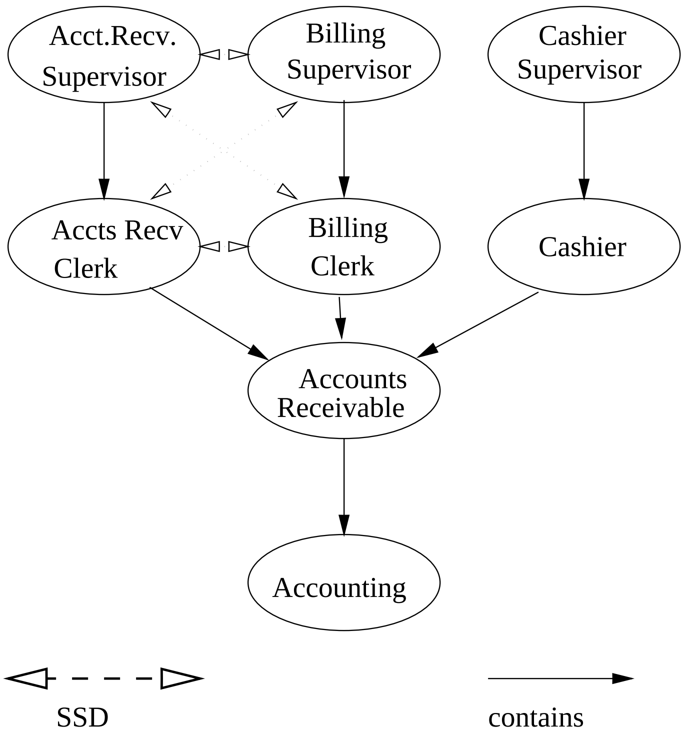

Fig. 4. Example, SSD Consistency for AR Clerk is in SSD with Billing Clerk

THEOREM CONSISTENCY OF SSD AND CONTAINMENT. *Two non-empty roles i and j (i.e., $RM[i] \neq \emptyset$) and $RM[j] \neq \emptyset$ can be mutually exclusive only if they are incomparable within the role hierarchy poset:* $(i, j) \in Ea \Rightarrow \neg(i \succeq j \vee j \succeq i)$.

Proof: Suppose $(i, j) \in Ea \wedge (i \succeq j \vee j \succeq i)$. Arbitrarily choose i as the containing role and let u be authorized for role i. Then by Role Inheritance,

$$u \in RM[j].$$

But by SSD,

$$(i, j) \notin Ea,$$

which contradicts the assumption. Q.E.D.

An immediate corollary is that if there are any mutually exclusive roles, then a non-empty role cannot be mutually exclusive with itself. This might have been required as one of the basic rules, but as it happens, it is a consequence of them.

COROLLARY CONSISTENCY AMONG CONTAINED ROLES. *A non-empty role cannot be mutually exclusive with itself:* $\forall i : (i, i) \notin Ea$

Proof: By the theorem above,

$$i \succeq j \vee j \succeq i \Rightarrow (i, j) \notin Ea.$$

Substituting $j := i$ gives

$$i \succeq i \Rightarrow (i, i) \notin Ea$$

By definition, $i \succeq i$, so for all i,
$(i, i) \notin Ea$. Q.E.D.

If there are any mutually exclusive roles, then those roles cannot have a common upper bound.

THEOREM NON-INHERITANCE OF MUTUALLY EXCLUSIVE ROLES. *If there is any pair $(i, j) \in Ea$, then there can be no non-empty role k such that $k \succeq i \wedge k \succeq j$.*

Proof: Suppose there is some non-empty role k, and mutually exclusive roles i, j such that
$k \succeq i \wedge k \succeq j \wedge (i, j) \in Ea.$
Then because
$k \succeq i \wedge k \succeq j,$
the role inheritance rule requires that
$u \in RM[i] \wedge u \in RM[j],$
so by the SSD rule,
$(i, j) \notin Ea,$
which contradicts the assumption. Q.E.D.

An immediate corollary is that the rules also prohibit the existence of a "superuser" or "root" role that contains all other roles on the system.

COROLLARY NON-EXISTENCE OF SUPERUSER ROLE. *If there is any pair $(i, j) \in Ea$, then there can be no non-empty role r such that for all i, $r \succeq i$.*

Users thus cannot gain access to unauthorized privileges directly through inheritance, but it is also important to show that access cannot be gained indirectly.

THEOREM TRANSITIVITY OF MUTUAL EXCLUSION. *If a user inherits a role that is mutually exclusive with another role, then the user cannot be authorized for the second role. That is, $u \in RM[i] \wedge i \succeq j \wedge (j, k) \in Ea \Rightarrow u \notin RM[k]$.*

Proof: Suppose $u \in RM[i] \wedge i \succeq j \wedge (j, k) \in Ea \wedge u \in RM[k]$.
Then by the Inheritance rule,
$u \in RM[j],$
But by SSD,
$u \in RM[j] \wedge u \in RM[k] \Rightarrow (j, k) \notin Ea,$
which contradicts the assumption. Q.E.D.

Mutual exclusion between roles and role cardinality are examples of static constraints, i.e., restrictions on user/role authorization. Inheritance rules apply to arbitrary static constraints $\mathcal{C}(RM[i])$ and $\mathcal{C}(RM[j])$ on users in roles i and j. In contrast to privileges, which can be thought of as inherited "downward," in general, static constraints are inherited "upward":

THEOREM INHERITANCE OF STATIC CONSTRAINTS. *For roles i, j, if $i \succeq j$ and $\mathcal{C}(RM[j])$, where $\mathcal{C}$ is some constraint on users authorized for role j, then $\mathcal{C}(RM[i])$*

Proof: If $i \succeq j$ and $\mathcal{C}(RM[j])$
then by Role Inheritance,

$$u \in RM[i] \Rightarrow u \in RM[j].$$

So by definition of $\in$,

$$RM[i] \subseteq RM[j],$$

so $\mathcal{C}(RM[i])$ holds. Q.E.D.

As stated earlier, the complexity of dealing with consistency issues is handled by the RBAC software through the implementation of a series of integrity checks that are derived from the RBAC model.

RBAC maintains the integrity of the RBAC authorization database by checking and enforcing the consistency rules described above. For example, administratively imposing Accounts Receivable Clerk in SSD with Billing Clerk, within the hierarchies illustrated in Figure 3, results in the SSD relationships illustrated in Figure 4. That is, through Inheritance of Static Constraints (2.5), the following relationships are guaranteed: AR Supervisor is in SSD with Billing Clerk; Billing Supervisor is in SSD with AR Clerk; and AR Supervisor is in SSD with Billing Supervisor.

2.6 MC_1:RBAC Activation Model

Dynamic properties complement static properties and refer to properties of the model that involve either subjects or mappings from subjects to other basic elements (i.e., subject to user and subject to active-roles). Dynamic properties include role activation, permission execution, dynamic separation of duties, and object access. Dynamic properties provide extended support for the principle of least privilege in that each user has different levels of permission at different times, depending on the role being performed. These properties ensure that permissions do not persist beyond the time that they are required for performance of duty. This aspect of least privilege is often referred to as *timely revocation of trust.* Revocation of privileges can be a rather complex issue without the facilities of dynamic separation of duty, and as such it has been ignored in the past for reasons of expediency.

In applying dynamic security policy to a computer system we speak of subjects, which are active entities whose access to roles, operations, and objects must be controlled. All requests by a user are carried out by subjects acting on the user's behalf. Each subject is uniquely referenced by an identifier, which is used to determine whether the subject is authorized for a role and can become active in the role. Role inheritance for the active role set can be handled in a number of ways. The model leaves open the question of whether roles are inherited automatically into the active role set. See Section 3.3 for the explanation of how RBAC/Web addresses this question. Subjects are defined by the following type:

type Subject of subject identifiers.

The following functions formalize mappings for users, subjects, and roles:

$SU(s : subject) \rightarrow user$, the subject to user mapping, which gives the user associated with subject s.

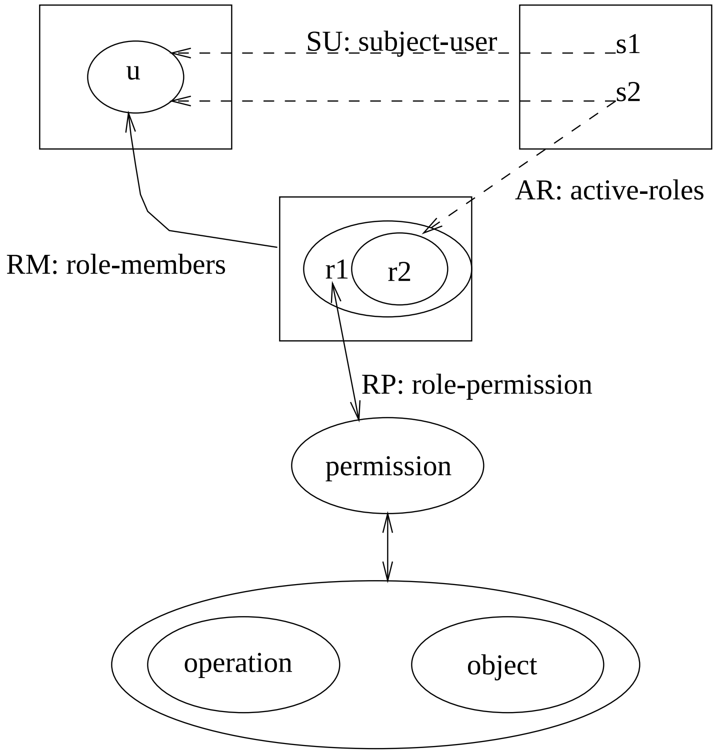

Fig. 5. RBAC Functional Mappings

$AR(s: subject) \rightarrow 2^{role}$, the active role mapping, which gives the set of roles in which subject, s, is active.

Role Activation A subject cannot have an active role that is not authorized for its associated user:

$$(\forall s: subect, u: user, r: roles) r \in AR[s] \Rightarrow SU[s] \in RM[r] \tag{4}$$

RBAC is described in terms of a series of mapping functions on the basic types. By considering the subject-user and active-role mapping functions of the model component M_1 along with the mapping functions of the model component M_0 we can now express RBAC in terms of these relationships. These mappings are illustrated in Figure 5 and are used along with a series of constraints on these mappings to express the properties of the model. The solid lines represent static mapping functions among RBAC elements, and the dotted lines represent dynamic mapping functions among RBAC elements.

2.7 Dynamic Constraints

RBAC provides administrators with the capability to enforce an organization specific policy of dynamic separation of duty (DSD). SSD provides an organization with the capability to address potential conflict-of-interest issues at the time a user's membership is authorized for a role. With DSD it is permissible for a user to be authorized as a member of a set of roles which do not constitute a conflict of interest when acted in independently, but produce policy concerns when allowed to be acted in simultaneously. For example, a user may be authorized for both the roles of Cashier and Cashier Supervisor, where the supervisor is allowed to acknowledge corrections to a Cashier's open cash drawer. If the individual acting in the role Cashier attempted to switch to the role Cashier Supervisor, RBAC would require the user to drop his or her Cashier role, and thereby force the closure of the cash drawer before assuming the role Cashier Supervisor. As long as the same user is not allowed to assume both of these roles at the same time, a conflict of interest situation will not arise. Although this effect could be achieved through the establishment of an SSD relationship, DSD relationships generally provide the enterprise with greater operational flexibility.

The DSD constraint is formally defined using the following constraint function and property:

$Er : role \times role =$ the set of role pairs (i, j) that are mutually exclusive with each other at activation, i.e., no user may be active in both i and j simultaneously

Dynamic Separation of Duty (DSD) A pair of roles may be designated as mutually exclusive of one another with regard to role activation. That is, a user may be active in only one of the two distinct roles so designated:

$$(\forall s, t : subject)(\forall i, j : role) : s \neq t :$$
$$i \in AR[s] \wedge j \in AR[t] \wedge (i, j) \in Er \Rightarrow SU[s] \neq SU[t] \quad (5)$$

Note that in the RBAC/Web implementation, dynamic separation of duties is only relative to a user on a single web server (see sec. 3.1.2).

2.7.1 *Operation Authorization.* Note that unlike roles in an SSD relation, roles in a DSD relation can be hierarchically related through the containment relation. This is consistent with the DSD property of restricting simultaneous activation of roles and that of a role hierarchy as a representation of a user's implicit and explicit authorizations for a role. As such, authorization and activation can be treated as independent notions.

In earlier models, authorization and activation were linked, so that if a subject selected a particular role r_i, then any junior roles r_j such that $r_i \succeq r_j$ would be brought into the active role set as well. Under this approach, if there were any roles r_j with a DSD relationship with r_i, then r_i could not be activated because doing so would require the conflicting roles to be simultaneously active [12] (see sec. 2.5). The model described in this paper does not dictate a specific way to handle role inheritance for active roles. An approach to addressing this problem by means of inheritance and activation hierarchies has been described in

[13]. An alternative, used in the RBAC/Web implementation for Unix, described in Section 3.3, is to forbid roles which are related hierarchically from having a DSD relationship. This prohibition is enforced when role relationships are added or when DSD relation Er is defined.

$$exec : subject \times operation \rightarrow boolean$$

$$exec(s, op) = \begin{cases} 1 & \text{if subject } s \text{ can execute operation } op \\ 0 & \text{otherwise} \end{cases}$$

Operation Authorization A subject can execute an operation only if the operation is authorized for the role in which the subject is currently active.

$$(\forall s : subject)(\forall op : operation) \tag{6}$$
$$exec(s, op) \Rightarrow (\exists r : role)(\exists p : permission) r \in AR[s] \wedge p \in RP[r] \wedge op \in POp[p]$$

2.7.2 *Object Access* . To ensure enforcement of enterprise policies for RBAC objects, subject access to RBAC objects must be controlled. The following rule determines if a subject can access an RBAC object:

$$access : subject \times operation \times object \rightarrow boolean$$

$$access(s, op, o) = \begin{cases} 1 & \text{if subject } s \text{ can access object } o \text{ using operation } op \\ 0 & \text{otherwise} \end{cases}$$

With the Role Activation property defined above, the Object Access Authorization property defined below ensures that a subject's access to an RBAC object can only be achieved through authorized operations by authorized active roles.

Object Access Authorization A subject can perform an operation on an object only if there exists a role that is an element of the subject's active role set and the role contains a permission that authorizes the operation on the object:

$$(\forall s : subject)(\forall o : object)(\forall op : operation) \tag{7}$$
$$access(s, p, op) \Rightarrow$$
$$(\exists r : role)(\exists p : permission) r \in AR[s] \wedge p \in RP[r] \wedge op \in POp[p] \wedge o \in POb[p]$$

3. RBAC/WEB IMPLEMENTATION

3.1 Design Issues

The NIST RBAC Model provides the opportunity for the implementor, based on requirements, to choose between several alternative implementation approaches. Implementation decisions must be made in the design of the administration tools and in the process used to activate roles.

3.1.1 *Administration.* While RBAC can be treated as either as discretionary or non-discretionary access control method, the treatment given in this paper is the latter. That is, one or more administration roles are required that are distinct from user roles, insofar as their permissions deal solely with the policy attribute elements of the model: User-to-Role and Role-to-Permission mappings, containment relations, cardinality constraints, and separation of duty constraints. Users not assigned to administration roles are denied these permissions and must operate within the confines of the roles defined for them and assigned to them by an administrator.

The administrative model for RBAC/Web follows this approach. There is a single role designated for administration. In the context of the implementation, this role is required in order to access administrative tools and consequently, the RBAC Database. The administrative model for RBAC/Web is minimal. An example of a richer, more flexible model, can be found in Sandhu[14].

Another design decision for administration in RBAC/Web is how to manage role authorization when there are role hierarchies. In the NIST RBAC Model, if a user is authorized for a role r_0 and r_0 inherits r_1, then that user is also authorized for r_1. What should happen when authorization for r_0 is removed? Should the user retain the authorization for r_1 or should the authorization for r_1 be removed as well?

In RBAC/Web, the design decision was made to also remove the authorization for r_1. The goal is to make administration in RBAC/Web as easy as possible while making it difficult for an administrator to make a serious mistake. To accomplish this, the decision was made to keep the number of operations involved in role authorizations to a minimum. Since authorizing a role automatically authorizes the roles it inherits (this is the definition of role inheritance), it is consistent to remove authorizations for inherited roles when authorization for a parent is removed. If the administrator wishes these inherited roles to be authorized for the user, those operations must be done explicitly. With this approach, administrative errors result in roles being unintentionally left unauthorized. The alternative approach, i.e., leaving inherited roles authorized, could result in roles being unintentionally left authorized.

Removing authorization for inherited roles when authorization for a parent is removed also results in constantly reminding the administrator of the organization's role hierarchy. To further support this approach, the RBAC/Web Admin Tool differentiates between the concept of role *assignment* and role *authorization* as defined in the NIST RBAC Model. A role is assigned to a user explicitly by means of the Admin Tool. A role is authorized as defined in the NIST RBAC Model if that role is assigned to the user or is inherited by a role which is assigned to the user. From this, it follows that if an administrator attempts to assign a role to a user who, by virtue of inheritance from an assigned role, is already authorized for that role, then that attempt is deemed an error.

3.1.2 *Role Activation.* In the NIST RBAC Model, there is a function, $AR(s : subject) \rightarrow 2^{role}$, which maps subjects that identify a user, e.g., a login name, to active role sets (ARS) (see sec. 2.6). The subject represents a user within the implementation environment. There is also a function in the NIST Model,

$SU(s : subject) \rightarrow user$, which maps subjects to users, i.e., given a subject, the function SU returns the user whom that subject represents. An implementation of the NIST RBAC Model must define the concept of *subject* within the context of the implementation environment, the function SU, and the function AR.

Within the environment of a single computer system, e.g., a Unix system, a subject could be defined as the process ID of a user process. The user process represents the user and performs operations in the user's name. In the World Wide Web environment, each access, e.g., click on a hyperlink, is analogous to the creation of a user process on the Web server to perform the access. Once the access completes, this user process is removed.

Because of the short lifetime of user processes within the Web server environment, defining the process ID of these processes to be the user's subject implies that an active role set must be established upon each access request. Since this is inefficient and unnecessary, RBAC/Web defines the user's subject as the user's login name on the Web server. On a Web server, each user has only one login name. Thus, there is only one subject per user, i.e., the function SU consists of ordered pairs of the form "$(login\_name, user)$," one for each Web server user. Consequently, there is only one active role set per user per Web server, i.e., the function AR consists of ordered pairs of the form "$(login\_name, ARS)$," one for each Web server user. There is only only one active role set per user per Web server regardless of the location and number of browser windows that may be open to the Web server by the user.

RBAC/Web implements this approach by maintaining the user's ARS in a file on the Web server. This file has the user's login name, i.e., the user's subject, as part of the file name, thus, making it possible to locate a user's ARS given the user's login name. Each time a user attempts to access a URL, RBAC/Web determines whether the user has access by consulting this file. The RBAC/Web Session Manager, a common gateway interface (CGI) script which is part of RBAC/Web, manages the contents of a user's ARS. In order to simplify the process of determining the contents of a user's ARS, when a user has authorized roles which have DSD relationships, the user is presented with a selection of the largest subsets of the user's set of authorized roles and asked to choose. For the scenario describing how users access a Web server enhanced with RBAC/Web, see section 3.6.2.

The DSD relationship of the NIST RBAC Model places a constraint on active role set contents, i.e, any pair of roles in the active role set cannot have a DSD relationship. Given this constraint, the question arises as to how to determine active role set contents given role pairs which have both a hierarchical and a DSD relationship.

Consider the following example: suppose role r_1 and role r_2 have a DSD relationship, r_1 is authorized for user U, and r_1 inherits r_2. When establishing U's active role set, the following apparent contradiction results. Role r_2 belongs in U's active role set because r_1 inherits r_2 but r_2 cannot be in U's active role set because r_1 and role r_2 have a DSD relationship. There are at least two possibilities for addressing this issue in an implementation:

—the inheritance relationship between a pair of roles in DSD is overridden and one role or the other (but not both) is placed in the active role set; or,

—the situation, i.e., a pair of roles related both hierarchically and having a DSD relationship, is never allowed in the RBAC Database.

The design decision made by RBAC/Web for Unix is to implement the latter. In RBAC/Web for Unix, the Admin Tool does not permit a role pair to simultaneously have both a hierarchical and a DSD relationship. Thus, the apparent contradiction in active role set contents can never occur. This design decision is based on the desire to have all role relationships specified in the RBAC Database hold at all times and in all situations. The goal is to simplify the task of administration. Administrators are not required to be aware of situation-sensitive rules. They are able to know that what is in the RBAC Database holds throughout the administration, role activation, and enforcement of role relationships and access.

Alternative approaches are equally valid depending on implementation requirements. One such alternative approach is described by Sandhu[13]. Th Sandhu paper describes the distinction between the *usage* and *activation* aspects of role hierarchies. In terms of these concepts, RBAC/Web's implementation of role hierarchies combines both aspects into one hierarchy model.

3.2 Components

RBAC for the World Wide Web (RBAC/Web) is an implementation of the NIST RBAC Model for use by World Wide Web (WEB) servers on both the internet and intranets. RBAC/Web can be used in conjunction with existing WWW authentication and confidentiality services. These include username/password and Secure Socket Library (SSL). User identification information is passed to RBAC/Web by the Web server. It is the responsibility of the Web server to authenticate user identification information and provide confidential data transmission as configured by the Web server administrator.

RBAC/Web places no requirements on a browser. Any browser that can be used with a particular Web server can be used with that server enhanced with RBAC/Web. RBAC/Web is implemented in two environments: UNIX (e.g., Netscape, Apache servers) and Windows NT (e.g., Internet Information Server, Website, or Purveyor).

Database	Files that specify the relationship between users and roles, the role hierarchy, the constraints on user/role relationships, current active roles, and relationship between roles and operations.
API Library	A specification which may be used by Web servers and CGIs to access the RBAC/Web Database. The application program interface (API) is the means by which RBAC may be added to any Web server implementation. The API Library is a C and Perl library which implements the RBAC/Web API.
CGI	Implements RBAC as a CGI for use with any currently existing Web server without having to modify the server. The RBAC/Web CGI uses the RBAC/Web API.
Session Manager	The RBAC/Web Session Manager activates roles by establishing a user's current active role set (ARS).
Admin Tool	Allows server administrators to create users, roles, and permitted operations; associate users with roles and roles with permitted operations; specify constraints on user/role relationships; and maintain the RBAC Database. Administrators access the RBAC/Web Admin Tool by means of a Web browser.

Table 1. RBAC/Web Components

3.3 RBAC/Web for Unix

RBAC/Web for Unix implements MC_0 and MC_1, i.e., the role properties: hierarchy, cardinality, SSD, and DSD. Components of RBAC/Web are shown in Table 1. RBAC/Web for UNIX uses all of the components in Table 1.

With RBAC/Web for UNIX, there are two ways to use RBAC/Web with a UNIX Web server. The simplest way is by means of the RBAC/Web CGI. The RBAC/Web CGI can be used with any existing UNIX server without modifying its source code. RBAC URLs are passed through the Web server and processed by the RBAC/Web CGI. RBAC/Web configuration files map URLs to file names, while providing access control based on the user's roles. Installation of the RBAC/Web CGI is similar to the installation of the Web server.

While RBAC/Web CGI is relatively simple to install and use, it is not as efficient as performing access control directly in the Web server. The other way to use RBAC/Web is to modify the UNIX Web server to call the RBAC/Web API to determine RBAC access. A URL is configured as an RBAC controlled URL by means of the Web server configuration files that map URLs to file names.

Some Web servers for a UNIX environment, such as Netscape and Apache, divide their operation into steps and provide the capability for each step to be enhanced or replaced by means of a configuration parameter. This allows Web server operation to be modified without having to change the server's source code. For these Web servers, the RBAC/Web API can be integrated by simply providing the appropriate calling sequence and modifying configuration parameters.

3.4 RBAC/Web for Windows NT

RBAC/Web for Windows NT only implements MC_0, i.e., the static properties: hierarchy, cardinality, and SSD. In order to implement DSD in a Windows NT environment, it is necessary to change the session establishment mechanisms of Windows NT. Such a difficult task was beyond the scope of the project. The task is particularly difficult in that neither documentation nor source code for Windows NT session establishment is public information. Because RBAC/Web for Windows NT only implements MC_0 and because built-in NT security mechanisms are closely compatible with RBAC, RBAC/Web for Windows NT needs only the Database and Administrative Tool components shown in table 1. RBAC/Web for Windows NT requires no modification of Web server internals or access to source code.

Because Windows NT security mechanisms are uniform between Windows NT itself and many Web servers which run on Windows NT, RBAC/Web for Windows NT can also manage access to the Windows NT File System as well as Web resources. In addition to the Admin Tool which manages user/role and role/role relationships, the tool RGP-Admin[15] was implemented to manage role/permission relationships in the Windows NT environment.

3.5 RBAC Database

The RBAC Database is an implementation of the RBAC model component MC_1 in the case of RBAC/Web for Unix and MC_0 in the case of RBAC/Web for Windows NT. The RBAC database elements are created through the use of the administrative tool that graphically displays and maintains these relationships. In addition the RBAC administrative tool maintains the integrity of the RBAC Database by checking and enforcing consistency rules[10].

An operation as defined in sec. 2.2 represents a particular method of access to a set of one or more protected RBAC objects. When authorizing user membership into a role, the user is implicitly provided with the potential to exercise the operations that are associated with the role. Permissions in RBAC/Web are the HTTP methods that the end-user can perform on RBAC controlled URLs.

In general, constraints provide confidence as to the adherence of enterprise wide policies. In theory, similar effects can be achieved through the establishment of procedures and the sedulous actions of administrators. For example, administrators can maintain and share a list of role pairs that are known to be mutually exclusive and ensure that an individual user is never authorized for both roles. However, the reality is that procedures break down and administrators get reassigned over time. The constraints imposed by RBAC/Web provide management and regulators with the confidence that critical security policies are uniformly and consistently enforced within the network, and as such, contribute to the network's operational assurance.

Associated with objects managed by RBAC/Web are the ACLs that reside with each Web server. With RBAC/Web for Unix, an ACL is organized as a list of roles, where for each role, there is a list of HTTP methods under which a user acting in the role is permitted to access an associated URL. RBAC/Web for NT makes use of the ACLs built into Windows NT where a role maps to a Windows NT Domain or Local Group. The collection of ACLs is organized and managed as the collection of role-privilege relationships.

In the context of RBAC/Web, each subject represents a user active in one or possibly many roles. A user establishes an active role set, i.e., a subset of the roles for which the user has membership is activated. A user's authorization (which is a consequence of role membership) is a necessary but not always sufficient condition for a user to be permitted to execute a privilege. Other organizational policy considerations or constraints may need to be taken into consideration that pertain to authorizing users to execute permissions.

RBAC/Web requires a user to first be authorized as active in a role before a user is permitted to perform an operation or access a URL. This provides the context for other policy checks to be imposed. As such, in the case of RBAC/Web for Unix, administrators are provided with the capability to enforce an organization-specific policy of Dynamic Separation of Duty (DSD).

3.6 Scenario of Use

There are two use scenarios. One describes how an administrator manages access policy. The other describes end-user interaction.

3.6.1 *Use by Administrators.* The RBAC/Web Admin Tool manages user/role and role/role associations. The RBAC/Web Admin Tool manages the specification of: user/role assignments, role hierarchies, Static Separation of Duty constraints, Dynamic Separation of Duty constraints (in the case of RBAC/Web for Unix), and Role Cardinality.

In order to reduce errors in administration, the Admin Tool differentiates between the concept of role *assignment* and role *authorization* as defined in the NIST RBAC Model. A role is assigned to a user explicitly by means of the Admin Tool. A role is authorized as defined in the NIST RBAC Model if that role is assigned to the user or is inherited by a role which is assigned to the user. This concept of role assignment helps an administrator maintain awareness of the role hierarchies which describe an organization.

The Admin Tool manages the RBAC Database by means of a fixed set of operations that can be performed on the Database. A formal description of these operations can be found in [10].

RBAC/Web is used to manage access to the Admin Tool itself. There is a single special administrative role which must be authorized for a user to access the Admin Tool and RBAC Database.

The example access control policy for an accounting department shown in figure 4 is illustrated in the Admin Tool screen captures of figures 6 and 7. The Role Hierarchy display of Figure 6 presents a graphical view of the sample accounting department policy. In particular, it shows the accounting department's role hierarchy. Figure 6 and the role relationships of the selected role AR_Supervisor (shaded) show that the role AR_Supervisor has a SSD relationship with the roles Billing_Supervisor and Billing_Clerk. These SSD relationships are indicated by enclosing the roles Billing_Supervisor and Billing_Clerk in pentagons instead of ovals. The parenthesized expressions, (n, m), under each role name in figure 6 indicates the Cardinality of each role (m) and for how many users the role has been authorized (n). The "U" indicates unlimited cardinality, i.e., there is no limit on the number of users which may be authorized for the role.

The left frame of figure 7 shows the display for managing user/role assignments. The frame shows that user Smith has been assigned the role AR_Supervisor. The frame also shows that, in concert with the role/role associations defined by accounting department policy as shown in figure 6, Smith could also be assigned

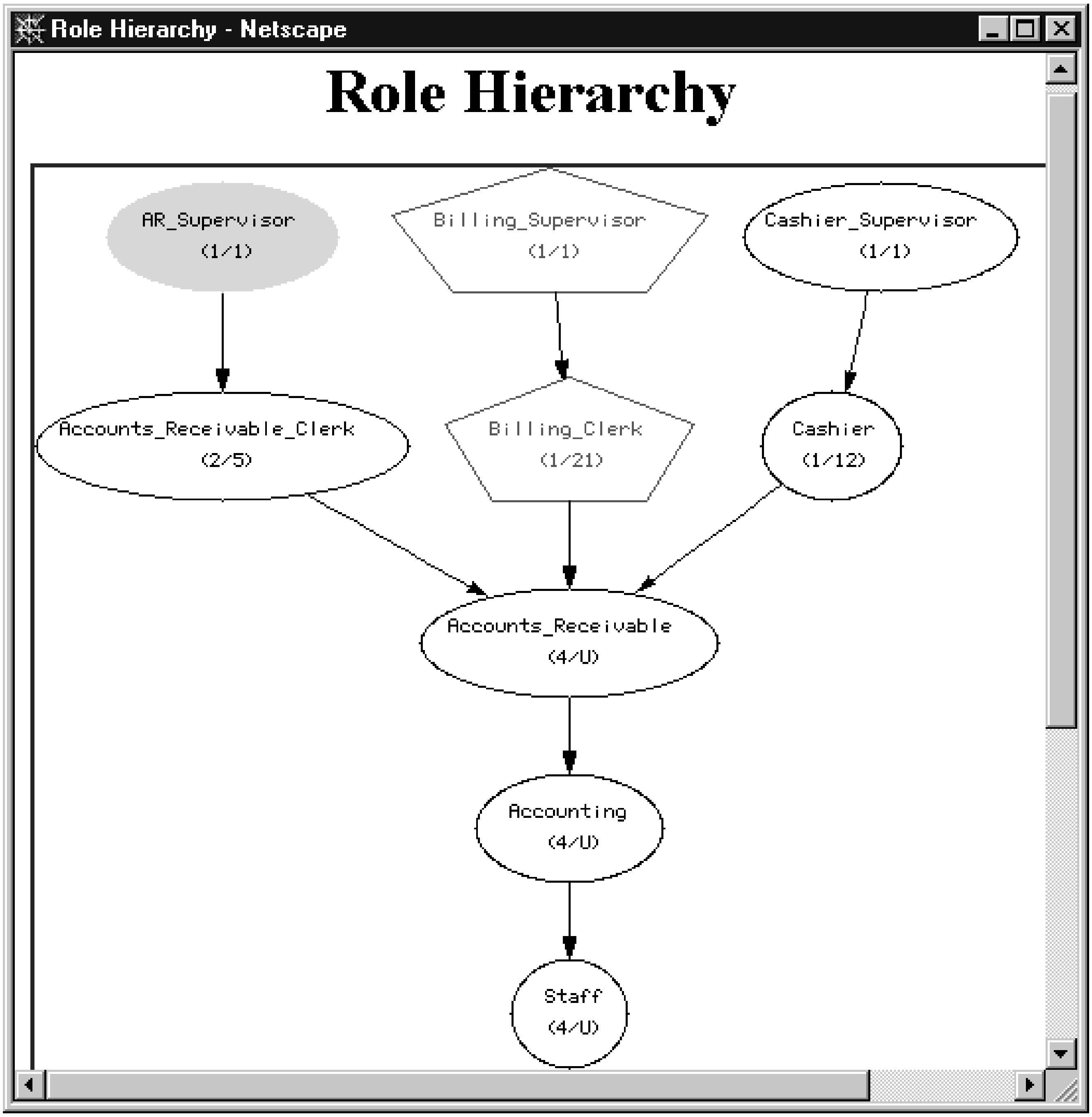

Fig. 6. RBAC/Web Admin Tool: Role Hierarchy Display.

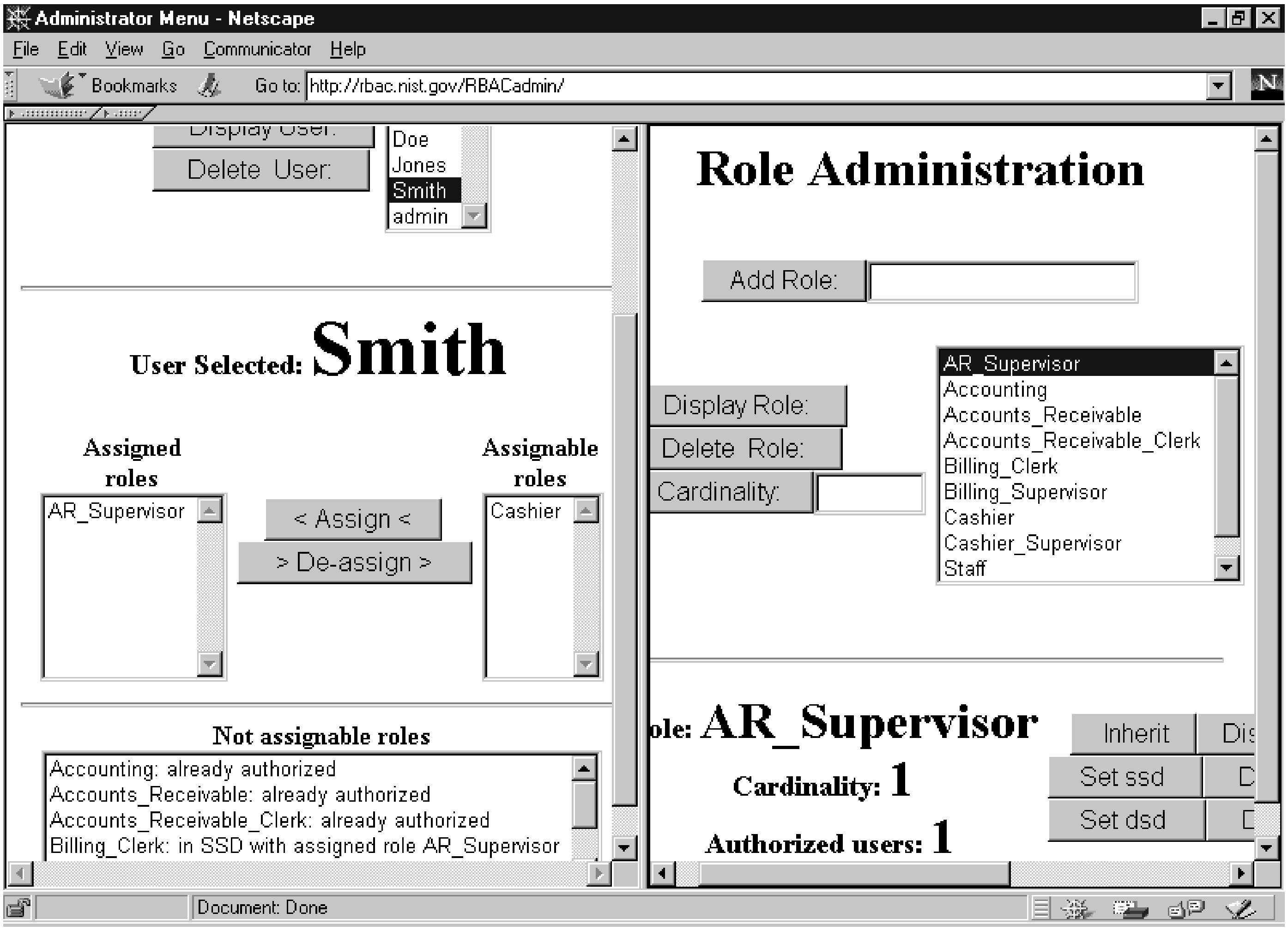

Fig. 7. RBAC/Web Admin Tool: Display for managing user/role and role/role relationships.

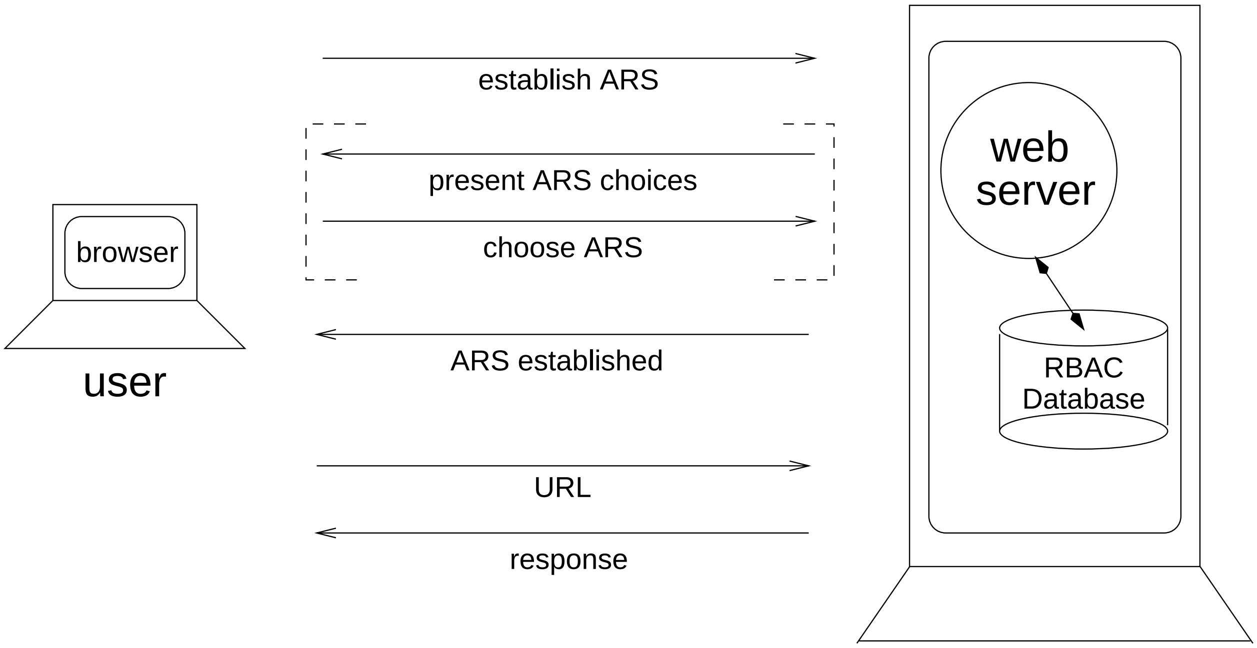

Fig. 8. RBAC/Web End-user Perspective.

the role Cashier. However, Smith could not be assigned the roles Accounting, Accounts_Receivable, or Accounts_Receivable_Clerk because these roles are inherited by the role AR_Supervisor; nor could Smith be assigned the role Billing_Clerk or Billing_Supervisor because these roles have an SSD relationship with the role AR_Supervisor. These role/role associations can be seen using the graphical display shown in figure 6.

The right frame of figure 7 shows the display for managing role/role associations. It presents buttons for establishing and removing all role/role associations. The frame shows that the role AR_Supervisor has been selected. Since the role AR_Supervisor was selected in the right frame of figure 7, the Role Hierarchy display of figure 6 also shows the role AR_Supervisor selected.

The Admin Tool enforces a set of consistency requirements on the RBAC/Web Database. These consistency requirements, described in [10], ensure that user/role and role/role relationships have the required relationships to each other. For example, as described in section 3, RBAC/Web for Unix requires that any roles which have a DSD relationship cannot have a hierarchical relationship.

3.6.2 *Use by End-users.* From the end-user's perspective, because DSD is not supported, interaction with a Web server enhanced with RBAC/Web for Windows NT is exactly the same as interacting with any other Windows NT Web server. However, with RBAC/Web for Unix, before access to a URL controlled by RBAC is permitted, end-users must establish an active role set (ARS) as shown in Figure 8. End-users choose and/or are assigned a current active role set. The ARS determines the HTTP methods that the end-user can perform on RBAC controlled

URLs. The ARS remains in effect until the end-user establishes a new ARS. It is the RBAC/Web Session Manager which allows users to change their ARSs.

A user may be assigned roles which have DSD relationships. If this is the case, the Session Manager enables users to choose the subset of their assigned role set that they would like to have in their ARS. Users are presented with a list of subsets which do not violate any DSD relationships and asked to choose. In order to minimize the number of choices, the subsets in the list, taken from the set of all possible subsets of a user's assigned roles, contains the largest subsets which do not violate any DSD relationships. Once the choice is made, an ARS is established consisting of all assigned roles in the chosen subset and all roles which the assigned roles inherit. If there are no DSD relationships among the roles assigned to a user, then the ARS is automatically established consisting of all authorized roles. Note that because the RBAC/Web implementation applies only to a single server, dynamic separation of duties is not implemented relative to users across multiple servers, (e.g., in an environment where a collection of servers constitutes an administrative security domain) but only relative to a user on a single web server. See section 3.1.2 for information on how RBAC/Web implements a user's ARS on a Web server.

4. CONCLUSIONS

Although intranets can offer great benefits to a company or government agency, security problems remain. For intranets to reach their full potential as a means for enterprise computing, access control mechanisms must be in place that can conveniently, and cost effectively regulate user access to information, while providing management with confidence that their critical policies are faithfully and consistently enforced across administrative boundaries. To solve these and other authorization problems, NIST has initiated an effort to provide and promote the use of Role-Based Access Control (RBAC) for intranet web servers. RBAC is particularly attractive for intranet applications because of its ability to reduce the complexity and cost of authorization management. In addition, RBAC provides a context for the specification and enforcement of complex security policies that are often impractical or even impossible to enforce through the direct use of conventional access control mechanisms.

RBAC is administered through the use of roles and role hierarchies that mirror an enterprise's job positions and organizational structure. Users are assigned membership into roles in a manner that is consistent with a user's duties, competency and responsibility. Constraints are imposed on user membership into roles and on a user's ability to activate a role to address conflict of interest issues. Complexities that are introduced by simultaneously supporting mutually exclusive roles and role hierarchies are handled by the RBAC software, making security administration easier. It is the roles, role hierarchies, and constraints that provide the context by which the intranet administrators can specify, and RBAC/Web servers can enforce, the specifics of a large variety of laws, regulations and business practices that can pertain to an organization.

RBAC has been shown to support several well-known security principles and policies that are important to commercial and government enterprises that process unclassified but sensitive information [7], [16]. These include: the specification of competency to perform specific tasks; the enforcement of Least Privilege for administrators and general users; and the specification, as well as the enforcement, of conflicts of interest rules, which may entail duty assignment and dynamic and static separation of duties. For RBAC/Web these policies can be enforced at the time that users are authorized as members of a role, at the time of role activation (e.g., when a role is established as part of a user's active session), or at the time the user attempts to access a URL.

Under RBAC, intranet administrators are provided with a single view of the RBAC authorization database which is at a level of abstraction that is intuitive and consistent with the way the enterprise is structured and conducts business. RBAC/Web thereby bridges the huge gap between the enterprise's laws, regulations, and business practices and the details of the underlying access control mechanisms of the Web servers.

REFERENCES

[1] D. Ferraiolo and D.R. Kuhn. Role based access control. In *15th National Computer Security Conference.* NIST/NSA, 1992.

[2] D. Ferraiolo, J. Cugini, and D.R. Kuhn. Role based access control: Features and motivations. In *Annual Computer Security Applications Conference.* IEEE Computer Society Press, 1995.

[3] W.A. Jansen. Revised model for role based access control; NIST-IR 6192. Technical report, NIST, January 1988.

[4] J. Hoffman. Implementing RBAC on type enforced systems. In *Proceedings, 13th Annual Computer security applications conference.* IEEE Computer Society Press, 1997.

[5] R. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role based access control models. *IEEE Computer*, 29(2), February 1996.

[6] M. Nyanchama and S.L. Osborn. Access rights administration in role-based security systems. *Proceedings of the IFIP WG11.3 working conference on database security*, 1994.

[7] D.F. Ferraiolo, D.M. Gilbert, and N. Lynch. An examination of federal and commercial access control policy needs. In *16th National Computer Security Conference.* NIST/NSA, 1993.

[8] H.L. Feinstein. Final report: NIST small business innovative research (SBIR) grant: role based access control: phase 1. Technical report, SETA Corp., January 20 1995.

[9] R. Sandhu and Q. Munawer. How to do discretionary access using roles. In *Proceedings of the Third ACM Workshop on Role Based Access Control*, pages 47–54, 1996.

[10] Serban Gavrila and John Barkley. Formal specification for role based access control user/role and role/role relationship management. In *Proceedings of the Third ACM Workshop on Role-Based Access Control.* ACM, October 1998.

[11] R. Simon and M.E. Zurko. Separation of duty in role based access control environments. In *Proceedings, New Security Paradigms Workshop*, September 1997.

[12] D.R. Kuhn. Mutual exclusion as a means of implementing separation of duty requirements in role based access control systems. In *Proceedings Second ACM Workshop on Role based access control*, 1997.

[13] Ravi Sandhu. Role activation hierarchies. In *Proceedings of the Third ACM Workshop on Role-Based Access Control.* ACM, October 1998.

[14] Ravi Sandhu, Venkata Bhamidipati, Edward Coyne, Srinivas Ganta, and Charles Youman. The ARBAC97 model for role-based administration of roles: Preliminary description and model. In *Proceedings of the Second ACM Workshop on Role-Based Access Control.* ACM, November 1997.

[15] John Barkley and Anthony Cincotta. Managing role/permission relationships using object access types. In *Proceedings of the Third ACM Workshop on Role-Based Access Control.* ACM, October 1998.

[16] S.H. von Solms and I. van der Merve. The management of computer security profiles using a role oriented approach. *Computers and Security*, 13(8), 1994.