The Role Graph Model and Conflict of Interest

MATUNDA NYANCHAMA and SYLVIA OSBORN
The University of Western Ontario

We describe in more detail than before the reference model for role-based access control introduced by Nyanchama and Osborn, and the role-graph model with its accompanying algorithms, which is one way of implementing role-role relationships. An alternative role insertion algorithm is added, and it is shown how the role creation policies of Fernandez et al. correspond to role addition algorithms in our model. We then use our reference model to provide a taxonomy for kinds of conflict. We then go on to consider in some detail privilege-privilege and role-role conflicts in conjunction with the role graph model. We show how role-role conflicts lead to a partitioning of the role graph into nonconflicting collections that can together be safely authorized to a given user. Finally, in an appendix, we present the role graph algorithms with additional logic to disallow roles that contain conflicting privileges.

Categories and Subject Descriptors: D.4.6 [**Operating Systems**]: Security and Protection—*access controls*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms*

General Terms: Algorithms, Management, Security

Additional Key Words and Phrases: role-based security, role graphs, conflict of interest

1. INTRODUCTION

Role-based access control provides a way of managing authorizations to perform tasks in complex systems with many users and many resources [Sandhu et al. 1996]. Roles are used to group permissions together in ways that make sense in the enterprise or the application environment. Individual users or groups of people can then be assigned to the roles as required.

Roles provide a very natural and powerful way for an enterprise administrator or security officer to describe the privileges of various job functions. This paper describes our reference model and how it fits into a system's authorization scheme. In our previous work, we also introduced a role

Authors' addresses: M. Nyanchama, Ernst & Young Tower, 90 Burnamthorpe Road West, Suite 1100, The University of Western Ontario, Mississauqa, ON L5B-3C3, Canada; email: matunda.nyanchama@ca.eyi.com; S. Osborn, Department of Computer Science, The University of Western Ontario, London, ON N6A 5B7, Canada; email: sylvia@csd.uwo.ca.

graph model, which provides a way of visualizing the interactions among roles and their seniors and juniors [Nyanchama and Osborn 1994]. We also introduced algorithms for manipulating these role graphs. Role-based models provide a way to better manage access rights in a system, and can be applied in discretionary access control and also be used to simulate a mandatory access control environment [Nyanchama and Osborn 1995; Osborn 1997; Sandhu 1996].

In this paper we briefly review our role graph algorithms [Nyanchama and Osborn 1994], and enhance them with some additional algorithms. These algorithms have all been implemented in an interactive tool, which allows roles to be created, deleted, and altered as described below. Since the algorithms deal with acyclic directed graphs, they all have efficient runtime complexity. One version of our role-graph system provides an interface with a relational database [Osborn et al. 1996]. Through this tool, it is possible to quickly alter the roles and the users assigned to the roles and have changes conveyed back to the database system.

An additional and important security task in a commercial environment is to define and deal with conflict of interest. Aside from our reference model, a key contribution of this paper is to consider conflict of interest within the model (which indicates five possible types of conflict) and show how our role graph model can be augmented to deal with privilege-privilege conflicts and role-role conflicts. The role manipulation algorithms are enhanced to disallow the creation of any role that would contain a conflict within itself, and thus not be authorizable to any user or group. We also show the structures that are induced in the role graph when role-role conflicts are present.

Roles have been studied in a variety of contexts and environments; we summarize some of them here. An early reference to roles is found in Lochovsky and Woo [1988], where roles are defined and arranged in a generalization hierarchy and agents representing people are assigned to roles as necessary. Early work by Ting [1988] describes the use of roles to develop application-dependent security controls. Ting's work was also incorporated into a software design system[Ting et al. 1992; Hu et al. 1994]. Thomsen's work talks about roles, subroles, and the mandatory enforcement of policies in a role-based environment [Thomsen 1991]. Baldwin's Named Protection Domains [Baldwin 1990] are very similar to our roles. In Baldwin's model only one Named Protection Domain can be active at one time. Mohammed and Dilts [1994] discuss the design of a role-based model for a specific application in an event-dependent, dynamic environment. von Solms and van der Merwe [1994] give a four-level model where roles form a layer between users on the one hand and transactions and projects on the other. There are obvious parallels between this model and our layered reference model presented in the next section. Fernandez et al. [1994] discuss user group relationships that have great similarity with our role graphs, as discussed in some detail below.

Section 2 contains a detailed description of the reference model for role-based access control (RBAC), introduced in Nyanchama and Osborn

[1994]. This is followed in Section 3 by a discussion of the role graphs that we use to implement role-based access control. Section 4 discusses the role graph administration algorithms that were introduced in Nyanchama and Osborn [1994], and which are enhanced in this paper. These algorithms allow for manipulations of the role graph to represent a dynamic role model for access control. In Section 5, we introduce conflict of interest and show how our reference model suggests five kinds of conflict of interest, some of which others have discussed and some that have not received much attention. Section 6 relates two of these kinds of conflict of interest to our role graph model. Conclusions are presented in Section 7.

2. DEFINITIONS AND REFERENCE MODEL

The role model presented in this section is a general reference model for role-based access control. Our reference model considers that there are three distinct entities to be combined: users, privileges, and roles. In order to be as general as possible, we use the term *group* to denote sets of users, *privilege* to denote an access mode on an object, and *role* to denote sets of privileges.

Access control models are based on some variation of the relationships between objects, subjects, and access modes. Due to the prominence of object-oriented modeling, it is appropriate to regard the relationship between an object and one of its methods or access modes as the basic form of activity or computation. This is why we regard privilege as the basic unit of authorization in this reference model.

In the remainder of this section we discuss the interactions among privileges, roles, and subjects, and authorizations that can be defined for them, as dictated by different security policies.

2.1 Privileges

A *privilege* is a pair (x, m) where x refers to an object, and m is an access mode for object x. The object referred to by x can be any object in an object-oriented environment, any system resource, etc. In specifying a privilege, x can be any name or identifier that uniquely specifies the associated object. The access mode m, can be any valid operation (or method) on x. In systems with simple access modes such as read, write, execute, etc., m is one of these access modes. Where x is an object in an object-oriented environment, m is the execute mode of one of the methods. In transactional systems, m is a transaction that facilitates access to x. The exact nature of x and m is a matter of the application environment and its associated security policy [Nyanchama and Osborn 1993].

2.2 Roles

A *role* is a named set of privileges. It is represented by a pair $(rname, rpset)$, where $rname$ is the name of the role and $rpset$ represents the set of privileges of the role. Given a role r, we use $r.rname$ and $r.rpset$ to refer to the role's name and privilege set, respectively.

Role-role relationships, typically represented by a role hierarchy [Sandhu et al. 1996], are especially interesting. Role relationships are based on role subsetting, which we call the is-junior relationship. We say that role r_i *is-junior to* r_j, iff $r_i.rpset \subset r_j.rpset$.[1] We also say that r_j *is senior to* r_i. By specifying that role r_i is-junior to r_j, one makes available all the privileges of r_i to role r_j. The transitivity of the is-junior relationship was discussed in Nyanchama and Osborn [1994], which also contains a discussion of the notions of common juniors and common seniors.

2.3 Users and Groups

The underlying motivation for RBAC is to help *manage* authorization. One aspect of this management task is that there may be a number of users who should be given the same authorizations. Our model uses the term *group* to refer to a set of users. For example, there could be a group consisting of all users in a given department, or all users working on a particular project. Such sets are constructed where necessary in the application environment, to ease the task of assigning the same authorizations to everyone in the group at once. This notion is similar to the use of the term "group" in UNIX.

Some role models use the term role to refer to a structure that includes both users and privileges [Rabitti et al. 1991; Hu et al. 1994; Simon and Zurko 1997; Sandhu et al. 1996]. Our purpose in separating the discussion about user sets from the role hierarchy discussion is to provide a framework in which interactions resulting from role-role relationships can be analyzed separately from those resulting from user-group memberships and group-group relationships. In any reference model there may be layers of detail that get blurred in some implementations. No matter how the role model is constructed, it is ultimately possible to construct an access matrix [Harrison et al. 1976] by following through on all the implications existing in the defined structures.

2.4 Authorization

The total management of the authorization of privileges to users takes place on three planes (see Figure 1). On the plane closest to the objects of concern are the privileges themselves, without any groupings. Relationships, or perhaps more precisely implications, can exist among privileges. One cause of these implications results from knowledge of the semantics or the procedural structure of the operation being authorized. Such implications are found in the authorization type lattice of Rabitti et al. [1991]. For example, the privilege which allows a user to update an object might imply that any such user should also be able to read the object. Another example is a complex method containing calls to other methods. Depending on the

[1]This is a strict subsetting relationship; roles with the same privileges are regarded as one role.

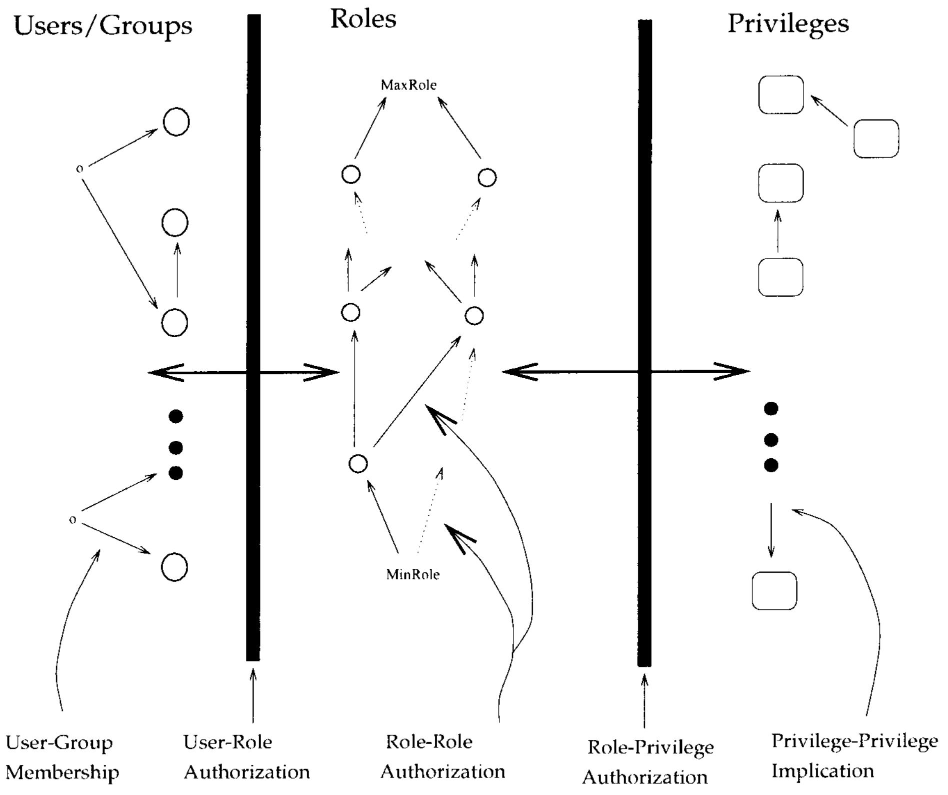

Fig. 1. Three kinds of authorization.

security policy, any authorization to such a complex method may or may not imply separate or direct authorization to these called methods.

Object containment is another cause of implications on the privileges plane. Authorization to read a whole object implies authorization to read any of its individual parts. Authorization to read a set of objects implies authorization to read the members of the set. These implications result from the kinds of information represented in the authorization *object* lattice in Rabitti et al. [1991], which deals with the granularity at which a privilege is specified.

Implications on the privileges plane can also result from inheritance in an object–oriented system. For example, if one can read the salaries of all *Employees* and *Professors* is a subclass of *Employees*, then one should also be allowed to read the salaries of professors.

Roles are considered in the middle plane. In the role plane, the nodes represent named sets of privileges. Here we examine role-role relationships in order to provide tools to help manage role-based security. This is the plane on which role hierarchies such as the role graph model exist [Nyanchama and Osborn 1994].

User and user group relationships can be modeled in the third plane. User groups are created to help manage the assignment of users to roles. For example, all the users in a given project might be put in one group. This group can then be assigned to roles relevant to the project, whereas individual users might be assigned to other roles one at a time. The nodes on the user/group plane represent individual users and groups that were defined because they are of some use in modelling authorization. Information such as the groups a user is assigned to, and which groups are or are not contained in other groups, and other user-user relationships are represented here. The implications that are modeled on this plane are the result of user-group membership or group-group subsetting. If a user is a member of a group and the group is assigned to a role, then the user is also authorized to perform the operations available through the role.

As well as the three planes, there are two other places where authorization takes place that ultimately determine who is authorized to do what. One is in the assignment of users/groups to roles, which is called the User-Role Authorization, and the other is in the assignment of privileges to roles, which is called the Role-Privilege Authorization (see Figure 1). Role-role relationships, which are defined within the role plane, are another way that privileges may be assigned to roles. Thus, given a user-privilege pair, the decision about whether the user is authorized to the privilege is the result of the implications in the user/group plane, the assignment of users/groups to roles, the role-role relationships, the assignment of privileges to roles, and the implications in the privilege plane. All of these implications and layers are intended to provide the necessary detail for a reference model, which can help compare other models or act as a model for an implementation. In an implementation based directly on this model, the layers help reduce the detail with which user-privilege authorization needs to be specified. This approach also makes the management of authorizations very flexible.

2.5 Policies

We gave some examples of different security policies in the above discussion. Separation of the kinds of authorization into the three planes and the two interfaces allows different security policies to be modeled and placed within one framework. An example is the assignment of users to roles. A user may be assigned to one role at one time, which captures the notion of logging on as a certain kind of user. An alternative is that users log on with some or all of their roles active. These are different policies that can be applied to the assignment of users/groups to roles. Another difference in policies is whether this assignment is fairly static or varies dynamically with the activity the user is performing. The model provides a point in the reference architecture containing the appropriate interface on which to focus discussions and comparisons of policy.

The assignment of privileges to roles is another example of where different policies can be modeled. Suppose there is a complex privilege, say hiring an employee, where an employee object is created and various payroll and account objects are modified. This complex privilege involves the execution of some simpler privileges on the employee, payroll, and account objects. The policy regarding the assignment of privileges to roles might be that the privilege of hiring an employee is the only privilege granted; i.e., that any users assigned to this role will not be allowed to individually perform the operations on employee, payroll, or account objects. An alternative policy is that any such complex privilege authorized to a user means that the user *is* automatically authorized to any implied privilege also. (When this is the case in the role graph model, presented in the next section, the implied privileges must also be explicitly added to the role in order for the discussions that follow to reflect this policy.) Whether the implied privileges are directly available is a matter of security policy. The presence of the interface between the role and the privileges planes in the model gives a framework within which such policies can be discussed.

3. ROLE GRAPHS

We have developed a particular model for the role-role relationships on the role plane based on the notion of a *role graph* [Nyanchama and Osborn 1994]. A role graph is an acyclic, directed graph in which the nodes represent the roles in a system, and the edges represent the $is - junior$ relationship. In addition to an arbitrary number of user-defined roles, every role graph has a MaxRole and a MinRole. MaxRole represents the union of all the privileges of the roles in the role graph. MaxRole does not need to have any users authorized to it. The role graph is the place to summarize all of the privileges in the system and to ensure that the common senior relationship [Nyanchama and Osborn 1994] is always defined. All roles in the role graph, except for MaxRole, have a common senior. MaxRole is required for completeness of defined graph properties. MinRole represents the minimum set of privileges available to all roles. MinRole. $rpset$ can be empty. Role graphs have the following *Role Graph Properties*:

—there is a single MaxRole;

—there is a single MinRole;

—the Role Graph is acyclic;

—there is a path from MinRole to every role r_i;

—there is a path from every role r_i to MaxRole;

—for any two roles r_i and r_j, if $r_i.rpset \subset r_j.rpset$, then there must be a path from r_i to r_j.

It is typical in role hierarchy models [Sandhu et al. 1996] for an edge in the role hierarchy, say from role r_i to r_j, to represent the desire for all of the privileges of r_i to be available to role r_j. It is, for example, possible in the role facility of Oracle 7 to assign one role to another for this precise purpose. Such edges are just a way of specifying the is-junior information between these two roles. Our model also shows where such edges are implied by showing at least a path, if not a direct edge, between any two nodes where a subsetting of the privilege sets exists because of addition or deletion of privileges to or from roles, or some other operation on the graph. The meaning of an edge or path in the role graph is that an edge or path exists from role r_i to role r_j iff r_i is-junior to r_j. In analyzing security, it is useful to see that if all the privileges of role r_i are available through role r_j, there is a connection between these two roles in the graph so that when the user is assigned to r_j, the user is also authorized to perform everything in r_i. If it is necessary to have a role with only some of the privileges of r_i, then it is easy, with the algorithms in the next section, to create such a role separately, without an edge from r_i to the new role.

We draw the graphs without redundant edges, i.e., we represent the (acyclic) graph by its transitive reduction [Aho et al. 1972]. We arrange the nodes on the page so that all is-junior edges go up the page. In addition, for every role, we distinguish between its effective privileges and its direct privileges. The *direct privileges* of role r are those that are not contained in the $rpset$ of any of r's juniors. As a consequence, a direct privilege appears only once in a given path in the role graph. The *effective privileges* of role r are the union of its direct privileges and the effective privileges of all its juniors. Note that $r.rpset$ corresponds to the effective privileges of r. Seeing the graph edges that result from privilege assignment and the specification of role-role relationships, as well as seeing direct and effective privileges, helps the person designing the roles to understand the implications of privilege assignments and role relationships.

Acyclicity is a basic property of the role graph [Nyanchama 1994]. It requires that any two roles with a directed edge between them are in a subsetting relationship [Nyanchama and Osborn 1994], and forms the basis for the *is-junior* relationship between two roles. Without this restriction, the union of all privileges in a cycle in the role graph would be available to every role in the cycle. Acyclicity is required so that roles offer differentiated access to the objects in the system.

Consider the example in Figure 2. The graph contains no redundant edges, and the privileges shown are the direct privileges. Table I gives both the direct and effective privileges for each role. Note that the $is - junior$ information can be deduced from the effective privileges and vice versa.

4. ROLE GRAPH ADMINISTRATION ALGORITHMS

Several algorithms are given in Nyanchama and Osborn]1994] for manipulating role graphs. They represent a beginning in the construction of a role management tool. The algorithms discussed there are role addition, role deletion, and two kinds of role partition. The role addition algorithm takes as input a role graph, a new role represented by a role name, a set of

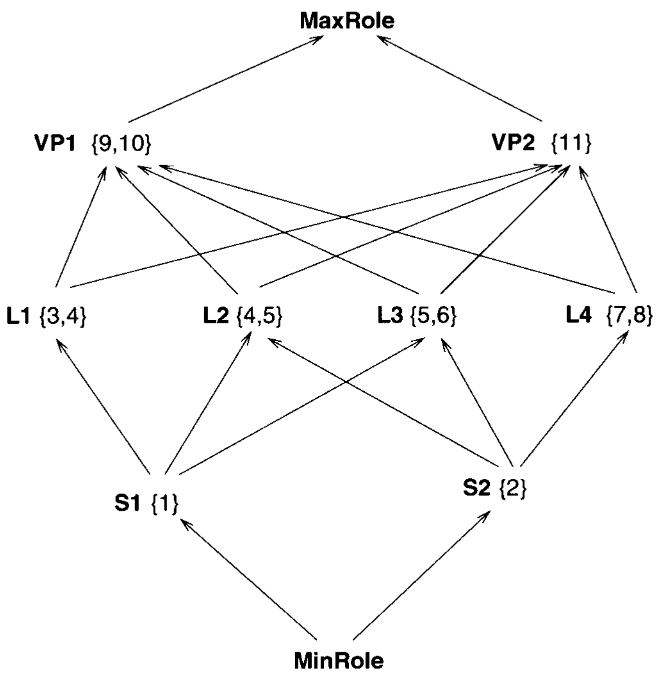

Fig. 2. A sample role graph.

Table I. Roles and Effective Privileges

Role Name	Direct Privileges	Effective Privileges
MaxRole	ϕ	{1,2,3,4,5,6,7,8,9,10,11}
VP1	{9,10}	{1,2,3,4,5,6,7,8,9,10}
VP	{11}	{1,2,3,4,5,6,7,8,11}
L	{3,4}	{1,3,4}
L2	{4,5}	{1,2,4,5}
L	{5,6}	{1,2,5,6}
L	{7,8}	{2,7,8}
S	{1}	{1}
S	{2}	{2}
MinRole	ϕ	ϕ

direct privileges, the roles that are to be its immediate juniors, and the roles that are to be its immediate seniors. With that role addition algorithm, the information is given in terms of role-role relationships. The algorithm uses this information to deduce the resulting effective privileges for the new role and to check the other role graph properties given above. The graph edges and the direct privileges of the new role may be modified as a result.

An alternative way to add a role to a role graph is to just specify its role name and (effective) privileges, and to have an insertion algorithm determine its junior and senior roles by comparing privilege sets. In the first role

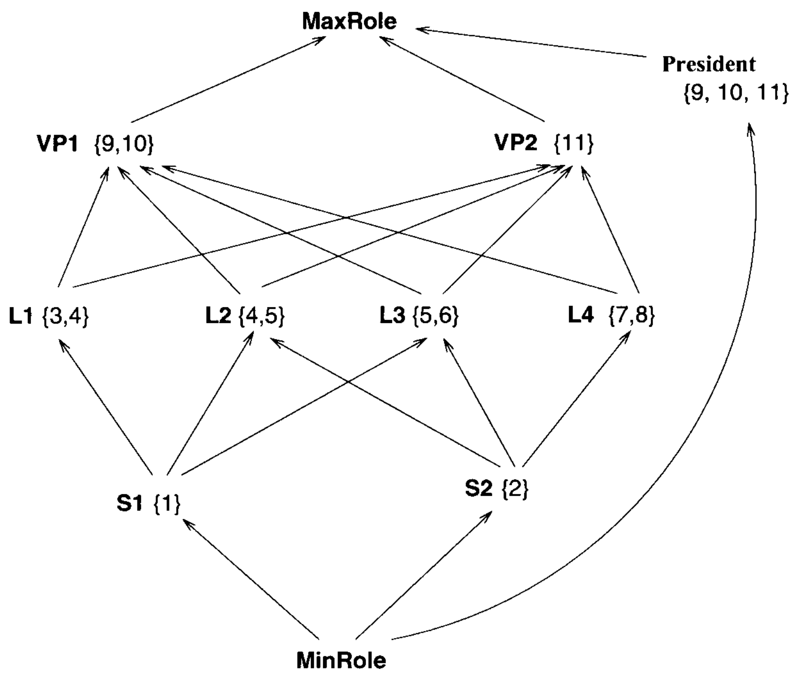

Fig. 3. Role graph showing President role.

addition algorithm, the effective privileges of the new role are determined by the given direct privileges and the specified junior roles. Here, the effective privileges are given, and this information is used to determine juniors, seniors, and the resulting direct privileges. For example, consider adding a role called President to the example in Figure 2, so that the President role contains the privileges that *do* include the direct privileges of the VP roles, but *do not* include any privileges of any role junior to the VP roles. In other words, the effective privileges of this new role should be precisely {9, 10, 11}. This set of effective privileges is not comparable to any other set of effective privileges for the roles in the graph (c.f., Table I). The resulting role graph is shown in Figure 3.

Role deletion described in Nyanchama and Osborn [1994] offers the option of deleting the direct privileges of the role being deleted (the target role) from the graph altogether, or of reassigning all of the direct privileges to the immediate seniors of the target role. Horizontal and vertical partition do not alter the effective privileges of the graph, but split a role into two or more roles. In the case of horizontal partition, the new roles share the juniors and seniors of the original role. With vertical partition, the new roles replace the original role with a path. All of the algorithms check and maintain the role graph properties given above.

In addition to the algorithms given in Nyanchama and Osborn [1994], some simpler ones are also desirable. Two such algorithms would be to add

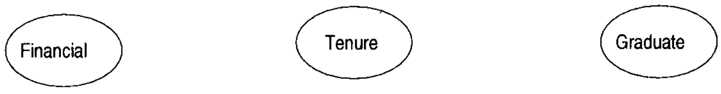

Fig. 4. Three secretarial roles.

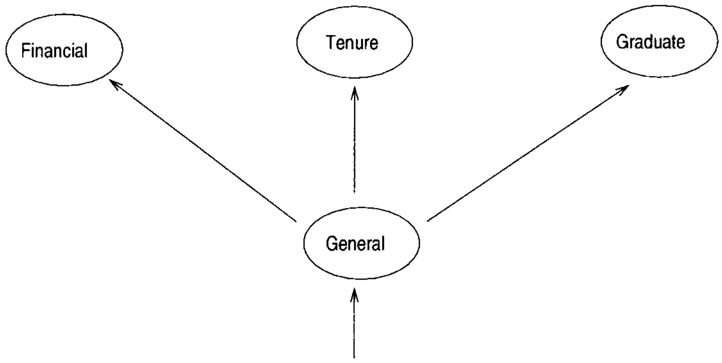

Fig. 5. Addition of general role.

a (new, direct) privilege to a role or to delete a (direct) privilege from a role. For example, in Figure 2, privilege 9 might be added to (the direct privileges of) role **L2**. This would result in privilege 9 being removed from the direct privileges of **VP1**, and will adjust the effective privileges of **VP2** (and of **L2**). The algorithms are given in Appendix A.

The role administration algorithms were implemented in a prototype [Osborn 1996]. The prototype presents the role graphs via an interactive graphical user interface, where each role can be clicked on and its users direct and effective privileges displayed. All the algorithms have runtime polynomial in the number of roles, edges, and, in some cases, the size of the privilege set of a given node (see Appendix A). Thus adding temporary roles, or of roles slightly different from existing ones, is very easy and efficient.

We now consider the policies GP1, GP2, and GP3 of Fernandez et al. [1994]. This model uses the term group to refer to a structure that combines users and privileges. The generalization of several groups in the Fernandez et al. model refers to a group that is more general than the groups of which it is to be a generalization. Consider the example in Figure 4. The three secretarial roles involve handling disjoint kinds of confidential information. A less specific role, the general secretary, would represent in the object-oriented model, a generalization of these three roles. Policy GP1 [Fernandez 1994] says that the subgroups (the more specific secretarial roles) inherit all the rights (or privileges) from supergroups. Supergroups are added to a model by generalization. Policy GP1 then says that the three original secretarial roles should inherit the rights of the new, general secretary role. The general secretary's position in the role graph is *below* the three specialized ones (see Figure 5); i.e., it *is-junior* to the three specialized roles. In Nyanchama and Osborn [1994], the general secretary's

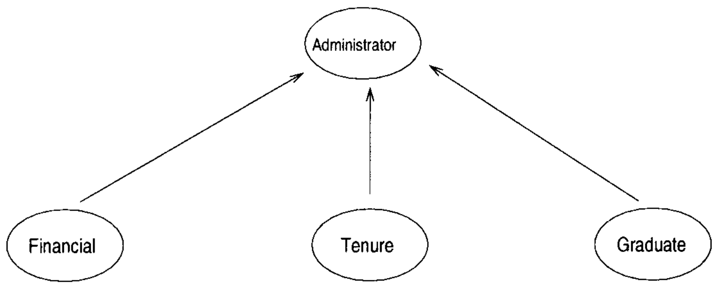

Fig. 6. Addition of administrator role.

role is inserted into the role graph by making it junior to the three roles and senior to some others (or MinRole). Thus the GP1 policy is handled by inserting a new role, a generalization of existing roles, by using the insertion algorithm already given in Nyanchama and Osborn [1994], with the specialized roles given as the desired seniors of the target (new) role.

Policy GP2 deals with specifying the rights of a role formed by the composition of other roles, which is is not completely compatible with defining a senior role, but it can be regarded as that. In Fernandez et al. [1994], an example is given where the company consists of three departments. Someone acting at the company level (or in the company role) has all the rights of people acting at any of the levels in the company. (In Fernandez [1994], the example refers to personnel, engineering, and manufacturing.) In terms of the role graph, creating the company role is equivalent to creating a role senior to the roles corresponding to the personnel, engineering, and manufacturing departments. Policy GP2, translated into role graph terminology, says that the company role acquires all the rights of its junior roles. Thus, creating this composition role is equivalent to creating a new role (for company) and specifying the component roles as its desired juniors. This is accomplished by the role insertion algorithm in Nyanchama and Osborn [1994] by specifying the juniors and specifying any appropriate roles as the desired seniors. If there are no such seniors, MaxRole becomes the (only) senior. For our secretarial example, a role that represents the "office" where the three secretarial roles are present, and which would be assigned to an office administrator or department head, is shown in Figure 6.

Policy GP3 [Fernandez 1994] involves creating a new role with a newly defined set of rights. These rights may be available in some other roles explicitly. The motivation is that a new group is to be created with a subset of the people assigned from other existing groups. The policy says that the rights for this group will be given explicitly. This is the kind of role addition that can be handled by the new role addition algorithm mentioned above. In Fernandez et al. [1994], the motivation is to create a specific new role by specifying its effective privileges, and not relating it to any existing role via a proper subset or proper superset relationship among rights, whereas here the role will be connected into the role graph to conform to role graph properties.

5. CONFLICT OF INTEREST

Consideration of various forms of conflict of interest, separation of duty, or mutual exclusion are often included in discussions of access control models [Simon and Zurko 1997; Ferraiolo et al. 1995; Sandhu 1988]. In complex environments where the actions of ill-intentioned users can create financial or other damage, it is usual to identify combinations of operations that should not be authorized to a single user. For example, for a wholesale application, it might be company policy that the same user should not set the price of a specific item and also be a customer for that item (the employee can either set an unusually low price or damage the item before buying it at a reduced price, thus committing fraud). It would not be a conflict if different items were involved. On the other hand, if the company's conflict policy states that no employee can be a customer of the company, then the roles for customer and employee would be defined to be in conflict. Policies for preventing such fraud are called conflict of interest or separation of duties policies.

We begin this section by enumerating different kinds of conflict of interest as presented in [Simon and Zurko 1997]. A different taxonomy was introduced by Kuhn [1997]. After examining these two taxonomies, we discuss a taxonomy of the kinds of conflict of interest motivated by our reference model in Section 2.

5.1 Conflict of Interest

Simon and Zurko [1997] have enumerated the different kinds of conflict of interest/separation of duties, as follows:

(1) *Strong exclusion or static separation of duty*: Two roles are strongly exclusive if a single user can never be assigned to both roles.

(2) *Weak exclusion or dynamic separation of duty*: Two roles are weakly exclusive if, during a session, a user has activated one of the roles, then they should not activate the weakly exclusive role at the same time. This is called Simple Dynamic Separation of Duty in Simon and Zurko [1997].

(3) *Object-based separation of duty*: A user may perform two different operations on different objects, but may not perform these two operations on the same object.

Simon and Zurko's enumeration of kinds of conflict of interest also includes four more kinds, all having to do with complex tasks involving several interrelated steps, say in a workflow management system. It is common in business transactions to require two signatures before a check is issued, or some like measure to avoid fraud. Such a model is considered, along with an RBAC system, in Bertino et al. [1997] and Thomas and Sandhu [1997]. It is also addressed by the safety condition in Kuhn [1997]. We deal with this to some extent in the discussion of separation of duties in Nyanchama [1994]. Such discussions, while very important, require a model of complex tasks in conjunction with the RBAC model. Such a task model is not presented in this paper, so we cannot discuss parallels to the Operational Separation of Duty, History-based Separation of Duty, Order-dependent Separation of Duty, and Order-independent Separation of Duty defined in Simon and Zurko [1997].

Kuhn's taxonomy [Kuhn 1997] has two axes: the time the exclusion is introduced and the degree to which two roles conflict. As far as time is concerned, mutual exclusion can be defined at role authorization time or at runtime. As Kuhn observes, these correspond to static and dynamic separation of duties, respectively. Kuhn also distinguishes between complete and partial mutual exclusion of roles. Complete exclusion is the same as role-role conflicts in our model, and partial exclusion corresponds to our privilege-privilege conflicts.

5.2 Conflicts of Interest in Our Reference Model

Using our reference model from Section 2, we propose here a different taxonomy for considering different kinds of conflict of interest. The main categories are suggested by the three planes in Figure 1; the last two are given by the interfaces between the planes. The reference model suggests the following kinds of conflicts:

(1) user-user/group-group/user-group conflicts;

(2) role-role conflicts;

(3) privilege-privilege conflicts;

(4) user-role assignment conflicts; and

(5) role-privilege assignment conflicts.

User-user/group-group conflicts arise if two users are never assigned to something together, be it to the same group or to the same role, or two groups must not be made a subset of the same group, etc. There is nothing in Simon and Zurko's list that corresponds to user-user or group-group conflicts.

Role-role conflict means that two roles must never be together. This only makes sense in the assignment of these two roles to a user/group, and corresponds exactly to Static Separation of Duty. (We discuss such conflicts in great detail with respect to our role graph model, below.)

Privilege-privilege conflict means that two privileges must not appear together. This means that these two privileges should never be in the same role. Object-based separation of duty as described above falls into this category. We say this is *static* object-based separation of duty (*dynamic* object-based separation of duty will be defined shortly). There might be other static privilege conflicts not based on objects; e.g., a conflict between performing the same operation on two different objects. An example is disallowing performance of two full-time jobs to appear together in a role (and subsequently be assigned to a user).

On the other hand, there may be two functions that can be performed by a user, but with restrictions that the roles associated with the functions cannot be held by the same user simultaneously. This may necessitate the definition of sessions, with specific roles associated with particular functions; a user will be allowed to execute only one function in a given session. It will require the user to terminate the session before executing the other function.

The user-role assignment interface of our reference model is the place where both static and dynamic issues of user-role assignment are discussed, i.e., if there is a difference between the roles to which a user is authorized and the roles that are active during a session, then there is both a static and a dynamic mapping that takes place at this interface. Similarly, in user-role assignment, there can be a static and a dynamic *conflict*. Dynamic separation of duty, as defined in Simon and Zurko [1997], is modeled by the dynamic aspects of user-role assignment, and therefore is modeled by dynamic user-role conflicts. Static user-role conflicts state that a user should never be assigned to a role (perhaps because of lack of qualifications or clearance), and are not present in Simon and Zurko's list.

Similarly, a role-privilege assignment can deal with static or dynamic issues. A static conflict says that a certain privilege should never be assigned to a certain role. (Someone acting in the clerk role should never be allowed to fire the president.) This is just good role design. Dynamic role-privilege assignment deals with whether or not a privilege should be assigned to a role, given that other privileges are already in the role. A conflict here models (dynamic) object-based separation of duty [Simon and Zurko 1997], and possibly some task-based constraints concerning two operations on the same object. Since our RBAC reference model defines privileges at the instance level (i.e., in terms of objects), dynamic role-privilege conflicts could specify that two operations on the same object must not appear in one role, but might appear in separate roles.

With a flexible role definition facility, enough roles could be defined, based on individual objects, to handle object-based separation of duty by just having a great many roles. One disadvantage of reference models is that, although they may provide a way of succinctly defining components and distinguishing between things, implementing them directly is not the most efficient thing to do. We prefer that object-based separation of duty be handled by a task management system [Bertino 1997], or by specifying privileges in terms of object types, with constraints based on instances.

In conclusion, Static Separation of Duty is modeled by role-role conflicts; Dynamic Separation of Duty by dynamic user-role conflicts; and Object-based Separation of Duty by dynamic role-privilege conflicts. The other types of conflicts in Simon and Zurko [1997] of interest here should not be discussed without a model for complex tasks. Although the above conflicts are the most important, we have also identified user-user conflicts, other privilege-privilege conflicts, static user-role conflicts, and possibly static role-privilege conflicts as worthy of some consideration.

6. CONFLICT OF INTEREST IN THE ROLE GRAPH MODEL

Of all the types of conflict of interest above, the ones that relate most closely to our role graph model are privilege-privilege conflict and role-role conflict. In this section we discuss these two kinds of conflict as they relate to our role-graph model and its current implementation.

6.1 Conflicting Privileges

Privilege-privilege conflicts, as discussed above, indicate that, for whatever reason, the two privileges declared to be in conflict should never appear together. The reason for this might be object-based or based on the access mode, or just on the totality of the two privileges.Whether or not these two privileges appear together in a role, whether they are added directly one at a time, or result in being there because of the relationships between roles that contain the conflicting privileges should be taken into account in our role graph model. The problem role might be a common senior of two roles that separately contain the conflicting privileges. This conflict must be considered whenever a role changes. A role that contains conflicting privileges should not be allowed in the role graph because such a role cannot be assigned to any user. The only exception is MaxRole, which must contain all privileges in the system. When there are privilege-privilege conflicts, MaxRole should not be assigned to any user. The rejection of other roles that contain privilege-privilege conflicts should not cause a problem in role design because the flexibility in defining roles with only slight differences, using the role graph algorithms, makes it possible to define assignable roles free of such conflicts to any level of detail consistent with the conflicts.

If two such privileges should never appear together, then they must ultimately not be assigned to a single user. Thus, privilege-privilege conflicts must also be taken into account when user-role assignments are made. Suppose we have two roles, r_1 and r_2, r_1 contains only privilege p_1, and r_2 contains only privilege p_2, their only common senior is MaxRole, and p_1 and p_2 conflict. We must not allow a user to be assigned to both r_1 and r_2.

As with all policy issues, the statement of the actual conflict is given by the role system administrators.

We denote *conflicting privileges* as follows: two privileges, p_1 and p_2, which have been defined to conflict, are denoted by $p_1 <> p_2$. As noted above, no role should contain conflicting privileges because that role could never be authorized to any user. This is captured by the first constraint:

Privilege Conflict of Interest Constraint: No role (except MaxRole) may contain two privileges that have been defined to conflict.

In order to maintain this constraint within our role management system, we need to assume the existence of a set of pairs of conflicting privileges, which we call $P-Conflicts$. P-Conflicts must be checked when the following role maintenance operations are executed:

(1) when a privilege is added to an existing role (the conflict might arise within the specified role or in a senior role when the effective privileges of such senior roles are adjusted);

(2) when a new role is created (existence of conflicting privileges could result from the given direct privileges, exist in the new role from the formation of the effective privileges from the new role's specified junior roles, or might result in a senior of the new role);

(3) when an edge is inserted (the role at the head of the new edge or one of the seniors of this role might inherit new privileges as the result of the new edge, which might result in a conflict.)

As with all operations on the role graph, if any inconsistency is discovered in performing the operation, the user is told of the problem, and the role graph is left unchanged. In the algorithms in Appendix A, $P-Conflicts$ is considered wherever appropriate.

6.2 Role-Role Conflicts

A statement by the role system administrator that two roles conflict is a very powerful statement. If it is really the case that only some parts of the two roles conflict, then this should be represented by privilege-privilege conflicts or the roles should be redesigned, so that only the minimal sets of privileges that need to be declared as conflicting roles are actually defined in the conflicting roles. An example from Figure 2 is found in Roles L1 and L4. Suppose the system administrator wants to declare that these two roles conflict, but really it is only privilege {3} that conflicts with all the privileges in L4. Then L1 can be expanded into L5, with direct privilege {3}, and L1 with direct privilege {4}. The graph is altered to have edges from S1 to L5, and L5 to L1, L1 has the same effective privileges as before, and the conflict can be declared to be between L5 and L4.

Two roles, r_1 and r_2, which have been defined to conflict, are denoted $r_1 << >> r_2$.

A role-role conflict means that the two roles should never appear together. This implies they should never be assigned to a single user, which is checked on user-role assignment. This gives us the following constraint:

User-Role Conflict of Interest Constraint: No user should be authorized to two roles that have been declared to conflict.

In the remainder of this section we use the role graph to characterize other properties arising from role-role conflicts. To realize this goal, we must discuss what a role-role conflict means. Since it is also possible to give privilege-privilege conflicts or to define other roles with fewer privileges, when two roles are declared to be in conflict, any user authorized to one of the roles is not allowed to perform *any* of the privileges of the conflicting role. If this is not the case, then the roles need to be redefined with a finer granularity. We formalize this into the following principle, which is a fundamental assumption underlying what follows:

Role Conflict of Interest Principle: If two roles are declared to conflict, then a user authorized to one of the roles must not be authorized to any of the privileges of the other role.

Consider the example from Figure 2 again. The roles L1 and L3 have disjoint direct privileges, but their effective privileges include {1} because both roles have role S1 as a junior. Thus, roles L1 and L3 cannot be declared to be in conflict because anyone authorized to one of them would be able to perform at least one of the privileges of the other.

In fact, this example points out a more subtle aspect of role conflicts; but to discuss this, we need to look more closely at the user-role authorization mapping. Authorizing a user to role L1 makes privileges {1, 3, 4} available to this user. This, by default, authorizes the user to role S1, since all of its privileges are authorized when L1 is authorized. Suppose that roles S1 and S2 are declared to conflict. Authorizing a user to L2 means that this user is authorized, indirectly or by default, to S1 and S2. These have been been declared to be in conflict. Thus, declaring roles to be in conflict with one another has consequences for their common seniors: it makes any common seniors unauthorizable. As with the privileges conflict case, any role that, due to the conflict of interest policy, is unauthorizable will not be allowed in the role graph. Thus we analyze these situations and disallow any role-role conflicts that have any common seniors other than MaxRole.

The following results formalize the above discussion. Note that Theorems1 and 2 were discovered independently by Kuhn [1997].

Definition 1. Role Independence: Two roles r_i and r_j are independent if their only common junior is MinRole.

THEOREM 1. *Conflicting roles must be independent.*

PROOF 1. Suppose two roles r_i and r_j conflict but are not independent. That is, there is a role r_k not equal to MinRole that is a common junior to r_i and r_j. The role r_k must contain some privilege common to $r_i.rpset$ and $r_j.rpset$. This means that a user authorized to r_i is also authorized to part of the privileges of r_j. This violates the role conflicts of interest principle. □

THEOREM 2. *Conflicting roles must have no, seniors other than MaxRole.*

PROOF 2. Suppose such a situation were allowed, i.e., there are two roles r_i and r_j declared to conflict that have a common senior r_k not equal to MaxRole. r_k contains privileges from both r_i and r_j, and by the role conflict of interest principle, cannot be authorized to any user. Thus conflicting roles must have no, seniors other than MaxRole. □

Definition 2. Conflict-Consistent: A role graph is said to be consistent with respect to conflict of interest if for every pair of roles r_i and r_j such that $r_i << >> r_j$, r_i and r_j have no, junior other than MinRole and no, senior other than MaxRole.

6.3 A Role Graph with Role-Role Conflicts

In this section we characterize some allowable role graph structures for conflict-consistent role graphs.

By the above two theorems, conflicting roles must be on independent paths from MinRole to MaxRole. In a conflict-consistent role graph, roles that are on the same path from MinRole to MaxRole may be assigned to the same user, and are not in conflict with each other. The conflict relationship induces a partitioning of the roles in the role graph into collections of roles that can be authorized together. Collections of roles that can be assigned to a user without causing conflict are termed *nonconflicting role collections*. Specifically:

Definition 3. Nonconflicting role collections: Two roles r_i and r_j belong to a nonconflicting collection if there are no conflicting privileges in $r_i.rpset$ and $r_j.rpset$, and r_i does not conflict with r_j.

Consider the example in Figure 7. Company policy forbids anyone with warehouse privileges from buying anything from the company. People in the personnel department have no warehouse privileges. The conflict then is between the roles Customer and Warehouse. All roles on any path from MinRole to MaxRole through Warehouse are also in conflict with Customer. Thus, if we look at Figure 8, we get two disjoint vertical regions of the graph for each role-role conflict. Each role involved in a conflict creates a vertical region, including *all* of its juniors and seniors, within the graph. Each role not belonging to any of these regions must not be in is-junior or is-senior relationships with any role in one of the vertical regions. In a sense, these noninvolved roles form another vertical region of their own.

Nonconflict, on the other hand, tells us which regions of the graph can be authorized to the same user. Roles in our example, like the Personnel/Payroll role, that are not involved in any conflict relationship could be assigned with Customer or with Warehouse, but not with both for a single user, since a single user cannot be assigned both a warehouse role and a customer role. Figure 9 shows what regions a single user's authorizations may be in.

We can compute the nonconflicting role collections in the following way. Construct an n x n matrix, C, where n is the number of roles not including MinRole or MaxRole. In this matrix, $c_{i,j}$ is 1 if there is a conflict between r_i and r_j, and 0 otherwise. For each role conflict defined by the policy, indicate the conflict in the matrix. For each such conflict $r_i << >> r_j$, for each

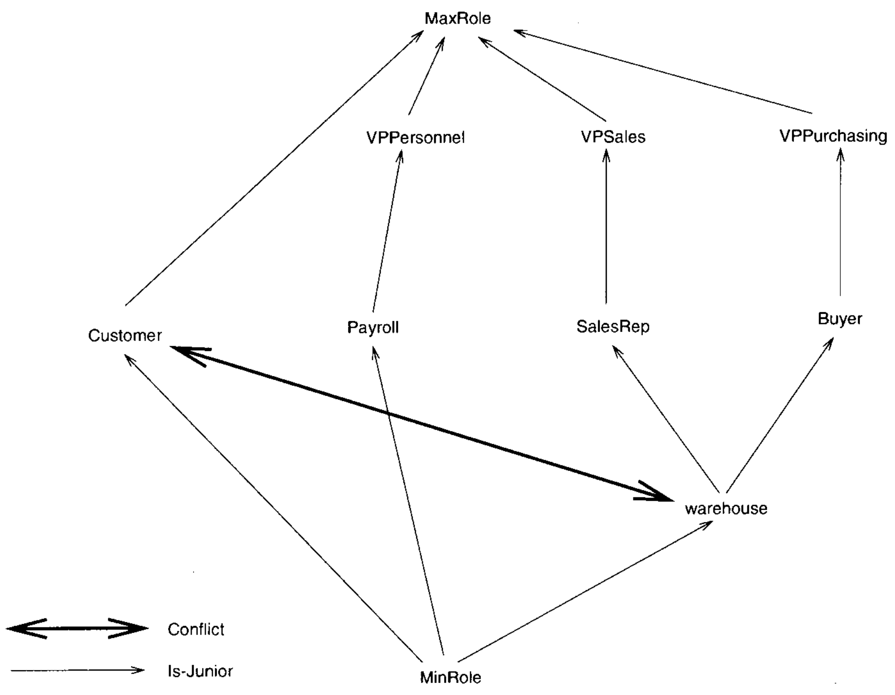

Fig. 7. Role graph with conflicting roles.

junior and senior of r_j, call it r_k, there is also a conflict between r_i and r_k, so $c_{i,k}$ is also set to 1. This matrix then represents all the role-role conflicts when juniors and seniors of the original role conflicts are taken into account. Table II contains the conflict matrix for the example in Figure 8.

Now take the dual of this matrix; i.e., substitute 0's for 1's and vice versa. Call this matrix N. This matrix has a 1 whenever the two roles can be assigned together, and 0 otherwise. This can be considered an undirected graph. Any clique [Bondy and Murty 1976] of this graph is a nonconflicting collection; i.e., any clique represents a set of roles such that, for every pair of roles in the set, the two roles can be assigned together to a single user without violating the conflict of interest information defined in the policy. This matrix appears in Table III. Figure 9 shows this information graphically, by indicating the regions in which roles can be assigned together, superimposed on the original graph.

The conflict relationship between roles is *not* transitive. Consider the example in Figure 10 where the warehouse and payroll roles conflict and distribution and payroll roles conflict, but the warehouse and distribution roles do not conflict with each other. This represents a company where the same person cannot decide what to pay him or herself, unless given the payroll job, but one person might be a part-time driver (i.e. be in distribution) and work in the warehouse part time. For each of these divisions of

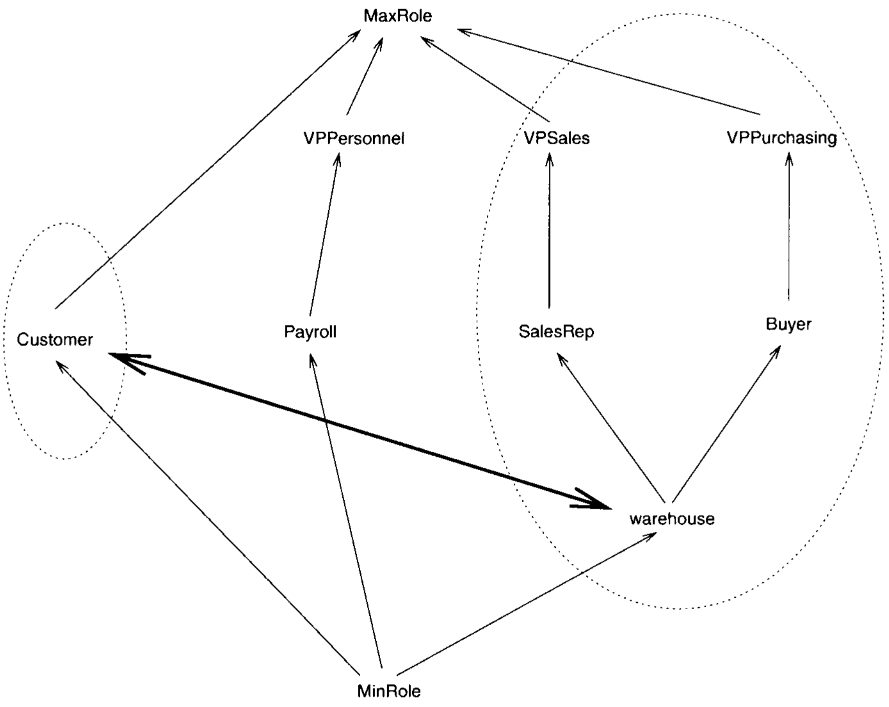

Fig. 8. Vertical regions defined by a conflict.

Table II. Role-Role Conflict Matrix

	Customer	VPPers	Payroll	VPS	Sales-Rep	Warehse	VPPurch	Buyer
Customer	0	0	0	1	1	1	1	1
VPPersonnel	0	0	0	0	0	0	0	0
Payroll	0	0	0	0	0	0	0	0
VPSales	1	0	0	0	0	0	0	0
Sales-Rep	1	0	0	0	0	0	0	0
warehouse	1	0	0	0	0	0	0	0
VPPurchasing	1	0	0	0	0	0	0	0
Buyer	1	0	0	0	0	0	0	0

Table III. Dual of Conflict Matrix

	Customer	VPPer	Payroll	VP	Sales-Rep	Warehse	VPPurch	Buyer
Customer	1	1	1	0	0	0	0	0
VPPersonnel	1	1	1	1	1	1	1	1
Payroll	1	1	1	1	1	1	1	1
VPSales	0	1	1	1	1	1	1	1
Sales-Rep	0	1	1	1	1	1	1	1
warehouse	0	1	1	1	1	1	1	1
VPPurchasing	0	1	1	1	1	1	1	1
Buyer	0	1	1	1	1	1	1	1

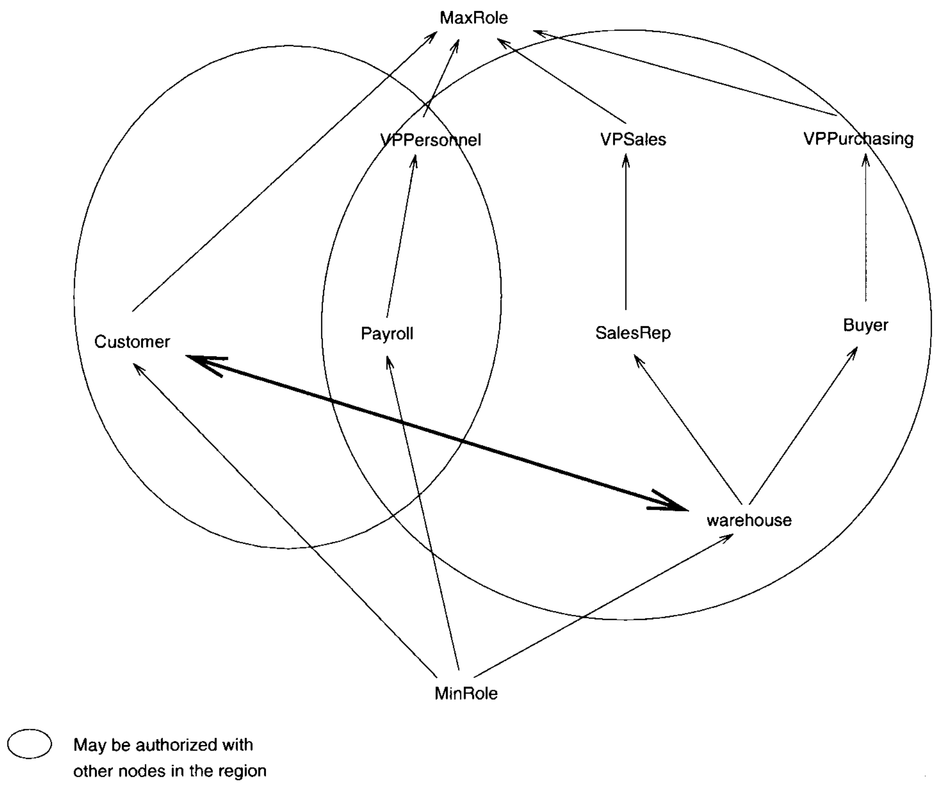

Fig. 9. Regions of the graph that can be assigned to a single user.

the company, just two roles are shown, and they are called top and bottom, thus giving us WT and WB for warehouse top and warehouse bottom, etc.

The conflict between WB and PB and between PB and DB induce the vertical conflict collections shown in Figure 11. The above algorithm for constructing the nonconflicting role collections produces the collections shown in Figure 12. Note that in Figure 12, the roles are rearranged on the page to allow the collections to be drawn.

6.4 Privilege versus Role Conflicts

In the above discussion of role conflicts, we assumed that when two roles r_i and r_j are defined to conflict, there is a privilege conflict between every privilege in r_i and every privilege in r_j. Thus we could easily convert from a statement of conflict of interest given in terms of a role-role conflict to a set of privilege conflicts. However, when the conflict of interest information is given in terms of privilege conflicts, there might be two roles r_i and r_j, such that some privilege in r_i conflicts with some, but not all, of the privileges of r_j. In this case there is no direct mapping from a conflict of interest

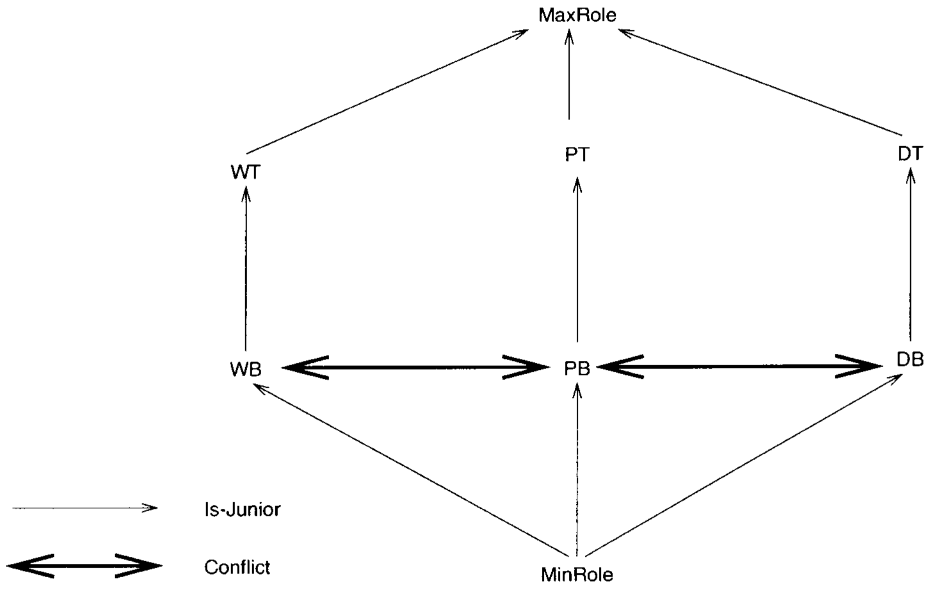

Fig. 10. Nontransitivity of conflict.

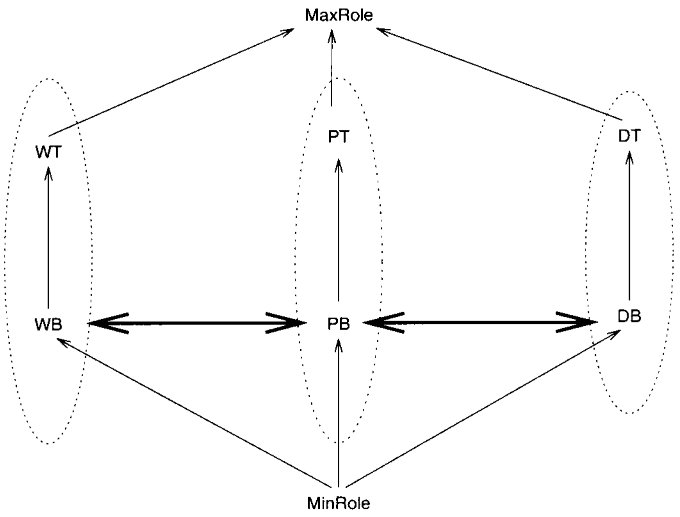

Fig. 11. Vertical regions.

statement in terms of privilege conflicts into a role-role statement of conflict of interest.

The main difference between the two kinds of knowledge comes when a user is to be assigned to a new role. If the conflict information is in terms of privilege conflicts, then the total privileges of all roles currently authorized to the user must be constructed and compared with the total privileges of

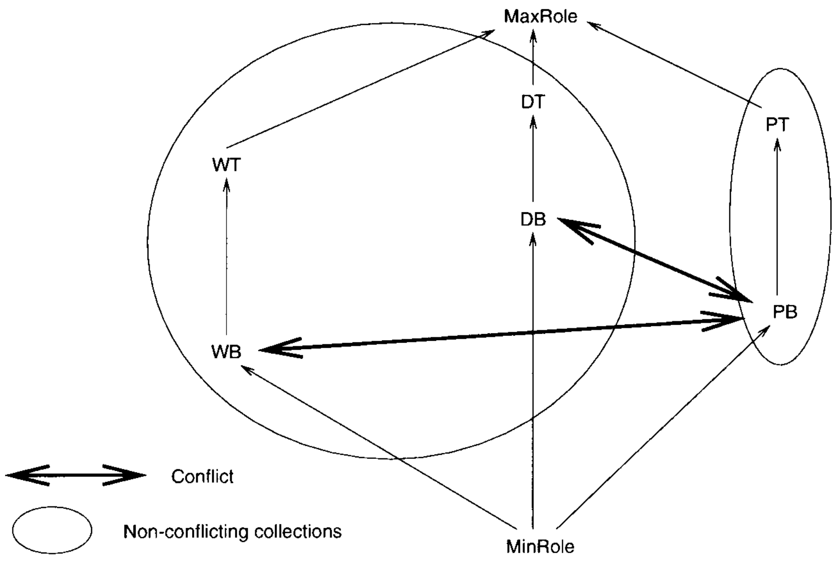

Fig. 12. Nonconflicting collections.

the desired role. These two sets must be compared with the information held in P-Conflicts to see if the desired role is allowed.

For role conflicts, when a user is to be authorized to a role, the nonconflicting role collection information could be used to see if the desired role is within the same collection as one that contains all of the user's current roles. Alternately, the role conflict information can be mapped into privilege conflict information, and the method for privilege conflict checking can be used.

It would seem, then, that although we can produce elegant information about nonconflicting role collections when conflicts are described as role-role conflicts, specifying conflicts in terms of privilege-privilege conflicts allows for a finer granularity. In both cases we can build the required safeguards to make sure that a single user cannot be authorized to do things that violate the given conflict of interest policy.

7. CONCLUSIONS

We have described in some detail the reference model for role-based access control, introduced in Nyanchama and Osborn [1994]. We have also used the reference model to provide a taxonomy of conflict of interest types that may occur. Two of the identified conflict types—privilege-privilege conflicts and role-role conflicts—were developed further for our role graph model for RBAC. When role-role conflicts are present, the role graph can be divided into collections of roles that can be assigned together to a single user. This provides a very elegant way of showing the consequences of conflict of interest considerations in role-based access control. Privilege-privilege conflicts allow a finer granularity for expressing conflicting activities but do not yield such elegant results on the properties of the role graph.

APPENDIX

APPENDIX A. ALGORITHMS

In this section we present algorithms for both versions of role addition, role deletion, privilege insertion and deletion, and edge insertion and deletion. Details of fixing the graph after a change, to make it conform to role graph properties, are omitted. In some cases, a complex adjustment of direct and effective privileges in roles senior to the place of change is required. Role splitting, discussed in Nyanchama and Osborn [1994], can be accomplished by a sequence of role deletions and insertions.

It is important to note that all of these algorithms have runtime polynomial in the number of nodes and edges in the role graph. Some of the algorithms indicate that a cycle-detecting algorithm must be run. Detecting cycles can be done by depth-first traversal, which is polynomial in the size of the graph. Reestablishing role graph properties involves seeing if new edges are implied, adding them, and then running the transitive reduction algorithm, which is polynomial in the size of the graph [Aho et al. 1972]. Other parts of the algorithms involve examining all the nodes to adjust the direct and effective privilege sets, which is polynomial in the number of nodes and the size of the privilege sets. The algorithms have all been implemented in an interactive tool, which allows very flexible management of role graphs.

We include the logic necessary to cope with privilege conflicts as expressed as pairs in a set called P-Conflicts.

Algorithm 1: RoleAddition1(RG, n, Seniors, Juniors, P-Conflicts).

```
Input: RG = $\langle \mathcal{R}, \rightarrow \rangle$ (the role graph),
  $n$, /*the new role to be added (role name along with its proposed
    direct privilege set) */
  Seniors, /* proposed immediate Seniors for $n$ */
  Juniors, /* proposed immediate Juniors for $n$ */
  P-Conflicts. /* set of pairs of conflicting privileges */
Output: The role graph with $n$ added and role graph properties
    intact, or RG unchanged if an error is detected.
Method:
  Var $r$, $r_i$, $r_j$, $r_s$: role;
  Begin
    $\mathcal{R} := \mathcal{R} \cup \{n\}$; /* Add new node to RG */
    For all $r_s \in$ Seniors Do add the edge $n \rightarrow r_s$; /* Add given edges
    to RG */
    For all $r_j \in$ Juniors Do add the edge $r_j \rightarrow n$;
    If RG has any cycles /* Detect cycles */
      then abort (message: Graph is not acyclic);
```

```
        /* Reestablish role graph properties */
        /* Adjust Direct and Effective of affected roles */
        For all $r_i$, $r_j \in \mathcal{R}$ Do /* Detect duplicate roles */
            If Effective($r_i$) = Effective($r_j$)
                Then abort (message: Duplicate roles have been created.
                    Give role names of $r_i$ and $r_j$.)
/* Conflict of Interest Detection */
    For every role $r \in \{n\} \cup \mathcal{R}$ - MaxRole Do
        If Effective($r$) contains a pair of privileges which is in P-Conflicts
            Then abort (message: Role addition creates a conflict);
    end.
```

Algorithm 2: RoleAddition2(RG, n***, P-Conflicts)***.

```
begin Input: RG = $\langle \mathcal{R}, \rightarrow \rangle$ (the role graph),
    $n$, /*the new role to be added (role name along with its proposed
        effective privilege set) */
    P-Conflicts. /* set of pairs of conflicting privileges */
Output: The role graph with $n$ added and role graph properties
        intact, or RG unchanged if an error is detected.
Method:
    Var $r$: role;
    Begin
        For all $r \in \mathcal{R}$ Do /* Check if node is already there */
            If Effective($n$) = Effective($r$)
                Then abort (message: this role is already in the graph);
        $\mathcal{R} := \mathcal{R} \cup \{n\}$; /* Add new node to RG */
        For all $r \in \mathcal{R}$ Do /* Create edges to seniors */
            If Effective($n$) $\subset$ Effective($r$)
                Then add the edge $n \rightarrow r$;
        For all $r \in \mathcal{R}$ Do /* Create edges from juniors */
            If Effective($r$) $\subset$ Effective($n$)
                Then add the edge $r \rightarrow n$;
        /* Reestablish role graph properties */
        /* Adjust Direct and Effective of affected roles, including $n$ */
/* Conflict of Interest Detection */
        For every role $r \in \{n\} \cup \mathcal{R}$ - MaxRole Do
            If Effective($r$) contains a pair of privileges which is in
                P-Conflicts
                Then abort (message: Role addition creates a conflict);
    end.
```

Algorithm 3: PrivilegeAddition(RG, n***,*** p***, P-Conflicts)***.

```
Input: RG = $\langle \mathcal{R}, \rightarrow \rangle$ (the role graph),
    $n$, /* the role to which a privilege is to be added */
    $p$, /* the privilege to be added to role $r$ */
```

```
P-Conflicts. /* set of pairs of conflicting privileges */
Output: The role graph with privilege $p$ added to role $r$, and the role
      graph properties intact, or RG unchanged if an error was
      detected.
Method:
  Var $r$: role;
  Begin
    If $p \in$ Effective( $n$) /* $p$ is already in $n$ -- do nothing */
      Then return;
    Direct($n$):= Direct($n$) $\cup$ {$p$}; /* add $p$ to Direct privileges of $n$ */
    Effective($n$):= Effective($n$) $\cup$ {$p$}; /* and to Effective of $n$ */
    /* Reestablish role graph properties */
    /* Adjust Direct and Effective of affected roles */
    If RG has any cycles /* Detect cycles */
      then abort (message: Graph is not acyclic);
/* Conflict of Interest Detection */
    For every role $r \in \{n\} \cup \mathcal{R}$ - MaxRole Do
      If Effective($r$) contains a pair of privileges which is in
        P-Conflicts
        Then abort (message: Privilege addition creates a conflict);
  end.
```

Algorithm 4: RoleDeletion(RG, n***,*** inv***, P-Conflicts)***.

```
Input: RG = $\langle \mathcal{R}, \rightarrow \rangle$ (the role graph),
  $n$, /* The role to be deleted */
  $inv$, /* Boolean, if true then $n$'s privileges are to be kept in the role
  graph */
  P-Conflicts. /* set of pairs of conflicting privileges */
Output: The role graph with $n$ deleted and role graph properties
      intact, or RG unchanged if an error was detected.
Method:
  Var $\mathcal{S}$, $\mathcal{J}$: set of roles;
    $r_j$, $r_s$: role;
    $\mathcal{S}$:= FindImmediateSeniors(RG, $n$); /* Find immediate seniors */
    $\mathcal{J}$:= FindImmediateJuniors(RG, $n$); /* Find immediate juniors*/
    For all $r_j \in \mathcal{J}$ Do /* Connect juniors to seniors */
      For all $r_s \in \mathcal{S}$ Do
        add edge $r_j \rightarrow r_s$;
    For all $r_s \in \mathcal{S}$ Do /* Remove edges adjacent to $n$ */
      remove edge $n \rightarrow r_s$;
    For all $r_j \in \mathcal{J}$ Do
      remove edge $r_j \rightarrow n$;
    If $inv$ /* Transfer privileges to seniors */
      Then For all $r_s \in \mathcal{S}$ Do
        Direct($r_s$):= Direct($r_s$) $\cup$ Direct($n$)
      /* Adjust Direct and Effective of affected roles */
      /* Reestablish role graph properties */
      remove $n$ from $\mathcal{R}$; /* Remove the role from the graph */
/*Conflict of interest consideration:
  this operation cannot create any new conflicts */
  end.
```

Algorithm 5: PrivilegeDeletion(RG, n, p, P-Conflicts).

```
  Input: RG = $\langle \mathcal{R}, \rightarrow \rangle$ (the role graph),
    $n$, /* The role containing the privilege to be deleted */
    $p$, /* The privilege to be deleted from role $n$ */
    P-Conflicts. /* set of pairs of conflicting privileges */
  Output: The role graph with $p$ deleted from $n$ and role graph
        properties intact, or RG unchanged if an error was detected.
    Begin
      If $p \notin$ Direct($n$) /* $p$ must be a direct privilege of $n$ */
        Then return (message: $p$ not a direct privilege of $n$);
      Direct($n$):= Direct($n$) - {$p$}; /* Delete $p$ from role $n$ */
      Effective($n$):= Effective($n$) - {$p$};
      For all $r \in \mathcal{R}$ Do /* Check for duplicate roles */
        If Effective($r$) = Effective($n$)
          Then abort (message: Duplicate roles have been created.
            Give role names of $r$ and $n$.)
      /* Reestablish role graph properties */
      /* Adjust Direct and Effective of affected roles */
  /*Conflict of interest consideration:
    this operation cannot create any new conflicts */
    end.
```

Algorithm 6: EdgeInsertion(RG, $r_1 \rightarrow r_2$, P-Conflicts).

```
  Input: RG = $\langle \mathcal{R}, \rightarrow \rangle$ (the role graph),
    $r_1 \rightarrow r_2$, /* the edge to be added to RG */
    P-Conflicts. /* set of pairs of conflicting privileges */
  Output: The role graph with edge $r_1 \rightarrow r_2$ added, and role graph
        properties intact, or RG unchanged if an error was detected.
  Method:
    Var $r_i$, $r_j$: role;
    Begin
      IF there is a path from $r_1$ to $r_2$ /* See if edge already there*/
        Then return; /* or can be inferred */
      Add edge $r_1 \rightarrow r_2$ to RG;
      Direct($r_2$):= Direct($r_2$) - Effective($r_1$); /* Adjust privileges */
      Effective($r_2$):= Effective($r_2$) $\cup$ Effective($r_1$);
      /* Reestablish role graph properties */
      /* Adjust Direct and Effective of affected roles */
      For all $r_i$, $r_j \in \mathcal{R}$ Do /* Detect duplicate roles *
        If Effective($r_i$) = Effective($r_j$)
          Then **abort** (message: Duplicate roles have been created.
            Give role names of $r_i$ and $r_j$.)
  /* Conflict of Interest Detection */
      For every role $r_i \in \mathcal{R}$ - MaxRole Do
        If Effective($r_i$) contains a pair of privileges which is in
          P-Conflicts
          Then **abort** (message: Edge insertion creates a conflict);
    end.
```

Algorithm 7: EdgeDeletion(RG, $r_1 \rightarrow r_2$***, P-Conflicts)***.

```
Input: RG = ⟨$\mathcal{R}$, →⟩ (the role graph),
  $r_1 \rightarrow r_2$, /* the edge to be deleted from RG */
  P-Conflicts. /* set of pairs of conflicting privileges */
Output: The role graph with edge $r_1 \rightarrow r_2$ deleted, and role graph
  properties intact, or RG unchanged if an error was detected.
Method:
  Var $r_i$, $r_j$: role;
  Begin
    If there is no edge $r_1 \rightarrow r_2$ /* See if valid input */
      Then return;
    If $r_1$ , $r_2$ is MinRole or MaxRole /* Do not delete edges adjacent */
      Then return; /* to MaxRole or MinRole */
    Delete edge $r_1 \rightarrow r_2$ from RG;
    /* Reestablish role graph properties */
    /* Adjust Direct and Effective of affected roles */
    For all $r_i$, $r_j \in \mathcal{R}$ Do /* Detect duplicate roles */
      If Effective( $r_i$) = Effective( $r_j$)
        Then abort (message: Duplicate roles have been created.
          Give role names of $r_i$ and $r_j$.)
/*Conflict of interest consideration:
  this operation cannot create any new conflicts */
  end.
```

ACKNOWLEDGMENTS

We thank Chi Zhang for a very careful implementation of the algorithms. T.C. Ting asked a good question that led to the inclusion of the second role addition algorithm. This research was supported by the Natural Sciences and Engineering Research Council of Canada.

REFERENCES

AHO, A. V., GAREY, M. R., AND ULLMAN, J. D. 1972. The transitive reduction of a directed graph. *SIAM J. Comput. 1*, 2 (June), 131–137.

BALDWIN, R. 1990. Naming and grouping privileges to simplify security management in large databases. In *Proceedings of the IEEE Symposium on Research in Security and Privacy* (Oakland, CA). IEEE Computer Society Press, Los Alamitos, CA, 116–132.

BERTINO, E., FERRARI, E., AND ALTURI, V. 1997. A flexible model for the specification and enforcement of role-based authorizations in a workflow management system. In *Proceedings of the 2nd ACM Workshop on Role-Based Access Control* (Fairfax, VA, Nov. 6-7). ACM Press, New York, NY, 1–12.

BONDY, J. A. AND MURTY, U. S. R. 1976. *Graph Theory with Applications*. Macmillan Press Ltd., Basingstoke, UK.

FERNANDEZ, E. G., WU, J., AND FERNANDEZ, M. H. 1994. User group structures in object-oriented database authorization. In *Proceedings of the IFIP Working Group 11.3 Working Conference on Database Security*. Elsevier North-Holland, Inc., Amsterdam, The Netherlands, 57–76.

FERRAIOLO, D., CUGINI, J., AND KUHN, D. R. 1995. Role based access control: Features and motivations. In *Proceedings of the 11th Annual Conference on Computer Security Applications*. IEEE Computer Society Press, Los Alamitos, CA, 241–248.

HARRISON, M., RUZZO, W., AND ULLMAN, J. 1976. Protection in operating systems. *Commun. ACM 19*, 8.

HU, M.-Y., DEMURJIAN, S. A., AND TING, T. C. 1994. Unifying structural and security modeling and analyses in the ADAM object-oriented design environment. In *Proceedings of the IFIP Working Group 11.3 Working Conference on Database Security*. Elsevier North-Holland, Inc., Amsterdam, The Netherlands.

KUHN, D. R. 1997. Mutual exclusion as a means of implementing separation of duty requirements in role-based access control systems. In *Proceedings of the 2nd ACM Workshop on Role-Based Access Control* (Fairfax, VA, Nov. 6-7). ACM Press, New York, NY, 23–30.

LOCHOVSKY, F. H. AND WOO, C. C. 1988. Role-based security in data base management systems. In *Database Security: Status and Prospects* (Annapolis, MD, Oct. 1987), C. E. Landwehr, Ed. North-Holland Publishing Co., Amsterdam, The Netherlands, 209–222.

MOHAMMED, I. AND DILTS, D. 1994. Design for dynamic user-role-based security. *Comput. Secur. 13*, 8, 661–671.

NYANCHAMA, M. 1994. Commercial integrity, roles and object orientation. Ph.D. Dissertation. University of Western Ontario, London, Canada.

NYANCHAMA, M. AND OSBORN, S. 1993. Role-based security, object oriented databases and separation of duty. *SIGMOD Rec. 22*, 4 (Dec. 1993), 45–51.

NYANCHAMA, M. AND OSBORN, S. L. 1994. Access rights administration in role-based security systems. In *Proceedings of the IFIP Working Group 11.3 Working Conference on Database Security*. Elsevier North-Holland, Inc., Amsterdam, The Netherlands.

NYANCHAMA, M. AND OSBORN, S. L 1995. Modeling mandatory access control in role-based security systems. In *Proceedings of the IFIP WG 11.3 Ninth Annual Working Conference on Database Security*, D. Spooner, S. Demurjian, and J. Dobson, Eds. Chapman & Hall, London, UK.

OSBORN, S. 1997. Mandatory access control and role-based access control revisited. In *Proceedings of the 2nd ACM Workshop on Role-Based Access Control* (Fairfax, VA, Nov. 6-7). ACM Press, New York, NY, 31–40.

OSBORN, S., REID, L., AND WESSON, G. 1996. On the interaction between role based access control and relational databases. In *Proceedings of the Tenth Annual IFIP WG 11.3 Working Conference on Database Security* (Aug.), P. Samarati and R. Sandhu, Eds. Chapman & Hall, London, UK.

RABITTI, F., BERTINO, E., KIM, W., AND WOELK, D. 1991. A model of authorization for next-generation database systems. *ACM Trans. Database Syst. 16*, 1 (Mar. 1991), 88–131.

SANDHU, R. 1996. Role hierarchies and constraints for lattice-based access controls. In *Proceedings of the Conference on Computer Security* (ESORICS 96, Rome, Italy), E. Bertino, H. Kurth, G. Martella, and E. Montolivo, Eds. Springer-Verlag, New York, NY, 65–79.

SANDHU, R., COYNE, E., FEINSTEIN, H., AND YOUMAN, C. 1996. Role-based access control models. *Computer 29*, 38–47.

SANDHU, R. S. 1988. Transaction control expressions for separation of duties. In *Proceedings of the 4th Annual Conference on Computer Security Application* (Orlando, FL, Dec.). 282–286.

SIMON, R. AND ZURKO, M. E. 1997. Separation of duty in role based access control environments. In *Proceedings of the 10th IEEE Workshop on Computer Security Foundations* (Rockport, MA, June 10-12). IEEE Computer Society Press, Los Alamitos, CA, 183–194.

THOMAS, R. AND SANDHU, R. 1997. Task-based authorization controls (TBAC): Models for active and enterprise-oriented authorization management. In *Database Security XI: Status and Prospects* (Lake Tahoe, CA), T. Y. Lin and X. Qian, Eds. Chapman & Hall, London, UK, 136–151.

THOMSEN, D. 1991. Role-based application design and enforcement. In *Database Security IV, Status and Prospects*, S. Jajodia and C. Landwehr, Eds. Elsevier North-Holland, Inc., New York, NY, 151–168.

TING, T. 1988. A user-role based data security approach. In *Database Security: Status and Prospects*, C. Landwehr, Ed. Elsevier North-Holland, Inc., New York, NY, 187–208.

TING, T., DEMURJIAN, S., AND HU, M.-Y. 1992. Requirements, capabilities and functionalities of user-role based security for an object-oriented design model. In *Database Security V, Status and Prospects*, C. Landwehr and S. Jajodia, Eds. Elsevier North-Holland, Inc., New York, NY.

VON SOLMS, S. H. AND VAN DER MERVE, I. 1994. The management of computer security profiles using a role-oriented approach. *Comput. Secur. 13*, 8, 673–680.

Received: October 1997; revised: June 1998; accepted: October 1998