The Role Control Center: Features and Case Studies

David F. Ferraiolo
NIST
820 West Diamond Dr.
Gaithersburg, MD 20899
dferraiolo@nist.gov

Gail-Joon Ahn
Univ. of NC at Charlotte
9801 University City Blvd.
Charlotte, NC 28223
gahn@uncc.edu

R.Chandramouli
NIST
820 West Diamond Dr.
Gaithersburg, MD 20899
mouli@nist.gov

Serban I. Gavrila
VDG Inc.
6009 Brookside Dr.
Chevy Chase, MD 20815
serban.gavrila@nist.gov

ABSTRACT

Role-based Access Control (RBAC) models have been implemented not only in self-contained resource management products such as DBMSs and Operating Systems but also in a class of products called Enterprise Security Management Systems (ESMS). ESMS products are used for centralized management of authorizations for resources resident in several heterogeneous systems (called target systems) distributed throughout the enterprise. The RBAC model used in an ESMS is called the Enterprise RBAC model (ERBAC). An ERBAC model can be used to specify not only sophisticated access requirements centrally for resources resident in several target systems, but also administrative data required to map those defined access requirements to the access control structures native to the target platforms. However, the ERBAC model (i.e., the RBAC implementation) supported in many commercial ESMS products has not taken full advantage of policy specification capabilities of RBAC. In this paper we describe an implementation of ESMS called the 'Role Control Center' (RCC) that supports an ERBAC model that includes features such as general role hierarchy, static separation of duty constraints, and an advanced permission review facility (as defined in NIST's proposed RBAC standard). We outline the various modules in the RCC architecture and describe how they collectively provide support for authorization administration tasks at the enterprise and target-system levels.

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; C.2.2 [**Computer-Communication Networks**]: Network Protocols – *applications.*

General Terms

Security, Design, Experimentation.

Keywords

Authorization Management, Role Hierarchy, Administrative Roles, Role Graph, Separation of Duty

SACMAT'03, June 2-3, 2003, Como, Italy.

1. INTRODUCTION

RBAC Models have come of age since they were first proposed. Researchers have extended the basic model to allow for increasingly sophisticated authorization policies. Along with these advancements, the application space for RBAC has also increased. Initially RBAC models were implemented within self-contained resource management products such as Database Management Systems [CS98] and Operating systems [FA99]. The scope of these models is access control for the resources under the control of those individual systems. Recently RBAC models have been used in a class of products called Enterprise Security Management Systems (ESMS). ESMS products are typically used for centralized management of authorizations for resources resident in several heterogeneous systems (called target systems) distributed throughout the enterprise though they may provide other security administration features such as password synchronization, single sign-on, and PKI.

We refer to the class of RBAC model used in an ESMS as the Enterprise RBAC model (ERBAC). An ERBAC model differs from an RBAC model used in a self-contained resource management product in the following ways. First, the objects whose permissions are specified in the ERBAC model are abstract objects as compared to actual system objects such as NT files and their associated permissions may also be abstract as compared to actual permissions such as read, write etc. Secondly, the ERBAC model contains administrative roles in addition to regular roles. The permissions associated with these administrative roles pertain to administrative operations as opposed to resource-level operations like Read, Write etc. Examples of administrative operations are: Assign Users to Roles, Assign a role to role (to build a hierarchy), Assign permission to a role etc. In addition, an ERBAC model includes data structures for mapping the authorization information at the enterprise-level (that is in terms of abstract objects and abstract permissions) to actual objects and permissions on the resources resident in various heterogeneous systems throughout the enterprise in the format required by native access control structures.

The ERBAC model (i.e., the RBAC implementation) supported in many commercial ESMS products [TIV02, BMC02] has not taken full advantage of policy specification and administrative capabilities of RBAC. To demonstrate the virtues of a robust RBAC feature set, we have developed an ESMS called the 'Role Control Center' (RCC). RCC supports an ERBAC model with general role hierarchies, static separation of duty constraints, and an advanced permission review facility (as defined in NIST's proposed RBAC standard [FSGKC01]).

The remainder of the paper is organized as follows. In section 2, we outline the architecture of RCC. In section 3, we provide an overview of RCC features. The next three sections provide a detailed description of each of the features. The ERBAC model definition and visualization features are dealt with in section 4. The enterprise-level administrative operations are the focus of section 5, while section 6 deals with target system-level administrative operations. Section 7 concludes the paper.

2. RCC ARCHITECTURE

RCC is a three-tiered application (viz., Fig. 1) that is made up of the following tiers: Presentation Layer, Application Logic Layer and Data Layer. The presentation layer is the RCC client, the application logic layer consists of the RCC server and RCC agents (resident in various target systems) and the data layer consists of the data repository. The RCC client is an application that uses the interface exposed by the RCC server. The standard RCC client provides the graphical user interface (see figure 2 for an example), the code responsible for displaying the ERBAC model graph (referred to as the role graph in the rest of the paper), capturing user actions and transforming them into calls to the APIs provided by the RCC server. The RCC client performs a function similar to a browser in a web application. The RCC client is however more sophisticated than a web browser. While a web browser requests services from a web server by sending just the URL string, the RCC client can send different types of commands that are available in the interface exposed by the RCC server. The RCC client communicates with the RCC Server and its agents using Secure Sockets Layer (SSL) protocol.

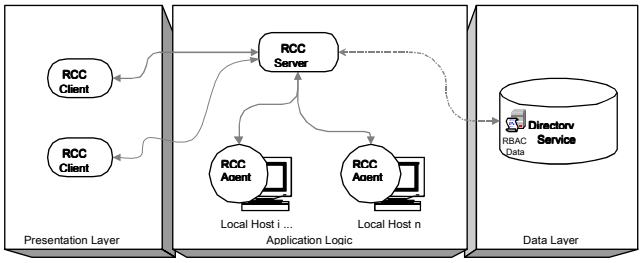

Figure 1 RCC Architecture

The RCC server (1) analyzes the commands received from the RCC client (2) retrieves the necessary data from the data layer and checks the commands' consistency with the data; (3) updates the RBAC sets of users and roles, and the user-role, role-role, role-permission relations according to the commands, and (4) sends the updated data back to the data layer. The RCC server is also responsible for mapping selected subgraphs of the role graph (called views) to user accounts and groups on heterogeneous hosts (called also target systems), and for mapping abstract objects and role permissions to actual objects and permission structures (e.g., ACLs) on those hosts. For these tasks, RCC uses agent software running on each host to create/delete groups and user accounts, populate the groups with user accounts, and set up ACLs, according to commands received from the RCC server. The RCC Server is currently implemented as a Windows NT/2000 application and there are now two versions – one written in Visual Basic and the other in Java.

The data layer consists of a directory service, which stores, retrieves, and protects the actual ERBAC data, i.e., the user and role sets, the various relations, the abstract objects, and the mappings between ERBAC data and target system data.

3. OVERVIEW OF RCC FEATURES

As an ESMS, RCC provides features for two levels of administrative operations – Enterprise-level administrative operations and Target system-level administrative operations. RCC server through the RCC client provides the features for enterprise-level administrative operations:

- Create and maintain the ERBAC model elements and relations– users, roles, permissions, constraints, user-role relations, role-role relations and role-permission relations. The role permissions are defined in terms of abstract objects/abstract operations. Their mappings to actual objects/actual operations on the various target systems are also defined. The roles include regular roles (defined in any RBAC model –e.g., payroll clerk) as well as administrative roles (e.g., financial_admin_role).

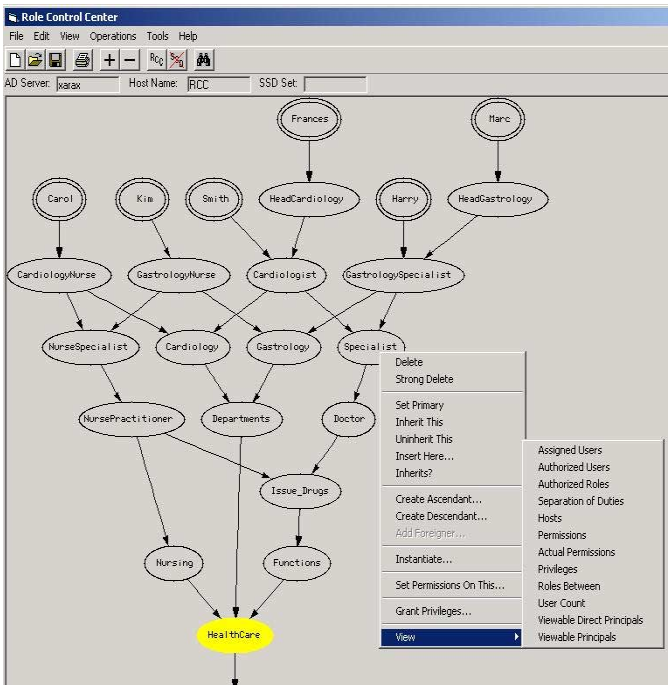

Figure 2 RCC Administrative Interface

- Visualization of various ERBAC model relations referred above.

The features for target-system-level operations include:

- Map Enterprise users/roles (defined in the ERBAC model) to user accounts/groups in target systems.
- Map abstract objects/operations in the ERBAC model to actual objects (e.g., NT file) and actual operations (e.g., Read, Write etc.) in target systems.

All the operations are carried out using menus in RCC's administrative interface. The role graph displayed in the interface uses single ellipses to represent roles and double ellipses to represent users. A role called 'rbac' is pre-defined in the graph to act as the root node for building the entire ERBAC model for the enterprise. There are two types of menus in the RCC interface. They are: (a) The menus that are associated with the objects in the

ERBAC model graph and (b) the top-level menus. The object-level menus support operations for creating/deleting roles, users and role-role relations (i.e., role hierarchies) as well as review of role permissions (both administrative permissions for administrative roles and resource permissions for regular roles). The top-level menus support static separation of duty specifications, definition of abstract objects/abstract permissions, visualization of role subgraph instantiated on a particular host, mappings from abstract objects/abstract permissions to actual objects/actual permissions on a designated host as well as assignment of resource permissions and administrative permissions to regular and administrative roles respectively.

4. RBAC MODEL FEATURES IN RCC

As we already stated, the distinguishing feature of RBAC is the support for policy specification capabilities through support for general hierarchies and static separation of duty constraints in its ERBAC model. In sections 4.3 and 4.4, we describe the implementation of these features in RCC and the visualization of these specifications through the RCC administrative interface. In order to perform visualization of specifications we need to have a quick review of navigational features available for the ERBAC model graph (role graph) provided by the RCC administrative interface in the RCC client. This is the focus of the next two sections.

4.1 Role Graph Navigation

A principal goal of RCC is to provide an easy and satisfactory way of navigating through a role graph containing a large number of roles, without imposing restrictions on the inheritance relation. We provide a simple schema that is very similar to the navigation through a directory structure (which is a tree) so that RCC can be familiar to users of Window-based systems.

For any node *a* of the graph, we call the subset of roles consisting of the node *a* and its direct ascendants (direct descendants), partially ordered by the inheritance relation →, the *upward (or downward) projection with anchor a* (see Section 4.3).

Figures 4 and 5 show upward and downward projections extracted from the graph in Figure 3. We can generalize the projection by introducing the *n*-tiered upward or downward projection:

$$up\_proj_n(a) = (\{a\} \cup \{r \mid r \rightarrow^n a\}, \rightarrow)$$

$$down\_proj_n(a) = (\{a\} \cup \{r \mid a \rightarrow^n r\}, \rightarrow)$$

where x $\rightarrow^n$ y means that x inherits y through at most *n* direct inheritances. Obviously, $up\_proj(a) = up\_proj_1(a)$, $down\_proj(a) = down\_proj_1(a)$, and the ∞-tiered projection is the entire role graph. In general, an *n*-tiered projection with *n*=2 is not a tree, while the 1-tiered projections are trees.

RCC uses a navigation scheme in which, at any moment, one of the graph nodes is selected as the *anchor* and is marked distinctively. The graphical user interface displays both the *n*-tiered upward and downward projections of the current anchor, where *n* is the currently chosen number of tiers. For example, Figure 6 shows the portion of the graph in Figure 3 displayed by RCC when the anchor is *PayrollSuper* and n=2.

To change the anchor, the user selects one of the anchor's descendants or ascendants. That node becomes the new anchor, and RCC displays the projections corresponding to the new anchor.

Two questions arise related to our navigation schema. The first is how to select the initial anchor? The second problem arises when the role graph is not connected: how does one navigate from one connected component of the graph to another?

A simple solution to both questions is solved by introducing an artificial graph node, called *the base role*, as the smallest element of the role set under the partial order →. The base role has the same semantics as the *minrole* as described in [NO99]. The graph becomes connected, and the base role can be used as the initial anchor. In addition, the base role can help simplify the problem of merging the role graphs corresponding to roles of two or more merging organizations. In Figure 3, the role *rbac* is the base role. The remainder of this document will use the name *rbac* to designate the base role.

Implementing the base role as smallest element of the role set requires the following actions. When a user or role is created without specifying whose ascendant it is, the new user or role is made a direct ascendant of *rbac*. When a role *r* is set to inherit another role *r'*, the direct inheritance *r→rbac* is deleted if it exists. When an inheritance *r→r'* is deleted and consequently role *r* has no more direct descendants, then the direct inheritance *r→rbac* is established.

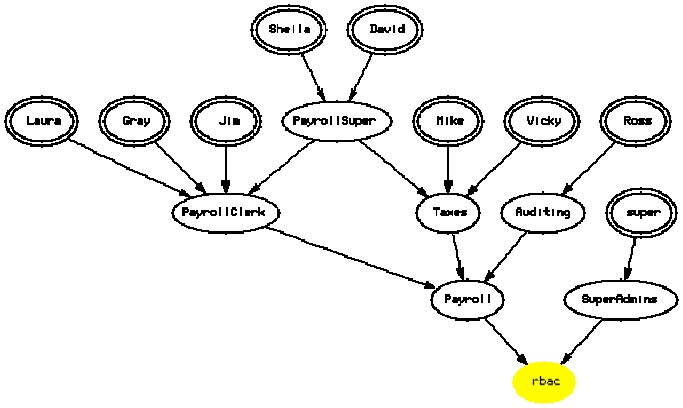

Figure 3 A role graph

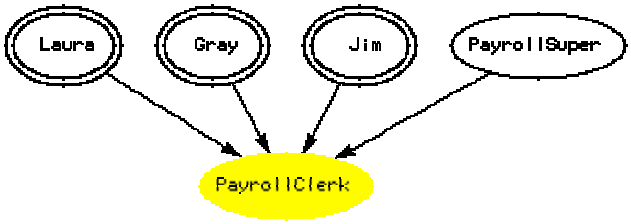

Figure 4 An upward projection (with anchor PayrollClerk)

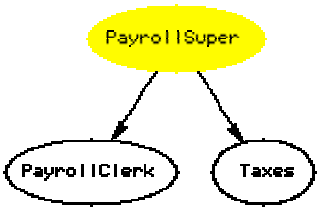

Figure 5 A downward projection (with anchor PayrollSuper)

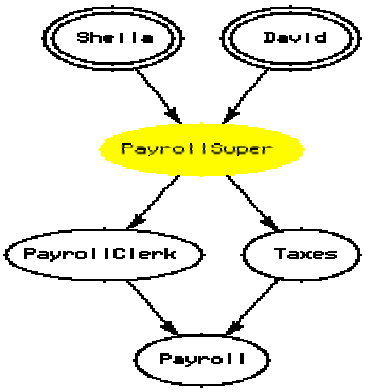

Figure 6 – Subgraph with anchor PayrollSuper and n=2

4.2 Role Views

RCC uses role views to define and instantiate sets of relevant users and roles on a target system, and to delegate administrative privileges for certain portions of the role graph to administrative roles (discussed in detail in section 6).

A *role view* defined by a set of roles $\{r_1,\ldots,r_n\}$is a sub-graph of the of the overall role graph with the following properties:

1. the view contains $r_1,\ldots,r_n$ as nodes;
2. if the view contains a role r, then it contains any user or role q such that $q \rightarrow r$;
3. the view contains no other nodes except those included by rules 1, 2;
4. the view contains an arc $q \rightarrow r$ iff q and r are included in the view and $q \rightarrow r$ is an arc in the original graph.

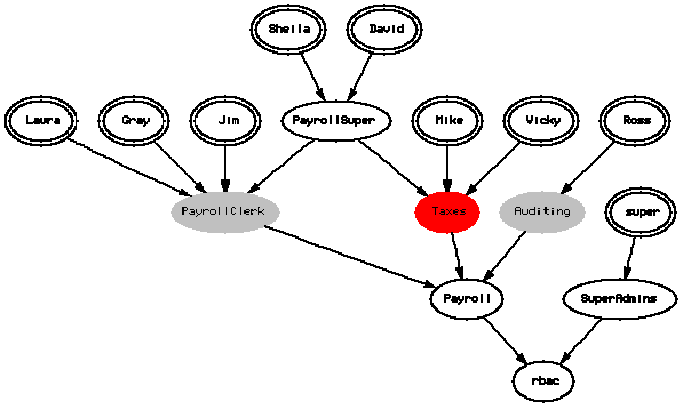

Figure 7 – Defining a role view

As a consequence, to define a view it suffices to indicate the "most general" roles contained in that view, which are called *principal nodes* of the view. If the view obtained from those principals applying the above properties is not a connected graph, then a base role is added when displaying and navigating the view. For example, (see Figure 7), roles *PayrollClerk* and *Auditing* define the view shown in Figure 8. Note that the sub-graph is built from the principals, their direct and indirect ascendants, and their inheritance relationships, and is augmented with the base node *rbac* as a direct descendant of the principals *PayrollClerk* and *Auditing*. Views may overlap, for example *PayrollSuper*, *Sheila*, and *David* are common to the views defined by the principals {*PayrollClerk*, *Auditing*} and {*Taxes*}.

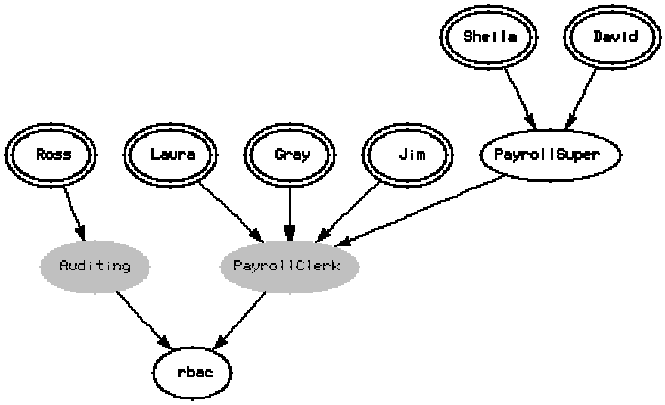

Figure 8 – A view displayed by RCC

4.3 Inheritance Structures

The role graph [FSGKC01, NO99] describes the hierarchical structure of roles in the enterprise. As a major component of an RBAC system, role hierarchies go beyond flat role structures, in their ability to depict and manage user privileges. Simply by virtue of a role's relative position in a role hierarchy, the permissions that are assigned to the role may be made to contain, or be contained by other roles in the hierarchy. By creating a suitable role hierarchy, administrators are better able to formulate access policies in terms of organization-specific functions and business structures.

The primary role-role relation that results in the hierarchy is the "containment" or "inheritance" relation. The inheritance relation between two roles r_1 and r_2 when represented as $r_1 \rightarrow r_2$ means that role r_1 inherits or contains role r_2. The inheritance property is with respect to privileges/permissions while the containment refers to membership. Hence $r_1 \rightarrow r_2$ if and only if all privileges of r_2 are also privileges of r_1, and all users of r_1 are also users of r_2. The role graph nodes represent users and roles, and the arcs represent the relation $\rightarrow$.

The combined property of privilege inheritance and membership containment is realized in RCC using the following implementation technique. In this technique permissions are not directly assigned to roles. Permissions are assigned to groups and groups are mapped to roles that are organized into a role hierarchy. Groups assigned to a role are included in all the roles higher up in the hierarchy. Hence if $r_1 \rightarrow r_2$, a user assigned to role r_1 becomes a member in all the groups mapped to role r_2 and by implication a member of role r_2 as well. Also since groups are

bundles of permissions, permission inheritance from role r_2 also occurs.

In practice role hierarchies are of two types – general role hierarchies and limited role hierarchies. As implemented in RCC, general role hierarchies provide support for an arbitrary partial order to serve as the role hierarchy, to include the concept of multiple inheritances of permissions and user membership among roles. In particular, general role hierarchies allow a role to have more than one immediate ascendant (potentially inheriting user membership from multiple sources) and at the same time one or more immediate descendents (potentially inheriting permissions from multiple sources). Limited role hierarchies impose restrictions resulting in a simpler tree structure (e.g., a role may have one or more immediate ascendants, but is restricted to a single immediate descendent, or visa versa).

As demonstrated by RCC, general hierarchies provide greater flexibility in defining roles and allows for greater visibility and understanding of the distribution and composition of permissions among users and roles. By taking advantages of the properties of a general hierarchy, RCC can apply an object-oriented approach at organize and manage users and roles. For instance the permissions that are authorized for a role can be decomposed into lower-level roles representing the positions, functions: such as duties, tasks, and entitlements, or organizational units: such as divisions, departments, groups, and teams. Once these lower-level roles are created, they may be re-used in the creation of one or more higher-level roles.

The graph depicted in Figure 9 illustrates several properties of a general hierarchy in terms of a hospital application. Because general hierarchies place no restrictions on the number of immediate role inheritance relations, the Cardiologist role is able to inherit permissions from both the functional roles beginning with Specialist and the organizational roles beginning with Cardiology, As such, the Cardiologist role is neither a functional nor an organizational role, but rather a hybrid.

We denote by $\rightarrow^*$ the reflexive-transitive closure of the inheritance relation, i.e., $r_1 \rightarrow^* r_2$ iff $r_1 = q_1 \rightarrow \ldots \rightarrow q_n = r_2$, where $n>0$ (note that the definition allows for roles r_1 and r_2 to coincide). RCC requires the inheritance relation $\rightarrow^*$ to be a partial order on the set of RBAC users and roles. Consequently, the role graph is a directed acyclic graph. Usually, we represent the graph with the arcs corresponding to the inheritance relation $\rightarrow$ oriented top-down. Thus, we can say the role membership is inherited top-down, and the role privileges are inherited bottom-up.

Another interesting property of a general hierarchy is its uniform treatment of a user/role assignment (user-role relation) and the immediate role inheritance relation. By virtue of a user's assignment to a role, the user inherits the permissions assigned to the role, and the user becomes a member of the role. Also, because a user may be assigned to multiple roles, a general role hierarchy allows for the representation of all user role assignments, without resorting to multiple instances of the user. For example, Figure 9 shows that user Smith is assigned to two roles – Consultant and Cardiologist. By virtue of the role hierarchy that exists under those assigned roles, Smith becomes authorized for other roles such as Development, Specialist, Doctor, Issue_drugs etc. To graphically illustrate the same relations within a limited role hierarchy would require two instances of user Smith.

The above discussion implies that the representation of inheritance relation $\rightarrow$ and its closure are used for representing user-role assignments and authorization respectively. Hence a user u is said to be *assigned* to role r if $u \rightarrow r$, while u is said to be *authorized* for role r if $u \rightarrow^+ r$, where $\rightarrow^+$ is the transitive closure of the $\rightarrow$ relation.

For example, in the role graph of Figure 10, user Kim is assigned to role *GastologyNurse*, and is authorized to roles *GastologyNurse*, *NurseSpeialist*, *NursePractitioner Issue_Drugs*, *Functions* and *HealthCare*. Also, the privileges available to Kim are those that are assigned to Kim plus those that are assigned to Kim's authorized roles.

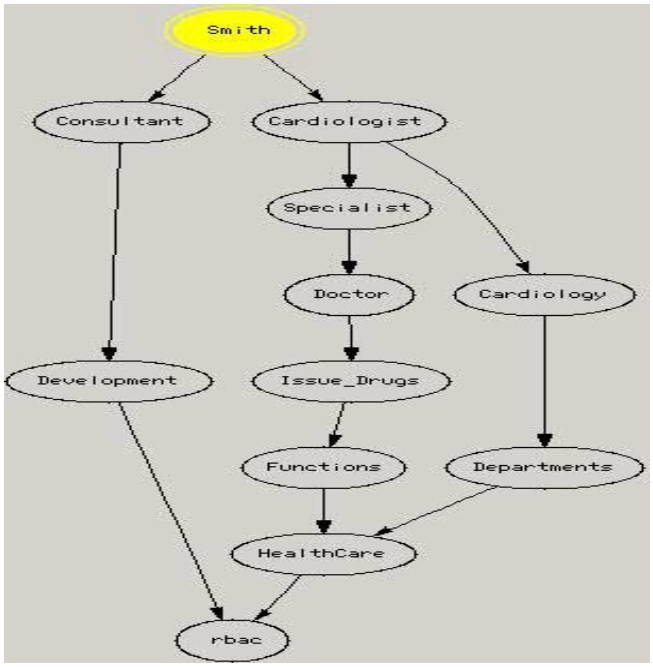

Figure 9 User Smith's authorized roles

The semantics of the inheritance relation and the uniformity in the representation of role-role and user-role relation greatly facilitates permission review. Since a role can represent either a function, department or organizational position, consider the case where a role represents a function (e.g., Issue_Drugs). By generating a graph view with Issue_Drugs as the anchor (Figure 10), it is possible to view all the roles and all the users who have been authorized to issue drugs.

4.4 Static Separation of Duty Constraints

The next policy specification capability of RCC pertains to Static Separation of duty (SSoD) constraints that is an example of a conflict of interest rule as they uniquely apply to an organization [SZ97, GGF98, AS00, JT01]. A SSoD constraint specifies that a user may be authorized as a member of a role only if that role is not designated as mutually exclusive with any of the other roles for which the user already is authorized.

A common example of a set of mutually exclusive roles is the purchasing manager and accounts payable manager roles. Generally, the same individual is not permitted to belong to both roles because this creates a possibility for committing fraud by an individual with combined permissions for purchasing and approving a payment.

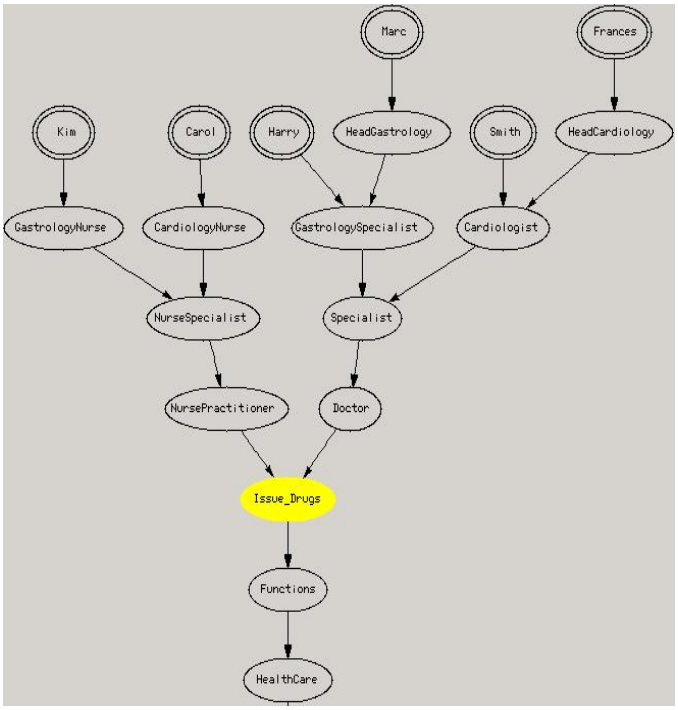

Figure 10 The roles/users that are authorized to Issue Drugs

Consider role graph in Figure 3, we can identify several conflicting roles. For example, the role *Auditing* should be mutually exclusive with *Taxes* and *PayrollClerk*. This means that no user may be authorized to *Auditing* and *PayrollClerk*, or *Auditing* and *Taxes*, or *Auditing* and *PayrollSuper*.

In order to support this Static SoD (SSoD), we first define the name of conflicting role set and determine conflicting role sets. Figure 5 shows a "Static Separation of Duties" dialog box illustrating a conflicting role set and conflicting roles selected. Suppose a new user *Ross* is assigned to the role *Auditing*. In the case of SSoD on these two roles *Auditing* and *PayrollClerk*, *Ross* should not be allowed to be a member of the role *PayrollClerk*. Current environment for the user *Ross* is summarized as Table 2.

From the above table, we can say that the user *Ross* can have privileges assigned to roles *Auditing* and *Payroll* but not on *PayrollClerk*. This constraint is imposed by the SSoD set (named Payroll_Auditing) as shown in Figure 11. RCC represents such a conflicting relationship with colored nodes in a role graph. The system enforces such constraints whenever a new user-role assignment (an instance of user-role relation) is initiated. For example, a supervisor may want to assign the user *Ross* to the role *PayrollClerk* or *PayrollSuper*. Such an assignment is not allowed by RCC because *Ross* is a member of the role *Auditing*, which is in conflict with the role *PayrollClerk*.[1] This indicates that any user, who is a member of one of roles in a conflicting role set either implicitly or explicitly, cannot be assigned to any other roles in the same conflicting role set.

In role-based systems, two conflicting permissions can be assigned to a role. An SSoD constraint may simply require that no user can be assigned to such a role or any role senior to it, which makes that role quite useless as identified in [AS99].

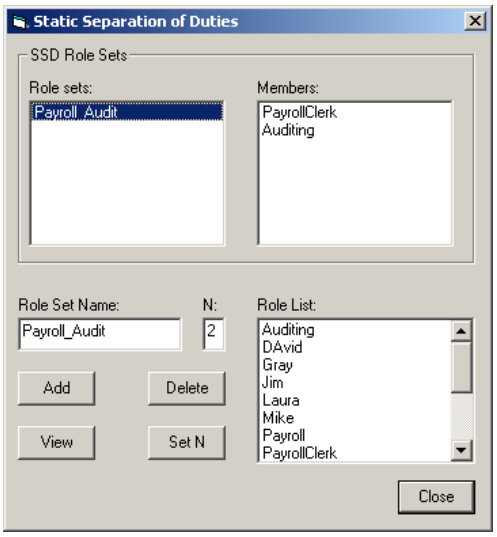

Figure 11 SSoD Set: Payroll_Audit

However, RCC tool can prevent certain kinds of mistakes in role-permissions and user-roles assignments. The RCC tool checks for the SSoD constraints, if any, before performing any assignment task. Other forms of mutual exclusivity, as static operational separation of duties, or static object-based separation of duties, can be enforced by RCC but are not implemented at this time.

The SSoD constraints are restrictions affecting creation of user-role relations. Restrictions governing creation of other relations such as role-permission relations are also possible in RCC thus opening up the potential for supporting a richer set of policies.

5. ENTERPRISE-LEVEL ADMINISTRATIVE OPERATIONS

Enterprise-level administrative operations are tasks involved in the creation and maintenance of the ERBAC model. The permissions to perform these administrative operations are called administrative privileges. The logic for assignment of these administrative privileges in turn is specified through an administrative model. One such administrative model is the URA97 model [SB97]. In this section, we demonstrate how URA97 features can be supported by RCC.

The main goal of URA97 model is to impose restrictions on which users can be added to a role by whom, as well as to provide different semantics for user assignment to roles and for user revocation from roles.

The URA97 model controls the assignment of user to roles by means of the relation "*can_assign*". The meaning of *can-assign* (x, y, $\{a,b,c\}$) is that a member of the administrative role x (or a member of an administrative role that is senior to x) can assign a user whose current membership, or non-membership, in regular roles satisfies the prerequisite condition y to be a member of regular roles a, b or c.

To simulate the notion of *can_assign* relationship in URA97 model, we first construct the role hierarchy as shown in Figure 12. Figure 12 shows two role hierarchies: one for regular roles and the other for administrative roles. The administrative role

[1] In addition, RCC displays an error message such as "Inheritance would contradict SSoD set."

hierarchy has the senior most role *Sec_Off_Senior* which inherits a junior administrative role *Sec_Off_Junior*. Suppose the *Sec_Off_Junior* role has a partial administrative control over the roles {*PayrollClerk, Taxes*} but not over *PayrollSuper* in the hierarchy. In the regular role hierarchy, a user *Alice* is a member

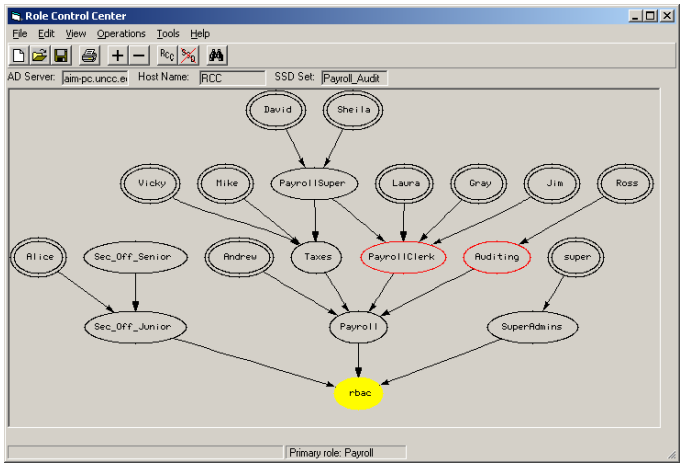

Figure 12 Role Hierarchy Example

of the *Sec_Off_Junior* role and *Andrew* is a member of the role *Payroll*. Therefore, *Alice* can assign *Andrew* to any roles between {*PayrollClerk*, *Taxes*} but not to *PayrollSuper*. In summary, *Alice* can assign any user to the roles in {*PayrollClerk*, *Taxes*} if and only if the user is a member of the *Payroll* role. This *can_assign* relation can be specified as shown in Table 1(a)[2].

Next, the aforementioned scenario requires that RCC assign the *Sec_Off_Junior* role as an administrative role for the set of roles {*PayrollClerk*, *Taxes*}. It allows the *Sec_Off_Junior* role to partially perform administrative control over those roles.

This can be done by assigning administrative permissions to the *Sec_Off_Junior* role. Figure 13 administrative permissions of the *Sec_Off_Junior* role over the roles {*PayrollClerk*, *Taxes*}. At the moment, RCC does not provide any means of specifying the notions of prerequisite condition and role range. Corresponding *can_assign* relation of RCC to Table 1(a) can be represented as shown in Table 1(b).

Another important feature in URA97 includes "weak" and "strong" revocations. RCC allows system administrators to revoke users from roles depending upon their inheritance structure. A user or role can be deleted only if it inherits the base role *rbac*. It supports a notion of weak revocation in URA97. However, RCC's weak revocation cannot delete any roles (or users) if those roles (or users) inherits any other user(s) or role(s) in role hierarchy. Such users (or roles) can be deleted only by strong revocation. RCC can support those revocation mechanisms. However, RCC does not have a way to specify *can_revoke* relation at the moment. The relation specification (or configuration) needs to be considered in RCC to fully support URA97.

[2] The Role Range specifies the *Sec_Off_Junior* role has an administrative control over the roles between *Payroll* and *PayrollSuper*.

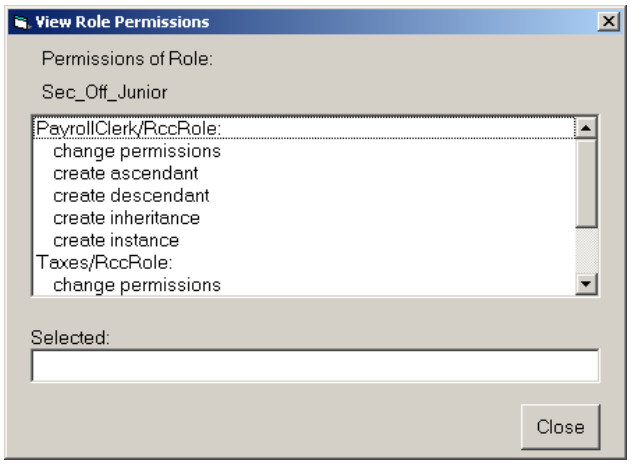

Figure 13 Representation of Administrative Permissions

Admin role	Prerequisite	Role set	Role range
Sec_off_junior	Payroll	{PayrollClerk, Taxes}	{Payroll, PayrollSuper}

(a) URA 97

Admin role	Prerequisite	Role set	Role range
Sec_off_junior	none	{PayrollClerk, Taxes}	none

(b) RCC

Table 1. *can_assign* relation

User	Assigned role	Authorized role	Conflicting role
Ross	Auditing	Auditing, Payroll	PayrollClerk

Table 2. Separation of Duty Example

5.1 Delegation of Administrative Privileges

The number of roles and role relationships within a large enterprise can become overwhelming for a single administrator to maintain. In addition, administrators who are closer to the day-to-day operations of a specific organizational entity are typically better suited to administer the roles and role relationships with respect to that organizational entity. To deal with these administrative issues, RCC supports the delegation of administrative privileges, i.e., assignment of access rights necessary to modify the RCC database of roles and role relationships from a senior administrator to a subordinate administrator. Furthermore, it is often desirable to impose policy constraints across administrative boundaries.

This process is best described by an example. Assume that the enterprise role graph is much more complex than that of Figure 3, but still contains the view with the principal *Payroll*. The *super* user, which may perform all RCC operations on all users and

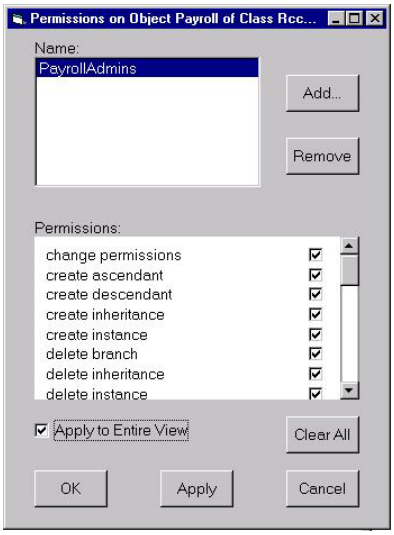

Figure 14 Granting Permissions to PayrollAdmins

roles, may want to delegate the administration of the *Payroll* view to a user *Ronald*. To that purpose, *super* may create a new, administrative, role, called *PayrollAdmins*, grant it all permissions on the roles included in the *Payroll* view (see Figure 12), and assign the user *Ronald* to *PayrollAdmins*. *Ronald* will be able to perform all RCC operations, but only on roles included in the *Payroll* view. Note that RCC user interface allows for defining permissions on all roles in a view by checking the "Apply to Entire View" box. Figure 15 shows per-user review of permissions for *Ronald*. The small arrow in front of permissions indicates the permission is inherited from an assigned role, and not directly granted to *Ronald*.

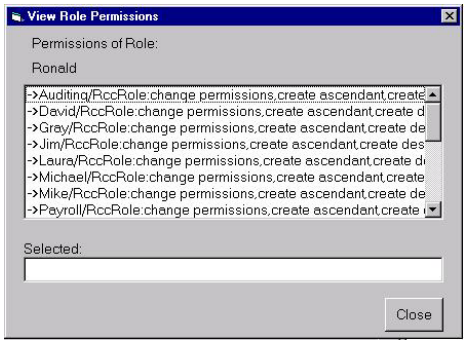

Figure 15 Ronald's permissions

In turn, *Ronald* may want to delegate the administration (or at least some administrative duties) of the *Taxes* view to a third administrator, *Michael*. To this purpose, Ronald may create a new administrative role *TaxAdmins* (but as an ascendant of *Payroll* – remember that *Ronald* has no access to roles outside the *Payroll* view), grant *TaxAdmins* some administrative permissions to the *Taxes* view, and assign *Michael* to *TaxAdmins*. Ronald is allowed to grant *TaxAdmins* permissions on roles in the Taxes view if and only if *super* has granted him (through the PayrollTaxes role) the permission *change permissions* on those roles.

6. TARGET SYSTEM-LEVEL ADMINISTRATIVE OPERATIONS

RCC is intended to help administrators in setting up access control information based on a user's function in an enterprise. To that aim, RCC uses the notions of RBAC users, roles, abstract objects and permissions at the abstract, central level of the enterprise. However, most existing operating systems and applications perform the actual access checks by using actual user accounts, groups, and ACLs on host computers, and do not recognize roles. RCC performs and automatically maintains a mapping between roles, users, and abstract permissions on one hand, and groups, user accounts and actual permissions on the target systems it controls, on the other hand.

Mapping users, roles, and membership to user accounts and groups is based on views. The RCC administrator has to *select a view relevant for a given host*, then map (or instantiate) that view to user accounts and groups of the given host by using RCC. The instantiation process is simple; the only rule being that the entire view has to be mapped. For each user in the view, RCC creates (through the agent running on that host) a user account on that host. For each role in the view, RCC creates a group on that host. Then RCC populates the new groups with user accounts according to the role membership defined by the role graph.

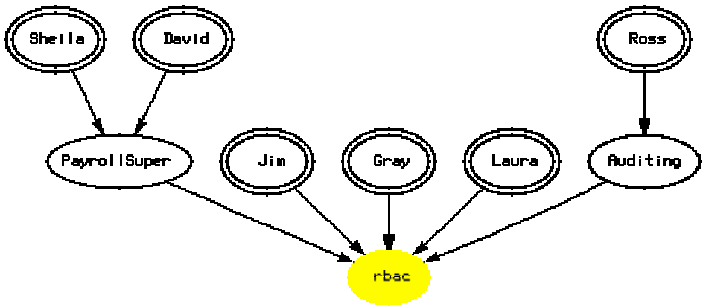

Figure 16 Instantiated PayrollClerk's instance view for the host Pear

For example, assume that the view of Figure 8 represents the users and roles relevant for a host computer called *pear*. By mapping the view on *pear*, RCC creates user accounts for *Ross*, *Laura*, *Gray*, *Jim*, *Sheila*, and *David*, groups for *Auditing*, *PayrollClerk*, and *PayrollSuper*, and populates these groups as follows:

- group *PayrollSuper* has *Sheila*, *David* as members;
- group *PayrollClerk* has *Laura*, *Gray*, *Jim*, *Sheila*, *David* as members;
- group *Auditing* has *Ross* as member.

The following lines show part of the commands sent to the RCC agent running on *pear*, used to accomplish the instantiation task:

```
connect
adduser&user=Ross&password=Ross_password
addglobalgroup&group=Auditing
addusertogroup&user=Ross&group=Auditing
```

On hosts running Windows NT™ operating system, the RCC administrator can choose to create local or global groups,

depending on the host's role in the NT domain. A version of RCC uses an instantiation algorithm that takes advantage of NT's group nesting mechanism to avoid unnecessary duplication of user membership.

It is worth noting that the so-called hosts do not need to be computers on a network; they can be any system that uses users and groups (and ACLs) as a security mechanism (for example, an Apache server).

RCC also allows the administrator to delete instances of users and roles from given hosts, with the requirement that at any moment the instantiated users/roles form a view. For example, if the view in Figure 8 were instantiated on a host, it would be possible to delete the instances of *PayrollClerk* or *Auditing*, but not that of *PayrollSuper* or *Ross*. After deleting the *PayrollClerk* instance, the view instantiated on that host would be that of Figure 16.

7. CONCLUSION

RCC is a tool that enables centralized management of authorizations for resources distributed throughout an enterprise. For this purpose RCC provides an interface through its Client to create and maintain an enterprise-level Role-based Access control Model (ERBAC) on the RCC server and map the authorization data contained in the ERBAC model to the various target systems through the resident RCC Agent module. The ERBAC model in RCC provides support for defining arbitrary role structures (generalized role hierarchies) and flexible separation of duty constraints. The present version of RCC contains agents for Windows NT and Apache Web Server platforms. Work is underway to develop agents for more target platforms and to enhance the policy support capabilities of the ERBAC model.

References

[AS99] Gail-J. Ahn and Ravi Sandhu. The RSL99 language for role-based separation of duty constraints. In Proceedings of 4th ACM Workshop on Role-Based Access Control, pages 43{54. ACM, 1999.

[AS00] Gail-J. Ahn, Ravi Sandhu, Role-based authorization constraints specification, ACM Transactions on Information and Systems Security 3 (4) (2000).

[BMC02] Enterprise Security Station – User Guide (Windows GUI) – BMC Software Inc., 2002.

[CS98] R. Chandramouli, R. Sandhu, "Role Based Access Control Features in Commercial Database Management Systems", 21st National Information Systems Security Conference, October, 1998. Crystal City, Virginia.

[FA99] G. Faden, "RBAC in UNIX Administration", 4th ACM workshop on Role-based Access Control, Fairfax, VA, USA, 1999.

[FSGKC01] D. Ferraiolo, R. Sandhu, S. Gavrila, R. Kuhn, R. Chandramouli, Proposed NIST standard for role-based access control, ACM Transactions on Information and Systems Security 4 (3) (2001).

[GGF98] V.D. Gligor, S.I. Gavrila, D.F. Ferraiolo: "On the Formal Definition of Separation-of-Duty Policies and their Composition," Proc. 1998 Symposium on Security and Privacy, May 1998, Oakland, California.

[JT01] Trent Jaeger and Jonathon Tidswell, "Practical Safety in Flexible Access Control Models." ACM Transactions on Information and Systems Security, Volume 4, Number 3, August 2001.

[NO99] M.Nyanchama, S. Osborn: "The Graph Model and Conflict of Interest," ACM Transactions on Information and System Security, 2(1), February 1999.

[SB97] R. Sandhu, V. Bhamidipati, The URA97 model for role-based administration of user-role assignment, in: T.Y. Lin, X. Qian (Eds.), Database Security XI: Status and Prospects, North-Holland, Amsterdam, 1997.

[SZ97] R.T. Simon, M.E. Zurko: Separation of Duty in Role-Based Environments, in Proc. Computer Security Foundations Workshop X, Rockport, Massachusetts, June 1997.

[TIV02] Enterprise Security Architecture using IBM Tivoli Security Solutions (2002) – IBM Corporation