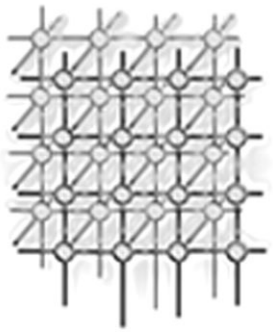
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2004; **16**:1121–1141 (DOI: 10.1002/cpe.807)

A role-based infrastructure management system: design and implementation



Dongwan Shin[1,*,†], Gail-Joon Ahn[1], Sangrae Cho[2] and Seunghun Jin[2]

[1]*Department of Software and Information Systems, University of North Carolina at Charlotte, Charlotte, NC 28223, U.S.A.*
[2]*Department of Information Security System, Electronics and Telecommunications Research Institute, Taejon, 305-350, South Korea*

SUMMARY

Over the last decade there has been a tremendous advance in the theory and practice of role-based access control (RBAC). One of the most significant aspects of RBAC can be viewed from its management of permissions on the basis of roles rather than individual users. Consequently, it reduces administrative costs and potential errors. The management of roles in various RBAC implementations, however, tends to be conducted on an *ad hoc* basis, closely coupled with a certain context of system environments. This paper discusses the development of a system whose purpose is to help manage a valid set of roles with assigned users and permissions for role-based authorization infrastructures. We have designed and implemented the system, called *RolePartner*. This system enables role administrators to build and configure various components of a RBAC model so as to embody organizational access control policies which can be separated from different enforcement mechanisms. Hence the system helps make it possible to lay a foundation for role-based authorization infrastructures. Three methodological constituents are introduced for our purposes, together with the design and implementation issues. The system has a role-centric view for easily managing constrained and hierarchical roles as well as assigned users and permissions. An LDAP-accessible directory service was used for a role database. We show that the system can be seamlessly integrated with an existing privilege-based authorization infrastructure. Copyright © 2004 John Wiley & Sons, Ltd.

KEY WORDS: role-based access control; role management; role engineering; role administration; authorization infrastructure

*Correspondence to: Dongwan Shin, Department of Software and Information Systems, University of North Carolina at Charlotte, Charlotte, NC 28223, U.S.A.
†E-mail: doshin@uncc.edu

Contract/grant sponsor: Electronics and Telecommunications Research Institute
Contract/grant sponsor: National Science Foundation; contract/grant number: IIS-0242393

Received March 2003
Revised June 2003
Accepted June 2003

INTRODUCTION

Pervasion of interest in using roles in access control has been quite salient in computer security communities over the last few years. Thus many formalized and practical approaches to recognizing the value of their usage in access control have been taken by both researchers and practitioners. The reference models have been proposed for role-based access control (RBAC) in [1,2], and the efforts to extend the models, for example in the area of constraint specification, role administration, and role delegation, have followed in [3–5]. With those formal models as its quintessence, RBAC has grown to be a proven solution for managing access control in a simple, flexible, and convenient manner. In RBAC, user authorization depends upon the roles of which a user is a member, and permissions are assigned to the roles. This greatly simplifies management of permissions, reducing complexity and potential errors in directly assigning permissions to users. Also, RBAC can be easily reconfigured to comply with different organizational access control policies such as least privilege, separation of duties and abstract operations. This flexibility is beneficial to organizations that need to modify their access control policies for their needs.

RBAC has been playing the role of a key component in the design of user authorization services for a variety of systems or applications. For instance, it is used in a simple Web-based application in order to provide users with different grades of services, or it is deployed in an entire operating system as an alternative to the *all-or-nothing* superuser model. In addition to those individual systems or applications, enterprise-wise large-scale systems also fall in the sphere of RBAC's influence. Enterprise users generally need to access multi-vendor and multi-layered applications and systems of which enterprise resources consist. Schemes like juxtaposing *enterprise-wise roles* with localized roles, which are effective only in the individual applications or systems, have been identified with the purpose of fertilizing RBAC's applicability into the enterprise-wise authorization boundary. In those schemes, simply put, enterprise users are authorized according to their membership of *enterprise-wise roles* so as to access multi-vendor and multi-layered enterprise-wise applications and systems.

However, the management of roles in a variety of implementations of RBAC tends to be conducted on an *ad hoc* basis, specifically with a certain context of system environments in mind. This often results from the lack of an approach to systematically administering roles or other RBAC components for role-based authorization infrastructures. Such an approach requires a sound understanding of RBAC components and organizational access control policies, and then an exhaustive investigation of how to configure those RBAC components to realize those policies within multi-vendor and complex systems or applications. In order for role administrators to carry out the approach, software tools that enable such systematic role management are required, and these tools should be built on the strong formal foundation of the reference models of RBAC [1,2,4].

In this paper we describe a role-based infrastructure management system, called *RolePartner*. The main purpose of the system is to help role administrators establish a valid set of roles and role hierarchies with assigned users and associated permissions. By valid we mean that roles and role hierarchies must be constrained according to organizational access control policies and have properly designed meanings within an organization's RBAC environment. The role administrator can define, build, and manage various components of a RBAC reference model (*RBAC96*) [1], thereby making it possible to lay a foundation for a role-based authorization infrastructure. We present how we designed and implemented *RolePartner*, with the introduction of our methodological approach to identifying the necessary constituents for *RolePartner* design, together with the general requirements of such a system.

Along with basic RBAC components such as user/role and permission/role assignment, *RolePartner* supports the advanced notion of role hierarchies and constraints. *RolePartner* also has a role-centric view for easily managing permissions and users, and it leverages an LDAP-accessible directory service for storing role-based authorization organizational policies composed of the configured RBAC components. We also show that the system can be seamlessly integrated with an existing privilege-based authorization infrastructure.

ROLES IN ROLE-BASED AUTHORIZATION INFRASTRUCTURES

While the meanings of roles are different, depending upon the context of their usage, they generally represent a set of competency and responsibility pairs [6]. In the reference models of RBAC [1,2], they describe the relationship between users and permissions. Roles are used as a middle layer in between users and permissions. Users are human beings and permissions are a set of many-to-many relations between objects and operations. Roles bring users and permissions together, representing the job functions or titles and making it easy to apply organizational policies to the job functions or titles. Roles also decouple users and permissions, thereby reducing administrative routine works.

More advanced notions of roles include their hierarchical and constrained existence. The reference models of RBAC discuss those aspects of roles as well. Roles can be hierarchically structured so as to describe the lines of competencies and responsibilities within an organization. In the hierarchical structures, senior roles generally inherit the permissions assigned to junior roles, and this enables the role layer to be multi-layered, thereby further reducing the number of relations between users and permissions. Roles must be constrained in their relations to users and permissions as well as in the role-hierarchies. Constraints are an essential construct needed for laying out higher-level access control policies within an organization. A well-known example of constraints is the separation of duty. For instance, the same user cannot be a member of roles in a conflicting role set such as `purchasing manager role` and `accounts payable manager role`. The separation of duty constraint reduces possible frauds or errors by controlling membership in, activation of, and use of roles as well as permission assignment.

RBAC reference models incorporate notions related to roles such as competency, responsibility, competency inheritance, and constraints into their integral components. Four conceptual models are discussed in a reference model [1]: RBAC_0, RBAC_1, RBAC_2, and RBAC_3. RBAC_0 is the base model, defining a minimum collection of RBAC components for the purpose of completely realizing a role-based access control system. It is made up of six components: users (U), roles (R), permissions (P), sessions (S), user assignment (UA), and permission assignment (PA). RBAC_1 adds the component of role hierarchies (RH). Role hierarchy is mathematically a partial order defining roles' inheritance of users and permissions. RBAC_2 includes the component of constraints. RBAC_3, as a consolidated model, combines RBAC_1 and RBAC_2.

Role engineering and related works

In order to fully leverage the concept of using roles in access control and benefit from the features of RBAC reference models, it is important to have a good understanding of role engineering process. Role engineering (RE) is essentially a requirement engineering for a later RBAC design and

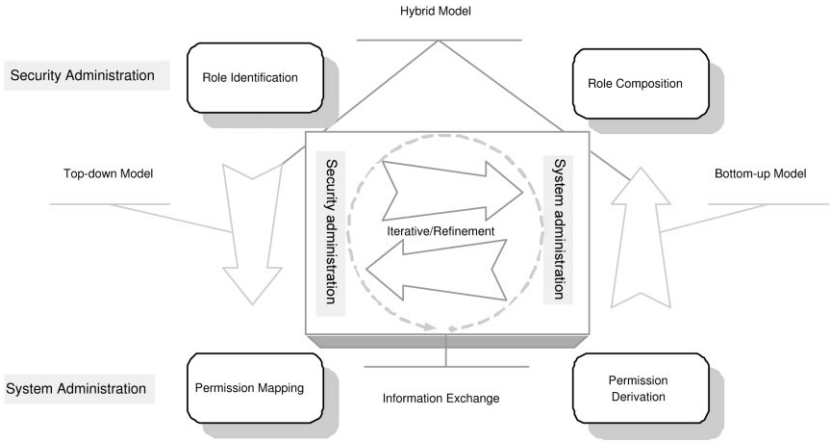


Figure 1. Three models in role engineering (RE).

implementation. It is an approach to defining roles and assigning permissions to the roles [7], and thus enables RBAC system developers to identify and build explicit objects used in access control from the implicit existence of roles within an organization. In RBAC reference models, role engineering activities concern the components such as roles, role hierarchies, permissions, permission assignment, and the constraints.

There have been various approaches to engineering roles, and they can be generally categorized into the following three models: *top-down*, *bottom-up*, and *hybrid* [8,9]. Figure 1 shows the three models and describes the iterative nature of information exchange between two authoritative domains which are most likely to be involved in the RE process, *security administration* and *system administration*. Security administration is in charge of organizational security (policy) management, whereas system administration concerns information system management.

Top-down model

The top-down model can be generally described as an approach to deriving permissions from roles through the use of abstract concepts such as work-patterns and business processes [8,10]. Looking at the model in some more detail, work-patterns or business processes in which roles are involved are analyzed and decomposed into smaller units in a functionally independent manner through the role identification process. Afterwards, those smaller units or tasks are mapped onto permissions in information systems for their execution through permission mapping process. As shown in Figure 1, role identification falls within the scope of security administration, while permission mapping is in the range of system administration.

Coyne [7] briefly describes how to identify roles in RBAC in a top-down manner. Taken as a whole, his approach uses system users' activities as a high-level of abstraction to identify candidate

roles and remove duplicate candidate roles. Then permissions can be identified as a minimal set of access rights on the systems required to perform the roles. Finally, constraints definition follows before role hierarchies are built. Though introducing the concept of RE to the point, his approach lacks many technical details of the RE process, only to be depicted in a highly conceptual manner.

Roeckle *et al.* [10] discuss a process-oriented approach to finding roles in a top-down manner. The concept of *role-finding* is described for the purpose of deducting roles from business needs or functions. Their approach deals with an RBAC metamodel (we will discuss this in more detail in 'Functional requirements') to describe the notion of roles and their relation to users and access rights. Three different layers are discussed in conjunction with the metamodel: process layer, role layer, and access rights layer. Simply put, business processes are initially analyzed in the process layer. Then roles are derived from the business processes in the role layer and access rights on systems are assigned to the roles in the access rights layer. While their metamodel is well defined and structured, their procedural approach to finding roles lacks the support of some important issues such as how to handle role information update.

Bottom-up model

In the bottom-up model, permissions are generally working as a building block so as to be aggregated into roles. Like the top-down model, this model often uses abstract concepts such as scenarios and business functions in order to both derive and group permissions [11]. In addition, certain attributes of target objects and operations can be used for that purpose as well. More specifically speaking, permissions (object and operation pairs) are derived from existing information systems and grouped on the basis of the certain attributes of permissions. The attributes can be drawn from objects, applications, and systems where those permissions are involved. For example, the owner information or ACLs in file objects can be security-relevant attributes and used for grouping permissions as a functional building block of roles. This is usually done through the permission derivation process which falls within the scope of system administration. Afterwards, those permissions are aggregated to roles through the role composition process which falls within the scope of security administration.

Thomsen *et al.* [12] propose an RBAC framework for network enterprises, in which permissions are derived from objects and their methods and roles are derived from the permissions. All these procedures are enabled by an introduction of seven abstract layers for security management: object, object handles, application constraints, application keys, enterprise keys, key chains, and enterprise constraints. The first four layers belong to application developers, who can use their in-depth knowledge of the applications in order to create generic security components. The last three layers are under the control of system administrators, who can use the generic security components as the security building blocks in order to customize the security policy for their organization. They developed the *NAPOLEON* tool, which implements the portion of the framework used by the application developer. Subsequently, Epstein and Sandhu [13] propose an approach to leveraging UML language for RE and discuss an exemplary UML modeling case which is based upon the framework proposed by Thomsen *et al.* Their approach is straightforward in representing the RBAC framework. However, their approach needs to be improved to address how UML can be used for modeling the process side of RE.

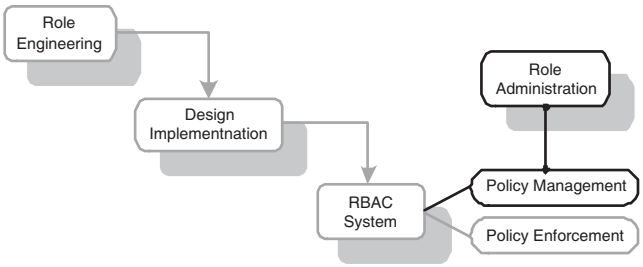


Figure 2. Role administration as access control policy management.

Hybrid model

The hybrid model can be described as a mixed approach of top-down and bottom-up models so as to engineer RBAC roles. For example, the role identification process and the permission derivation process in Figure 1 can be conducted in parallel for a later role definition.

Epstein and Sandhu [8] propose a conceptual framework where roles can be defined in either a top-down or a bottom-up manner. They extend RBAC reference models by introducing three additional layers in between roles and permissions. They use the concepts of jobs, work-patterns, and tasks, to represent those respective layers and facilitate role-permission assignment into smaller and better steps. Top-down and bottom-up models are discussed in conjunction with the notions of *focus* and *bucket*, respectively. However, their work addresses RE in a conceptual manner without discussing how those concepts are specified, constructed, or concretized.

Role administration and related works

After the role engineering process, roles and other RBAC components are designed and implemented for RBAC system development. In general, an RBAC system is composed of two modules: a policy management module and a policy enforcement module. The former concerns proper configuration of RBAC components so as to reflect organizational access control policies, while the latter is related to how to practice those configured RBAC components for access control services. As shown in Figure 2, role administration is an integral aspect of RBAC policy management, allowing the configuration of RBAC components. Hence it is needless to say that the administration of roles should be done cautiously so as not to diverge from organizational access control policies.

Ferraiolo *et al.* [14] discuss an approach to designing and implementing RBAC features for Web servers. To support a comprehensive implementation of RBAC services for the Web servers, they developed a system, called *Admin Tool*, to manage user/role and role/role relationships and to store those relationships in the RBAC database. The system supports role-hierarchies, cardinality constraints and dynamic and static separation of duties. However, the use of *Admin Tool* is limited to only Web-based applications.

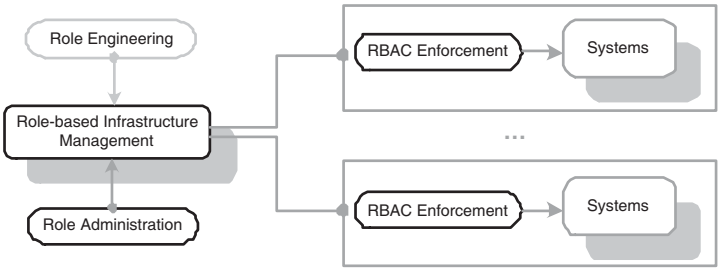


Figure 3. Role-based infrastructure management, role engineering, and role administration.

Neumann and Strembeck [15] present the design and implementation of a flexible RBAC service using the language of *XOTcl*, called xoRBAC. The xoRBAC implementation conforms to level 4a of the unified NIST model for RBAC. xoRBAC supports the functionality for user/role assignment, permission/role assignment, arbitrary role hierarchies, static separation of duties, and cardinality constraints. However, their work focuses on mainly RBAC enforcement, barely discussing role administration or policy management.

Kern *et al.* [9] propose a life cycle model of roles, interwoven with role engineering and role administration processes as dicussed. The life cycle model is based on an iterative-incremental process similar to those found in the area of software engineering. Four stages of the life cycle of roles are identified: role analysis, role design, role management, and role maintenance. Role analysis is the activity of identifying roles as they occur within the target domain. Role design involves with mapping roles onto the system-dependent syntax and semantics as well as designing roles for administration. Role management is the routine role administration such as creation or deletion of a user or a permission and changes in the role model. Role maintenance activities are composed of changes in the mapping of organizational structures to role and changes in the definition of user–role and role–permission relationships.

OUR APPROACH

Figure 3 describes our understanding of role-based infrastructure management in conjunction with role engineering and administration. Role-based authorization infrastructures are likely to encompass a variety of complex systems or applications within or across enterprise whose access control services depend upon roles. Considering the number of roles existing in enterprise-wise environments, role engineering and administration are unarguably a challenging task for role-based infrastructure management. Though they are equally important, we mainly restrict our attention to the system development for role administration in this paper due to the space limitation (for more details on role engineering, refer to [16]).

Before describing how to design and implement a role-based infrastructure management system concentrating on role administration, we believe that it is important to be cognizant of the general

requirements of such a system. We identify and depict those requirements, which are to be factored later into our design and implementation decisions, as follows.

- *Availability*: a role-based infrastructure management system should be available in as many system environments as possible. This issue pertains to the platform-independence of, the support of various formats of authorization policies in, and the scalability of the system.
- *Applicability*: a role-based infrastructure management system should be applicable in different kinds of organizational policy environments. In order to do that, the system should support all RBAC components whose configuration and customization enhance the system's applicability into different organization policy environments.
- *Ease-of-use*: a role-based infrastructure management system should enable role administrators to perform role management activities with ease. The system should also have an intuitive visual interface to make it usable without extensive formal training.
- *Standardization*: a role-based infrastructure management should adhere to RBAC standards in its desired functionality, such as multiple inheritances in role hierarchies.

Methodology

RBAC reference models and their components are conceptual. For the purpose of building a concrete role-based infrastructure management supporting RBAC, RBAC components in the reference models need to be reorganized according to the system's structural and behavioral characteristics. Hence we classify RBAC components into three constituents: structural, functional, and informational. We use the RBAC96 model and its components with the support of constraints such as cardinality (users and roles), static separation of duty, and prerequisite condition.

Structural constituents

The structural constituents consist of users, roles, permissions, role hierarchies, objects, operations, and constraints. The structural constituents represent both their semantics and syntaxes. The semantic of each of those components is well defined in the RBAC framework. For instance, roles may have the meaning of job functions or titles and users may have the meaning of employees in an organization. However their syntactic representations within concrete role-based systems are wholly dependent upon the developers. Thus we define the structural constituents to address both semantic and syntactical representation of those RBAC components.

Functional constituents

The functional constituents comprise some RBAC components which can represent the assignment relationships among users, roles, and permissions. Hence we include user assignment and permission assignment in functional constituents. Additionally, search functions such as *USERS* and *ROLES* are included in this category. Using *USERS*, we find out which roles are associated with a user, or which users are entitled to a role. *ROLES* returns which roles are associated with a permission, or which permissions are assigned to a role.

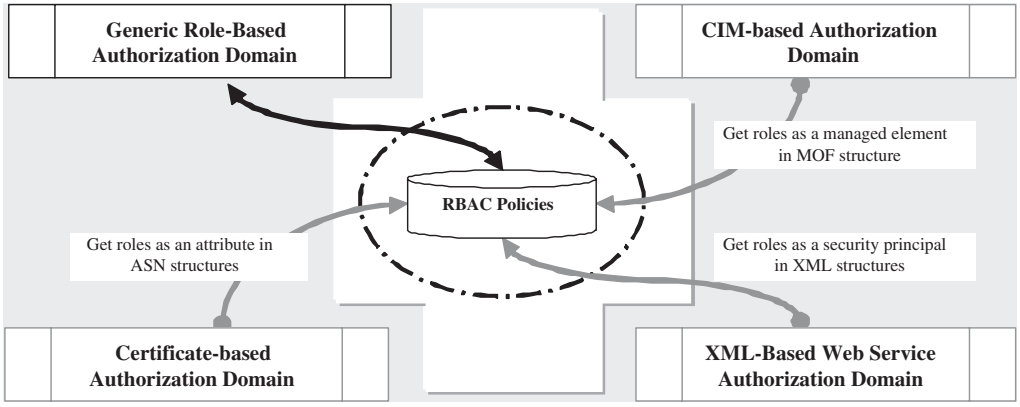


Figure 4. Separating RBAC policies from their enforcement mechanisms.

Informational constituents

The informational constituents represent repositories such as directory services and relational database systems. Informational constituents are composed of a database for roles and role–hierarchy information, a database for role–permission information, and a database for role–user information.

THE ROLEPARTNER: DESIGN

One of our primary concerns in designing a role-based infrastructure management system is the separation of RBAC policy from its enforcement mechanisms, and *RolePartner* is designed to work as a centralized RBAC policy administration system which is separated from various policy enforcement mechanisms. Figure 4 shows four different domains which possibly leverage RBAC policies built through *RolePartner* for their access control services: generic role-based, CIM-based, XML-based, and certificate-based. Note that each of them may need RBAC policies in its preferable format, such as ASN and XML.

Functional requirements

In addition to the general requirements discussed earlier, a role-based infrastructure management system should have the functional requirements which describe its behavior within role-based authorization environments. To determine the functional requirements of the system, we defined its use cases and their relationships, as shown in Figure 5. The use cases are grouped into three use cases groups according to our methodological approach: *use case group A*, related to the structural constituents, *use case group B*, relevant to the functional constituents, and *use case group C*, pertaining

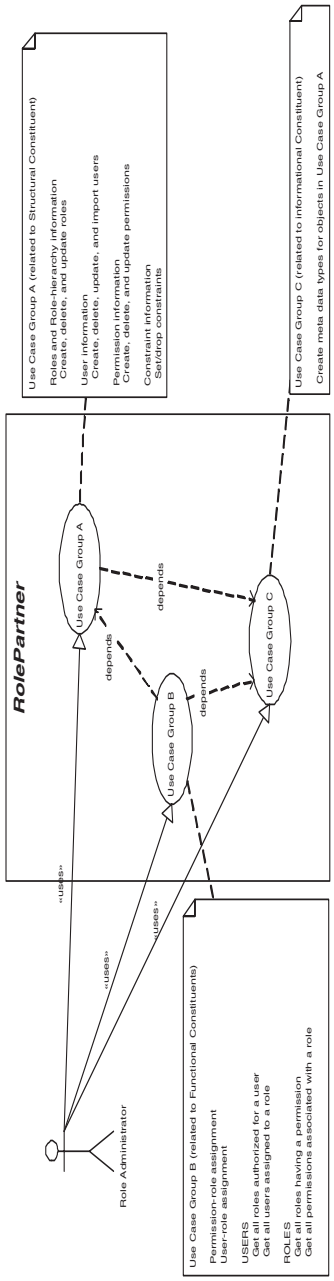


Figure 5. Functional requirements of the *RolePartner*, defined by three groups of use cases.

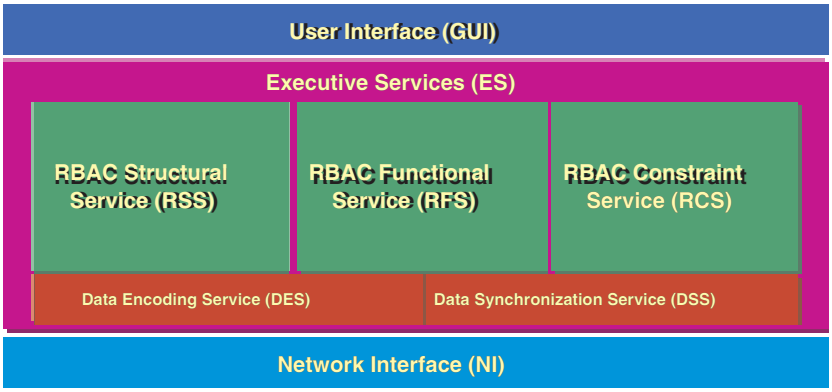


Figure 6. Structural overview of *RolePartner*.

to the informational constituents. Since these use cases describe all the required interactions of role administrators with *RolePartner*, they improve the general understanding of the system. Note that in *use case group A*, not only being able to create, delete, and update user information, but role administrators can also import the user information from an existing database through *RolePartner*. This is because organizations are most likely to have their existing user databases, and we believe the use case of importing users is essential in order to improve the system's integration flexibility.

Structural components

RolePartner is composed of three components: user interface (UI), executive services (ES), and network interface (NI). UI is responsible for the graphical user interface for role administrators. NI is responsible for database connections to store or retrieve static role-based authorization policies, which are created or maintained through ES. ES is further divided into five sub-services which are RBAC functional service (RFS), RBAC structural service (RSS), RBAC constraints service (RCS), data synchronization service (DSS), and data encoding service (DES). As their names imply, the scope of services provided by each of them is straightforward. RSS concerns the *use case group A* excluding constraint information, whereas RFS involves *use case group B*. RCS is in charge of constraint information that *RolePartner* supports such as cardinality (both *user* and *role*), static separation of duty, and prerequisite condition.

DSS is involved in data synchronization in between role database and *RolePartner* in case of data changes occurring in either of them. DES is responsible for encoding and decoding services for data, which allow for the support of various formats of authorization policies. For example, the data for role-based authorization policies often need to be encoded into XML or binaries in some authorization service environments we discussed earlier. Finally, the activities of storing RBAC component data and creating meta data types for them are carried out through collaboration of ES and

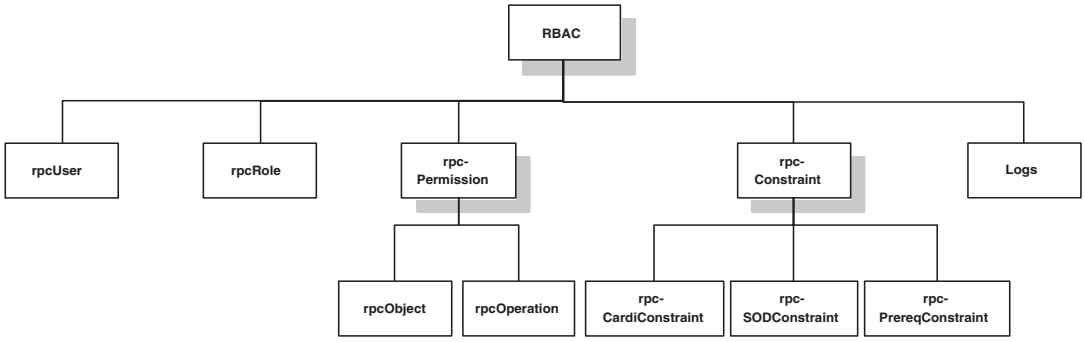


Figure 7. Overview of directory service for *RolePartner*.

NI components. This collaboration concerns *Use case group C*. Figure 6 gives an overview of those structural components of *RolePartner*.

Directory as informational constituent

We leverage directory service as our informational constituents for *RolePartner*. Directory service is one of the common data repositories, strongly believed to be an optimal solution for making enterprise-wide information available to different systems within enterprise. Figure 7 shows the overview of the directory service designed to hold RBAC policies. On its top level, there are *rpcUser*, *rpcRole*, *rpcPermission*, and *rpcConstraint*, which represent users, roles, permissions, and constraint in RBAC reference models, respectively. *Logs* represents the logging service for access and error information.

Developing databases in the directory service necessitates the schema design for the needed objects such as role, user, and permission. Table I describes the schema of the roles component in a database of roles and role hierarchy, called `rpcRole object class`. The `rpcRole object class` with its attributes embodies a conceptual roles component in RBAC in directory service. The `rpcRole object class` must have attributes such as `rpCompID` (*role ID*), `rpCompName` (*role name*), and `rpCompType` (*role type*), and may have attributes for role hierarchy information (`rpImmediateSeniorRole` and `rpImmediateJuniorRole`), for role member (`rpRoleMember`), for associated permission (`rpRolePermission`), and for encoded data format (`rpBerEncodedVal` and `rpXACMLEncodedVal`). The operations on the roles in the directory service should follow the rules and structures defined in `rpcRole object class`.

Similarly the schemas for the components in the role–permission database and the role–user database are designed. The permissions component can be further divided into two: general permissions and specific permissions, which are differentiated by the level of abstraction. For example, permissions such as *credit* or *debit* can be represented by general permissions, while specific permissions can account for privileges such as *modify* and *execute*, file *f1* (operations and object pair).

Table I. Schema design of roles, defined as `objectclass rpcRole`.

rpcRole		
Definition	Object class for Roles in RBAC	
Superior class	top	
OID	RP-OID.2.2	
Attribute	Description	Value (if multivalue, *)
Required attribute		
objectClass	Entry's objectclass	
rpCompID	Role ID	DirectoryString
rpCompName	Role name	DirectoryString
rpCompType	Role type (1:local, 2:global)	Integer
Allowed attribute		
rpRoleMember	Role members	DN*
rpRolePermission	Role privileges	DN*
rpImmediateSeniorRole	Immediate senior roles to this role	DirectoryString*
rpImmediateJuniorRole	Immediate junior roles to this role	DirectoryString*
rpBerEncodedVal	Value of role information encoded in BER	Binary
rpXACMLEncodedVal	Value of role information encoded in XML	DirectoryString
rpLastModified	Last modification time for logging purpose	GeneralizedTime
createTimestamp	Creation time for logging purpose	GeneralizedTime
creatorsName	Creator's name	DN
description	Description of this role	DirectoryString*
rpExtensions	Extension field	Binary*

Operational architecture

RolePartner is a client and server-based application enabling RBAC policy management through role administration. It uses LDAP protocol to communicate with the directory service server. The role administrator interacts with *RolePartner* through its UI component, which is playing the role of delivering commands issued by the role administrator to a facade of ES services. The facade of ES services provides a single interface to RFS, RSS, and RCS. RSS and RFS are in charge of operations such as building roles or building assignment relationship. By contrast, in addition to the functionality of creating static constraint information, RCS has another task, which is monitoring the operations of RSS and RFS in runtime in order to enforce the constraint policies. As for DES, it is responsible for data encoding and decoding service. DSS is in charge of synchronization of data between role databases and *RolePartner*. Finally, NI provides an interface to an LDAP-accessible directory service server for synchronous operations. Figure 8 shows the operational architecture, which gives the details of relationships among the components.

For further detailed interactions among the components, we need to mention the objects within and their interactions among the components. For instance, we design the following classes which involve a process of adding a new role into a role hierarchy.

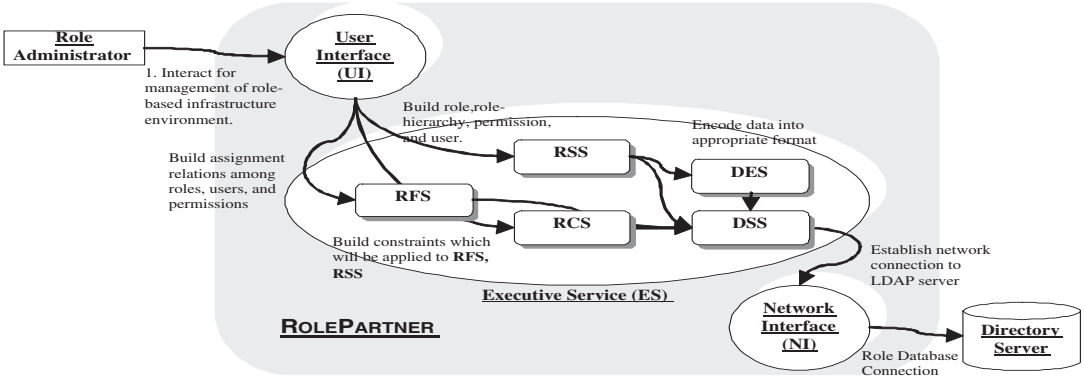


Figure 8. Operational architecture of *RolePartner*.

- Class `UIWorker` works as a main user interface, managing all input from a role administrator and communicating with `ESFacade`.
- Class `ESFacade` provides `UI` with a simple interface to sub-components of the `ES` component.
- Class `StrucCompWorker` manages the creation of the structural components (users, roles, permissions, objects, and operations) through `StrucCompFactory`, encodes role information into appropriate formats such as BER or XACML, and handles role hierarchy through `RoleHierarchyWorker` class.
- Class `RoleHierarchyWorker` is in charge of managing role hierarchy in the process of creation, deletion, and modification of roles.
- Class `SyncWorker` primarily manages the synchronization of data in between the role database (class `DirEventNotifier`) and data pools which are local to `RolePartner` (Class `ComponentPool`).
- Class `DirEventNotifier` provides support for event notification when changes in directory service occur.
- Class `NIWorker` is in charge of connection to, creation of entries (both schema and objects) in, and manipulation of entries in the directory service (role database).

To add a role into role hierarchy, `UIWorker` accepts a role administrator's input value for attributes associated with the role to be created. `UIWorker` passes the input value to `ESFacade`. Afterwards, `ESFacade` invokes `StrucCompWorker` to create a role object (type of `RoleSComp`) from `StrucCompFactory`. Then `StrucCompWorker` passes the created role object to `RoleHierarchyWorker`, which checks the integrity of role hierarchy with the addition of the role object. Only after getting an OK response from `RoleHierarchyWorker`, `StrucCompWorker` passes the role object into `SyncWorker`. Otherwise, the process ends. Subsequently, `SyncWorker` stores the object into its role object pool as well as passing the object to `NIWorker` so that `NIWorker` can update a role database with the object. If updating the role database is a success, `SyncWorker` will get an update notification from `DirEventNotifier` and

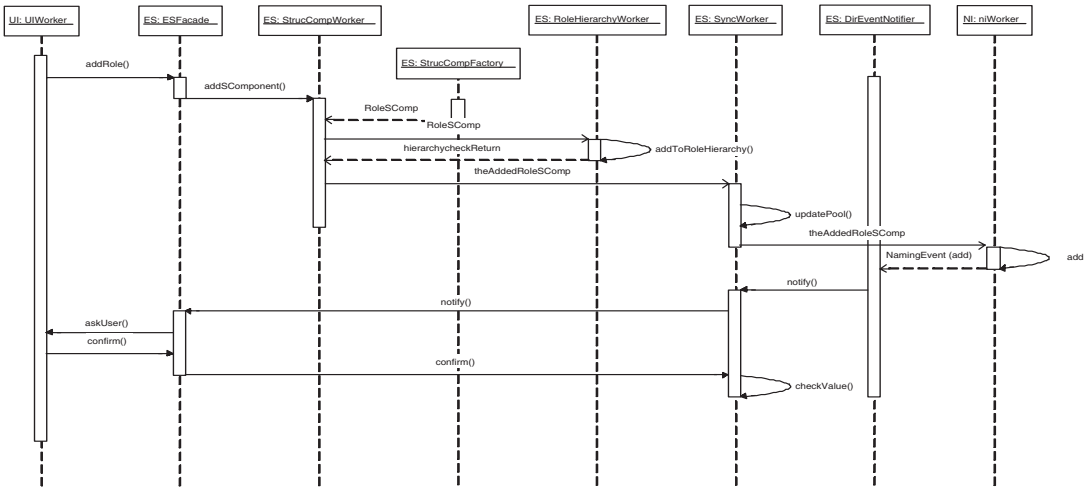


Figure 9. A sequence diagram for adding a new role in role hierarchy.

it will notify `ESFacade` of the update. Afterwards, the role will get a confirmation prompt from `ESFacade`. Finally, the confirmation message will be passed to `SyncWorker` to finish this process. Figure 9 shows a sequence diagram describing the process.

THE ROLEPARTNER: IMPLEMENTATION

We implemented *RolePartner* using Java language, primarily because of its system-independent feature. We used JDK 1.4 to develop the components identified in the design phase: Swing for UI and JNDI for NI. For directory service as a role database, we used iPlanet directory service 5.0. One thing to note in leveraging iPlanet directory service is that we need to add the access control information (ACI) attribute value similar to the following ACI into RBAC component container entries with a view to managing access control to the entries on the basis of administrative roles in *RolePartner*.

- (targetattr = "*") (version 3.0; acl "RBACRoleAdmin"; allow (all) (userdn = "ldap:/// *base DN* ??sub?(rprole = *DN of administrative role*)");)

There are four administrative roles for *RolePartner*: `rpSuperMgr`, `rpRoleMgr`, `rpUserMgr`, and `rpConsMgr`. The `rpSuperMgr` role has all the privileges needed for operating *RolePartner*. The `rpRoleMgr` role has privileges effective only upon roles, permissions, and permission assignment. Alternatively, the `rpUserMgr` role has privileges limited only to users and user assignment. The `rpConsMgr` role, as the last one, is capable of performing operations only related to constraint management.

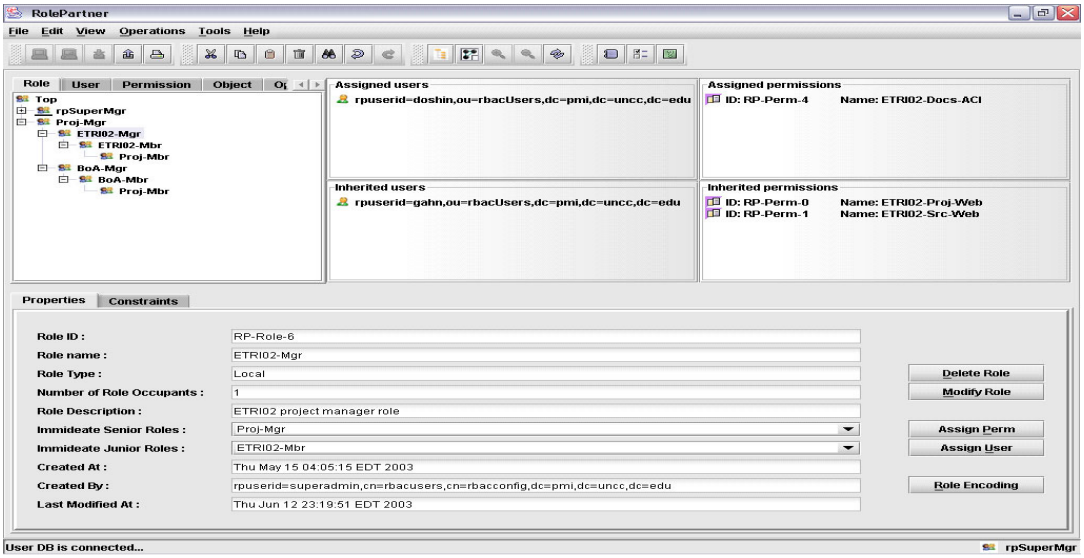


Figure 10. User interface of *RolePartner*.

Figure 10 shows the interface of our system, logged in with `rpSuperMgr` role. In the upper leftmost tabbed pane of the system is displayed a role hierarchy tree, where two disjoint role hierarchies are connected through the root node of the tree, Top. The root node is not included in the role hierarchies, since it only works as a base node connecting two or more disconnected role hierarchy trees. The same role hierarchies can be viewed as a role graph as shown in Figure 11. The *RolePartner* displays a role-centric view, which allows for managing RBAC components in an efficient and easy-to-use manner. When a role in the upper leftmost tabbed pane is clicked, the system displays in the remaining six panes all information relevant to the role, i.e. currently assigned members/inherited members, currently assigned permissions/inherited permissions, its properties, and relevant constraints.

User assignment, permission assignment, and constraint management

We implemented two functions for user assignment: `assignUser2Role` and `assignRole2User`. `assignUser2Role` handles a role administrator's request for assignment or de-assignment of users to a role, while the role administrator views the information or manages the properties related to the role. By contrast, `assignRole2User` takes care of a role administrator's request for assignment or de-assignment of roles to a user, while the role administrator views the information or manages the properties related to the user. In `assignUser2Role`, a set of assignable users to role `r` (AU_r) can be derived as follows:

$$\mathrm{AU}_r \subseteq \mathrm{U} - \{\mathrm{CAU}_r \cup \mathrm{UAU\text{-}SSOD} \cup \mathrm{UAU\text{-}PRE} \cup \mathrm{UAU\text{-}CARDI\text{-}U}\},$$

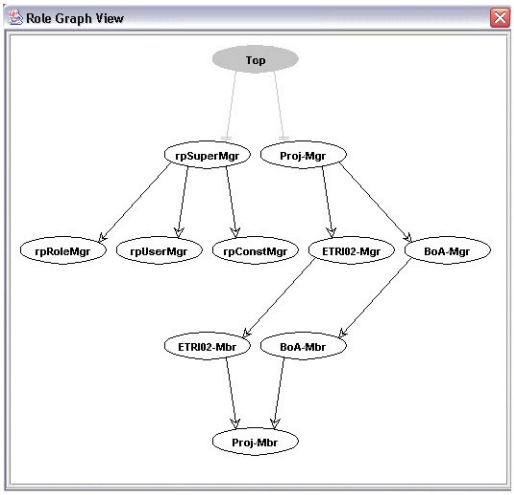


Figure 11. Graph view of role hierarchy tree shown in Figure 10.

where

- U: all users;
- CAU_r: all users currently assigned to role r;
- UAU-SSOD: all unassignable users due to separation of duty constraints;
- UAU-PRE: all unassignable users due to prerequisite role constraint;
- UAU-CARDI-U: all unassignable users due to user cardinality constraint.

By the same token, a set of assignable roles to user u (AR_u) can be derived as follows:

$$\mathrm{AR}_u \subseteq \mathrm{R} - \{\mathrm{CAR}_u \cup \text{UAR-SSOD} \cup \text{UAR-PRE} \cup \text{UAR-CARDI-R}\},$$

where

- R: all roles;
- CAR_u: all roles currently associated with user u;
- UAR-SSOD: all unassignable roles due to separation of duty constraints;
- UAR-PRE: all unassignable roles due to prerequisite role constraint;
- UAR-CARDI-R: all unassignable roles due to role cardinality constraint.

Similarly, we implemented two functions for permission assignment: `assignPerm2Role` and `assignRole2Perm`. `assignPerm2Role` allows a role administrator to assign or de-assign permissions to a role while he/she views the information or manages the properties related to the role. `assignRole2Perm` lets the role administrator assign or de-assign roles to a permission while he/she views the information or manages the properties related to the permission. In `assignPerm2Role`,

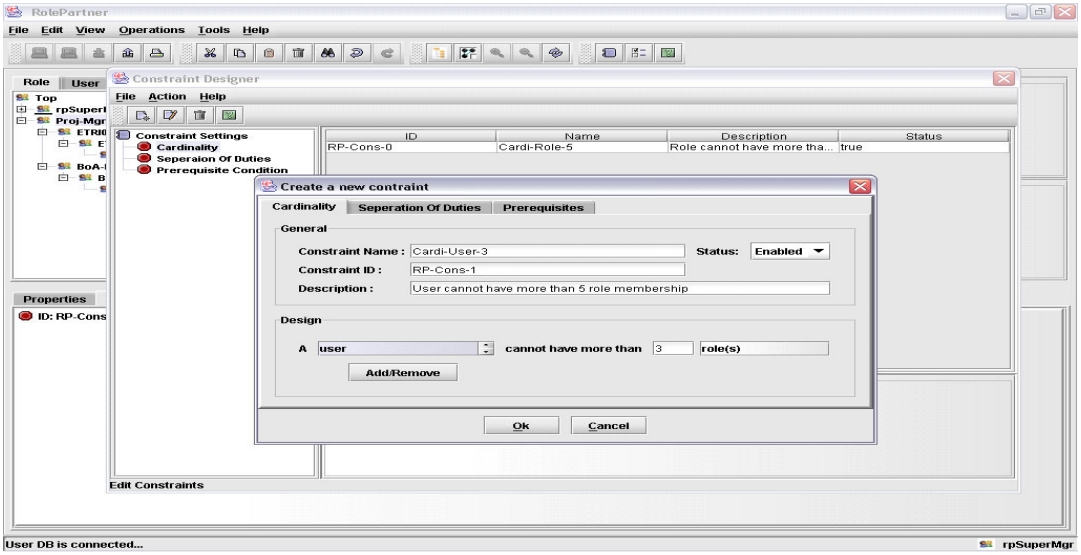


Figure 12. User interface of constraint designer provided by *RolePartner*.

a set of assignable permissions to role r ($\mathtt{AP}_r$) can be derived as follows:

$$\mathtt{AP}_r \subseteq \mathtt{P} - \{\mathtt{CAP}_r \cup \mathtt{UAP\text{-}SSOD}\},$$

where

- P: all permissions;
- $\mathtt{CAP}_r$: all permissions currently assigned to role r;
- UAP-SSOD: all unassignable permissions due to separation of duty constraints.

In a similar way, a set of assignable roles to the permission p ($\mathtt{AR}_p$) can be derived as follows:

$$\mathtt{AR}_p \subseteq \mathtt{R} - \{\mathtt{CAR}_p \cup \mathtt{UAR\text{-}SSOD}\},$$

where

- R: all roles;
- $\mathtt{CAR}_p$: all roles currently associated with permission p;
- UAR-SSOD: all unassignable roles due to separation of duty constraints.

In addition to the basic role management operations such as user/permission assignment, *RolePartner* supports constraint management by providing role administrators with a constraint designer, which is shown in Figure 12. The constraint designer is capable of managing the following sets of constraints.

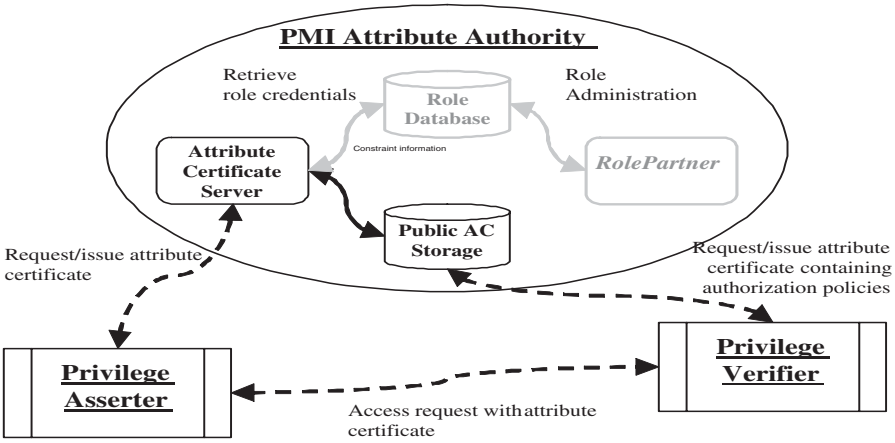


Figure 13. Integration of the *RolePartner* into a PMI.

- `Cardi-R`$_n$: a set of roles to which the maximum n number of users can be assigned;
- `Cardi-U`$_n$: a set of users with which the maximum n number of roles can be associated;
- `SSOD-CR`: a set of roles conflicting each other;
- `SSOD-CU`: a set of users conflicting each other;
- `SSOD-CP`: a set of permissions conflicting each other;
- `Prereq-R`$_r^{UA}$: a set of roles to which a user has to be assigned before being assigned to role `r`.

INTEGRATION INTO A PRIVILEGE MANAGEMENT INFRASTRUCTURE

In order to demonstrate the feasibility of our approach, we integrated *RolePartner* into the privilege management infrastructure (PMI) leveraging X.509 attribute certificate [17–19]. PMI is a collection of attribute certificates, attribute authorities, repositories, entities involved such as privilege asserters and verifiers. It provides certificate-based scalable and interoperable authorization. The attribute certificate binds entities to attributes, which may be the entities' role or group information. The PMI control model explains how access control is managed when privilege asserters request services.

Figure 13 describes where our *RolePartner* is positioned and how it functions in the authorization service of the PMI. *RolePartner* is placed in PMI attribute authority (AA), which is the entity responsible for digitally signing two types of the attribute certificate: role assignment attribute certificate (*RAAC*), which contains a user's roles, and role specification attribute certificate (*RSAC*), which contains all associated permissions to a role. Using *RolePartner*, role administrators in PMI AA manage and configure *RBAC* components in the role database. As for data encoding, *RolePartner* supports BER encoding of data, which will be included in *RSAC*. At the privilege asserter's request, the attribute certificate server issues the attribute certificate containing the role information of the

privilege asserter. When the privilege asserter requests services through presenting his/her privileges in attribute certificates, the privilege verifier makes access control decisions based upon the role presented and authorization policies.

DISCUSSION AND FUTURE WORKS

RolePartner allows role administrators to manage role-based security policies in a centralized manner. Thus, its usage is more appropriate for the authorization environments where tight control of security policies by a limited number of role administrators is required. However, taking into account possible overburden of those administrators in managing a large number of roles and associated security policies, decentralizing administrative authority would be desirable. RolePartner supports decentralized administration in a limited way. Sandhu *et al.* [4] proposed a model, called ARBAC97, for managing RBAC with RBAC itself through decentralization of administrative authority. It has three components: URA97 (user–role assignment), PRA97 (permission–role assignment), and RRA97 (role–role assignment). Central notion of ARBAC97 is *role ranges*, which is used to impose restrictions on the role administration boundary. Our short-term plan for enhancing the system pertains to this issue as well as role delegation.

Comprehensive constraint support would be advantageous in many aspects, such as improving the applicability of *RolePartner* or facilitating new RBAC features like context sensitive roles. Currently, *RolePartner* supports static separation of duty, role/user cardinality, and prerequisite constraints. A more advanced concept of constraints entails location, time, role activation, as well as dynamic separation of duty. Future development of the system should address this issue as well.

CONCLUSION

Role management in various implementations of RBAC services tends to be conducted on an *ad hoc* basis or in a system-dependent way. In this paper we describe a role-based infrastructure management system, called *RolePartner*. The main purpose of the system is to help a role administrator establish a valid set of roles and role hierarchies with assigned users and associated permissions. The role administrator can define, build, and manage the components of RBAC96 model. We present how we designed and implemented the system in a systematic way, with the help of a methodological approach to facilitating role administration through the system. The *RolePartner* has a role-centric view for easily managing constrained roles and associated permissions and users, and it leverages the LDAP-accessible directory service for storing role-based authorization policies. We also show that the system can be seamlessly integrated within the existing privilege-based authorization infrastructure.

ACKNOWLEDGEMENTS

This work was partially supported by grants from the Electronics and Telecommunications Research Institute and National Science Foundation (IIS-0242393). Portions of this paper appeared in preliminary form in *Proceedings of the 18th ACM Symposium on Applied Computing (SAC03)*.

REFERENCES

1. Sandhu RS, Coyne EJ, Feinstein HL, Youman CE. Role-based access control models. *IEEE Computer* 1996; **29**(2):38–47.
2. Ferraiolo DF, Sandhu R, Gavrila S, Kuhn DR, Chandramouli R. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security* 2001; **4**(3):224–274.
3. Ahn G-J, Sandhu R. Role-based authorization constraints specification. *ACM Transactions on Information and System Security* 2000; **3**(4):207–226.
4. Sandhu R, Bhamidipati V, Munawer Q. The ARBAC97 model for role-based administration of roles. *ACM Transactions on Information and System Security* 1999; **2**(1):105–135.
5. Zhang L, Ahn G-J, Chu B-T. A rule-based framework for role-based delegation. *ACM Transactions on Information and System Security* 2003; **6**(3):404–441.
6. Goh C, Baldwin A. Towards a more complete model for role. *Proceedings of 3rd ACM Workshop on Role-Based Access Control*, Fairfax, VA, 22–23 October 1998.
7. Coyne E. Role engineering. *Proceedings of the 1st ACM Workshop on Role-Based Access Control*, Gaithersburg, MD, November 1995.
8. Epstein P, Sandhu R. Engineering of role/permission assignment. *Proceedings 17th Annual Computer Security Application Conference*, New Orleans, LA, December 2001. IEEE, 2001.
9. Kern A, Kuhlmann M, Schaad A, Moffett J. Observations on the role life-cycle in the context of enterprise security management. *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies*, Monterey, CA, June 2002.
10. Roeckle H, Schimpf G, Weidinger R. Process-oriented approach for role-finding to implement role-based security administration in a large industrial organization. *Proceedings of the 5th ACM Workshop on Role-Based Access Control*, Berlin, Germany, 26–27 July 2000.
11. Neumann G, Strembeck M. A scenario-driven role engineering process for functional RBAC roles. *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies*, Monterey, CA, June 2002.
12. Thomsen D, O'Brien D, Bogle J. Role based access control framework for network enterprises. *Proceedings 14th Annual Computer Security Application Conference*, Scotsdale, AZ, 7–11 December 1998. IEEE, 1998; 50–58.
13. Epstein P, Sandhu R. Towards a UML based approach to role engineering. *Proceedings of the 4th ACM Workshop on Role-Based Access Control*, Fairfax, VA, 28–29 October 1999; 33–42.
14. Ferraiolo DF, Barkley JF, Kuhn DR. A role based access control model and reference implementation within a corporate intranet. *ACM Transactions on Information and System Security* 1999; **2**(1):34–64.
15. Neumann G, Strembeck M. Design and implementation of a flexible RBAC-service in an object-oriented scripting language. *Proceedings of the 8th ACM Conference on Computer and Communication Security*, Philadelphia, PA, November 2001.
16. Shin D, Ahn G-J, Cho S, Jin S. On modeling system-centric information for role engineering. *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies*, Como, Italy, 2–3 June 2003.
17. Farrell S, Housley R. An internet attribute certificate profile for authorization. *Technical Report*, PKIX Working Group, June 2001.
18. ITU-T Recommendation X.509. *Information Technology: Open Systems Interconnection. The Directory: Public-Key and Attribute Certificate Frameworks*. ISO/IEC 9594-8, 2000.
19. Shin D, Ahn G-J, Cho S. Role-based EAM using x.509 attribute certificate. *Proceedings of the Sixteenth Annual IFIP WG 11.3 Working Conference on Data and Application Security*, Cambridge, U.K., 29–31 July 2002.

