Specification and Enforcement of Object-Oriented RBAC Model

Chang. N. Zhang , Cungang Yang

Department of Computer Science
University of Regina, TRLabs
Regina, Saskatchewan, S4S 0A2
zhang@cs.uregina.ca
cungang@cs.uregina.ca

Abstract:

Access control for protection and sharing of information and physical resources is an essential component of any multi-user computer systems. Role-Based-Access-Control (RBAC) has been introduced and has offered a powerful means of specifying access control decisions, as well as reducing the cost of administrating access control policies and making them less error-prone. In this paper, we proposed an object-oriented RBAC model (ORBAC) and its formal specifications to describe the relationships of the basic elements of the model. Furthermore, an efficient ORBAC implementation method was proposed to deal with statically and dynamically role authorization so that the problem of seperation of duties can be solved.

Keywords

RBAC, ORBAC, Seperation of Duties, Constraint, Least Priviledge.

1. Introduction

Access control for protection and sharing of information and physical resources is an essential component of any multi-user computer system. A popular approach for security management is Access Control List (ACL). In ACL, each object has an access control list, indicating that all the accesses to those subjects are authorized on that object. However, in a large distributed system there are millions of objects, and each of which is assigned to thousands of subjects, so the access control list will be enormous in size and their maintainance will be much difficult and costly. To give an acceptable solution to this problem, Role-Based-Access-Control(RBAC) as a key security technology was proposed[1].

The central notion of RBAC is that users do not directly access to enterprise objects, instead, access priviledges are associated with roles, and each user is assigned to one or multiple roles. This idea greatly simplifies management of authorization while providing an appropriate for great flexibility in specifying and enforcing enterprise-specific protection policies and reduce the management cost. Users can be assigned to members of roles as determined by their responsibilities and qualifications, they can be easily reassigned without modifying the underlying access structure.

In the last few years, the fundamentals of RBAC policies have been clearly identified[1], and many RBAC models have been proposed to satisfy security requirements in different areas, such as for role-based-access-control administration model[2][3][4], lattice-based access control model[5], but they are all logic models and have not efficiently represented the real world. In this paper, we proposed a new variation of RBAC model called object-oriented RBAC (ORBAC), which is a an object-oriented one and more easy to be used on distributed applications. Moreover, in this model, the dynamic role authorization and the constraint of seperation of duty problem are also be considered and implemented.

2. Role-Based-Access-Control (RBAC) Model

The RBAC model used in this paper is shown as fig. 1, which is basically the one proposed by Sandhu et al [1]. It consists of four basic components: a set of users (**Users**), a set of roles (**Roles**), a set of priviledges (**Priviledges**), and a set of sessions (**Sessions**). A user is a human being or an autonomous agent, a role is a collection of priviledges needed to perform a certain job function within an organization, a priviledge is an access mode that can be exercised on objects in the system, and each session is a mapping of one user to possible many roles, a user can have multiple session and a session includes multiple activated roles, each session is associated with a single user. A user can be a member of many roles, and a role can have multiple members. A role may have many priviledges, and the same priviledge can be associated to many roles. When a user logs in the system he/she requests to activate some subset of the roles he/she is authorized to play. An activation request is granted only if the corresponding roles is activated at the time of the request.

If an activation request is satisfied, the user submits the request to obtain all the priviledges associated with the role he/she has required to activate. RBAC introduces role hierarchies to reflect an organization lines of authority and responsibility. On the set of roles, a hierarchy is defined by: If $r_i > r_j$, then role r_i will inherite the priviledges of role r_j. Moreover, RBAC introduces the concept of constraints, a common example is of mutual exclusive roles, such as purchasing manager role and account payable manager role, in most organizations the same individual will not be permitted to be a member of both roles, because this will create a possibility of committing fraud, this is the well-known principle called seperation of duties. Constraints ensure the role specifications that actually enforce the access control requirements. A typical RBAC model consists of roles to which users and permissions may be assigned[1]. The assignment of users and priviledges to roles is limited by constraints.

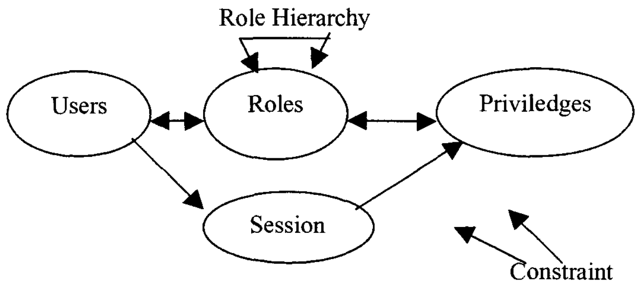

Fig 1: RBAC Model

3. Object-Oriented Role-Based-Access-Control Model (ORBAC)

The proposed object-oriented Role-Based-Access-Control model (ORBAC) described in Fig 2 fully realizes the original RBAC model and can be implemented on a distributed environment. In this section we describe some basic specifications based on RBAC for the ORBAC Model. A number of different viewpoints about RBAC has been discussed[6][7][8]. The abstract model defined in this paper intends to capture the essential feature of RBAC and extend it to satisfy the requirements in the distributed environment. Because the seperation of duty policies are often much important in many commercial applications, the specification for seperation of duty is also proposed.

3.1 Basic elements and their specifications

In this model, class User is a many-to-many relationship with class Role, and class Role is also a many-to-many relationship with class priviledge. Formally User/Role and Role/Priviledge relations can be expressed by the following mapping functions:

(1) $S(t : \text{User}) \rightarrow 2^{\text{Role}}$

2^{Role} : represents any subset of the role

$S[t]$: the user/role mapping, which gives the subset of Role, every element of the role subset is authorized for the user t;

(2) $R(i : \text{Role}) \rightarrow 2^{\text{user}}$

2^{User} : represents any subset of the user

$R[i]$: the role/user mapping, which gives the subset of User, every element of the user subset is authorized for the Role i;

The class Session has been described as below:

Session id: identifying the session.

User: reference the user object of the session.

Roles: reference all the role objects hold by the session.

Functions of the class include adding roles to session, drop roles from session, etc.

The class User is defined as:

User id: identify the user.

Roles: reference to all the role objects of the user.

Sessions: reference to all the session objects of the user.

A priviledge is an approval of a particular operation to be performed on one or more objects, the relationship between roles and priviledges is also many-to-many mapping as shown in fig 2, we describe it by the following mapping functions:

(3) $T(l : \text{Role}) \rightarrow 2^{\text{Priviledge}}$

$2^{\text{priviledge}}$: represents any subset of the priviledge

$T[l]$: the role/priviledge mapping, which gives the subset of priviledge.

every element of the priviledge subset is authorized for the role l;

(4) $C(u : \text{Priviledge}) \rightarrow 2^{\text{role}}$

2^{role} : represents any subset of the role

$C[u]$: the priviledge/role mapping, which gives the subset of Role, every element of the user subset is authorized for the Priviledge u;

We define class Role as below:

Role id: identify the role.

Priviledges: references to all priviledge objects of the role.

Users: references to all user objects of this role.

Parent roles: references to all direct parent roles.

Child roles: references to all direct child roles.

Class Role has functions such as adding, deleting, modifying parent or child roles, adding roles to users, adding, deleting priviledge objects, also, class role has multiple constraint functions, we call them role constraint functions, these functions are used to check role authorization and solve role-role related problems, such as mutual exclusive problems.

The class priviledge is also defined as:

Priviledge id: identifying the priviledge.
Actions: define the actions of the proviledge.
Targets: objects which actions apply.
Roles: references to all role objects of this priviledge.

Functions of the class priviledge includes adding priviledges to roles, deleting priviledge from roles, also, class priviledge has multiple constraint functions, we call them priviledge constraint functions which are used to check whether the authorized roles are satisfied with the priviledges constraints.

3.2 Constraint

ORBAC assigns constraints to user-role and role-priviledge authorization. The association class UR defines user assignment between users and roles, and class static UR and dynamic UR describe that users can be statically or dynamically authorized during a session. In the normal conditions, a user can be assigned many different static roles as it satisfied the principle of "Least Priviledge", which means that a user was assigned least roles to finish a certain task which is benefit for the system security. Static UR object also guarentees the system to prevent the assignment of mutual exclusive roles, but for a business or enterprise environment, flexible and efficient role authorization is also important, it may be acceptable for a user to be a member of two mutual exclusive roles as long as the user can not be active in both roles at the same time. Moreover, object UR has its life cycle, when a user applied for the roles, the UR object will be created, after the task finished, it will be destroyed and system resources will be released.

The relationship class UR can be described as:
User id: identifying the user.
Role id: identifying the role.
The main function of UR is to realize role authorization by calling role constraints functions.

The associated class RP establishs the relationship between roles and priviledges. The activated RP object will check if there is any problem between a uer's authorized priviledges with his/her priviledge requirement, any difference will lead to access failure. Moreover, RP object has its life cycle too, it is created and activated on the server when user submits his priviledge requirements. The main job of RP is to make the decision whether permitting his/her access. Finally, the RP objects will be destroyed and the system resource allocated will be released. Fig 3 describes the relationship of UR and RP on the proposed model.

The relationship class RP can be described as:
Role id: identifying the role.
Priviledge id: identifying the priviledge.
The main function of the class is priviledge authorization by calling priviledge constraints functions.

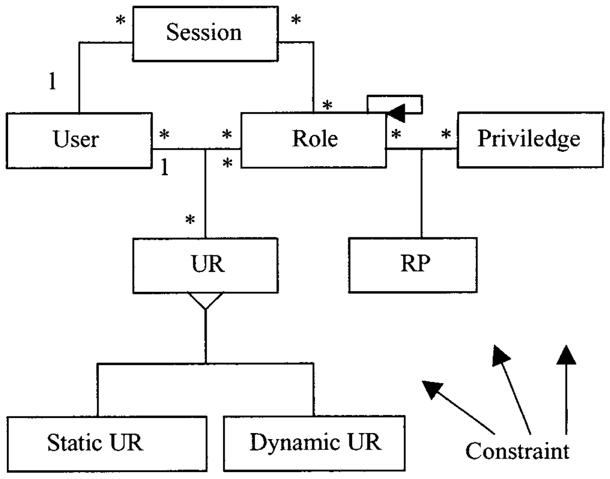

Fig 2 A block diagram of ORBAC Model

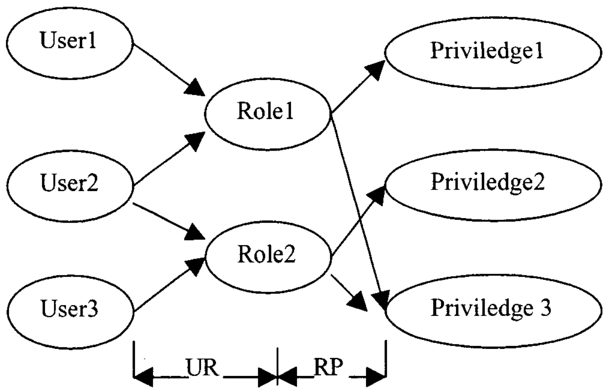

Fig 3: UR and RP on ORBAC

3.3 Mutual Exclusive

The constraint for mutual exclusive roles can be used to enforce conflicts of interest policies that may arise as a result of a user gaining authorization for priviledges associated with conflict roles. That is, if a user is authorized as a member of one of the conflicted two roles, the user is prohibited from being a member of another role.

The constraint functions for mutual exclusive roles are specified as follows:

(5) $E : \text{role} \times \text{role}$

$E[l, m : \text{Role}]$: the set of role pair l and m that are mutual exclusive with each other

(6) The user can not has two exclusive roles.

$(\forall\, l, m)(l \neq m) \wedge E(l, m) \wedge (t \in R[l]) \Rightarrow t \notin R[m]$

(7) Mutual exclusive roles can not inherited each other.

$\forall(l, m)\, \exists(n)\, E(l, m) \Rightarrow \neg((l > m) \wedge (m > l))$

(8) If there are two roles l, m mutual exclusive then there is no role n exists to inherite both of them

$\forall(l, m) \forall(n) E(l, m) \Rightarrow (\neg \exists\, n)((l > n) \wedge (m > n))$

3.4 Dynamic properties of ORBAC

ORBAC dynamic properties include role activation, priviledge execution and dynamic seperation of duties. Dynamic properties provide extended support for the principle of least priviledge, such that each user has different levels of priviledges at different time, depending on the role being performed. The following functions formalize the mappings for these dynamic properties.

(9) Active role

$A(t : \text{User}) \rightarrow 2^{\text{Role}}$

2^{Role} : represent any subset of the role.

$A[t]$: the subset of the role, every element of the subset is a current active role for user t.

(10) $P : \text{user} \times \text{Priviledge} \rightarrow \text{boolean}$

$P[t, u]$: true if and only if user t can execute priviledge u.

(11) Priviledge Authorization :

a user can execute a priviledge only if the priviledge is authorized for a role which the user activated

$(\forall t)(\forall u)(\exists l)(l \in A[t] \wedge u \in T[l]) \Rightarrow P[t, u] = true$

(12) Role Assignment :

A user t can execute a priviledge u only if he/she has selected an active role for priviledge u.

$(\forall t)(\forall u)(\exists\, l)((A[t] \neq 0) \wedge (l \in A(t) \wedge u \in T[l])) \Rightarrow P[t, u] = \text{True}$

(13) Role authorization :

Static Role Authorization : a user's active role n must be in the set of authorized roles for the user t.

$(\forall t)(n \in A[t] \Rightarrow n \in S[t])$

(14)Dynamic seperation of duties :

With dynamic seperation of duties, an organization can address potential conflict - of - interest issues at the time a user's membership is authorized for a role, a pair of roles may be designated as mutual exclusive regarding role activation, that is, a user may be active in only one of the two distinct roles.

$(\forall t : \text{User})(\forall l, m : \text{Role}) E[l, m] \Rightarrow \neg(A[l] \wedge A[m])$

(15) Role hiearchy : Roles are organized into a ordered set so that if a role is included in the authorized or active role sets, roles below it are included also :

$$(\forall l, m)(\forall t)(l \in A[t] \wedge (l > m) \Rightarrow m \in A[t])$$
$$\wedge ((l \in S[t]) \wedge (l > m) \Rightarrow (m \in S[t]))$$

4. The general method for ORBAC implementation

The proposed ORBAC implementation diagram is shown in Fig 4. Each user can implement multiple tasks so he/she can create multiple sessions. In the meantime, each session can activate many different roles. In order to prevent the problem of seperation of duties, a session-table was created by security manager to monitor all the active roles of each user so that there is no mutual exclusive roles are activated simultaneously. The role allocation table is issued to indicate all the roles assigned to each user by the security manager. The basic security architecture table defines role hiearchy and role constraints to present the relationship between roles and their constraints. Resource requirement table defines the relationships between roles, priviledges and their priviledge constraints.

The detail implementation on ORBAC can be described as follows (see Fig 4):

(1) User K first logs in client A with his username, password.

(2) User K opens an application and creates a session number and sents it with his username to security manager B.

(3) Security manager B got it and creates a UR object to check role allocation table and returns all user K's allocated roles back to K, in the meantime, UR will create a session for K with his session id and username.

(4) K chooses suitable roles for his current application and sends them back to B.
(5) UR got it, then checks role hiearchy in the Basic security Architecture table and got all chosen roles' child roles, furthermore got all child roles' constraints and check whether the chosen roles violate role constraints. To prevent the problem of mutual exclusive roles, UR will following check the session table to see if there exists mutual exclusive problem after adding the chosen roles to the session table, if not, the roles will be activated and be added into the session table, .otherwise, this role application will be refused.
(6) After roles were authorized, the authorized roles will return back to A.
(7) Client A got the authorized role, then accesses the server C with his priviledge requirement.
(8) In the server C, an RP object will be create after the roles and K's priviledges is received,
then RP object got the authorized roles' priviledges and their constraints from the resource requirement table and judge whether the authorized roles corresponded with the priviledge constraints, if yes the priviledges will be authorized.
(9) Compare the authorized priviledges with K's required priviledge, if the require priviledges less than or equal to the authorized priviledges, K's access will be granted, otherwise, the access application will be refused.
After the application finished, session will be closed and the application session item on the session table will be deleted, also, UR and RP object will be destroyed.

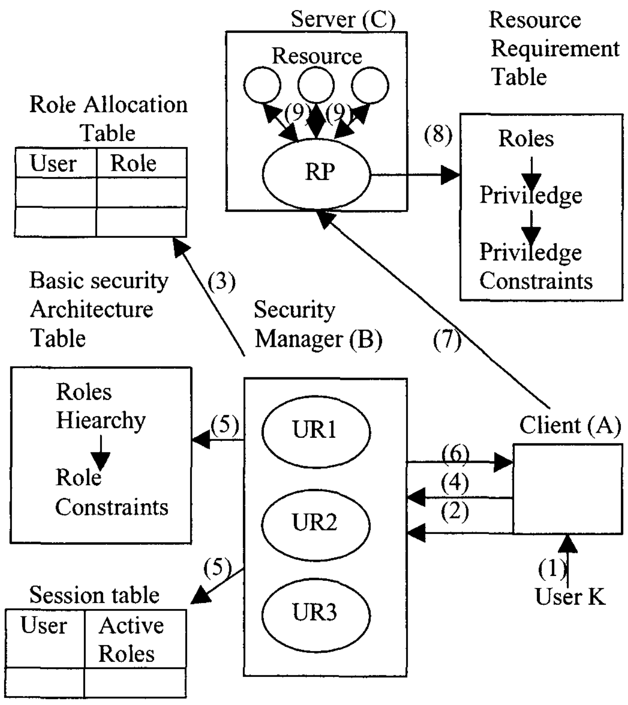

Fig 4. ORBAC implementation diagram

4. Conclusion:

In this paper we have presented an objected-oriented RBAC model(ORBAC). The driving motive of it is to simplify security policy administration. The session table was established to prevent the problem of seperation of duty, and it provides a way to prevent the domain security manager assign multiple exclusive role to a user at one time. Moreover, this paper also discussed some ORBAC and seperation of duty specifications.

Reference:

(1) Sandhu,R.S, Coyne, E.J., Feinseein, H.L., and Younman C. E., *Proceedings of the first ACM Workshop on Role-Based-Access Control* , ACM, 1996.

(2) Ravi Sandhu and Venkata Bhamidipati. The URA97 model for Role-based administration of user-role assignment. In T.Y. Lin and Xiaolei Qian, editor, *Database Security : Status and prospects.* North-Holland, 1997.

(3) Ravi sandhu and Venkata Bhamidipati, The ARBAC97 Model for Role-Based Administration of Roles: Preliminary Description and outline, *Second ACM workshop on Role-Based-Access-Control* , Fairfax, Virginia, USA, November, 6-7, 1997.

(4) Trent Jaeger, Frederquegiraud, A Role-Based Access Control Model for Protection domain Derivation and Management, *Second ACM Workshop on Role-Based-Access-Control*, Fairfax, Virginia, USA, November 6-7, 1997.

(5) R.S. Sandhu, Lattice-based access control . *Computer*, 26: 9-19, Nov 1993.

(6) D.Ferraiolo, J. Cugini, and D.R. Kulin. Role based access control: Features and motivacation. *In annual Computer security applications conference. IEEE Computer Society Press, 1995.*

(7) Sylvia sborn, Yuxiao Guo, Modeling users in role-based access control, *Fifth ACM workshop on Role-Based Access Control, Berlin, Germany,* July 26-27, 2000.

(8) Ravi sandhu, David Ferraiodo and Richard Kulin, The NIST Model for Role-Based Access Control: Towards a unified Standard, *Fifth ACM Workshop on Role-Based Access Control,* Berlin, Germany, July 26-27,2000.