An Object-Oriented RBAC Model for Distributed System

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:

In the distributed computing environments, users would like to share resources and communicate with each other to perform their jobs more efficiently. For better performance, it is important to keep resources and the information integrity from the unexpected use by unauthorized users. Therefore, there is a strong demand for the access control of distributed shared resources in the last few years. Role-Based-Access-Control (RBAC) has been introduced and has offered a powerful means of specifying access control decisions. In this paper, we propose an object-oriented RBAC model for distributed system (ORBAC), it efficiently represents the real world. Moreover, under the decentralized ORBAC management architecture, an implementation of the model has realized multiple-domain access control. Finally, statically and dynamically role authorization has been considered and a method to deal with the problem of seperation of duties has been presented.

Keywords

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

1. Introduction

Distributed systems are increasingly being used in commercial environments necessitating the development of trustworthy and reliable security mechanisms. 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 members of appropriate 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 in 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].

0-7695-1360-3/01 $10.00 © 2001 IEEE

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.

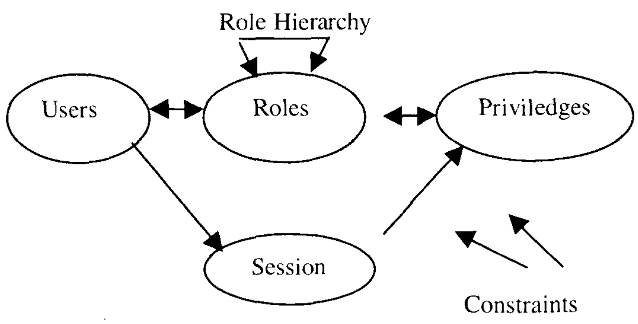

Fig 1: RBAC Model

3. Decentralized Security management Architecture

For distributed system, like Internet, centralized network administration is impossible and unflexible. The implementation of ORBAC model is based on a decentralized management architecture shown in Fig.2. A distributed environment has multiple different administration domains such as domain1, domain2, etc. The basic elements for each domain basically include client, server, domain security manager and foreign security manager. The main function of each element is described below .

Client: Accepts the requirements of a user to get access to local or foreign domain resources and returns the result to user.

Server: permits authorized accesses.

Domain Security Manager: Design and maintain the security policy (domain security policy and foreign security policy), authorize roles and access priviledges to its local domain users according to domain security policy.

Foreign Security Manager: In order to realize multi-domain access control, the foreign domain security manager is introduced, it accepts the requirements of the local domain user for foreign domain resources and returns the result. On the other hand, under the foreign security policies, it also supports foreign domain users accessing to its local domain resources.

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

The proposed object-oriented Role-Based-Access-Control model (ORBAC) described in Fig.3 fully realize the original RBAC model and can be implemented on a multi-domain distributed environment. In this section, we describe some basic specifications for ORBAC Model based on RBAC. 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.

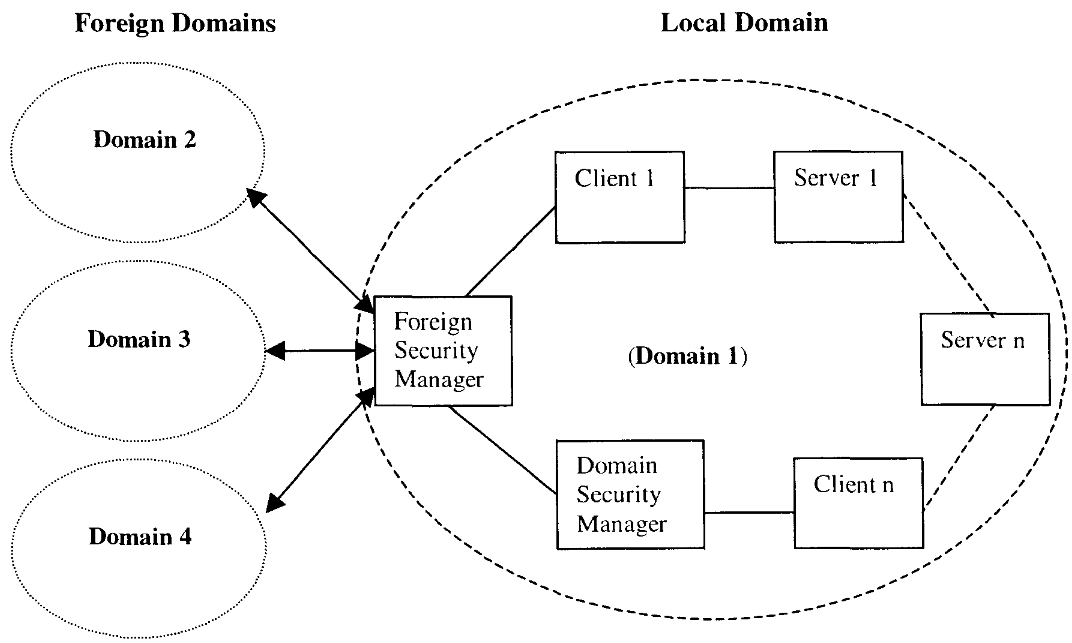

Figure 2. Decentralized Security Management Architecture

4.1 ORBAC: Basic elements and their specifications

4.1.1 User, Role, Priviledge, Session

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 mappings. functions:

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

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

S(t), the user/role mapping, which gives the subset of Role, every element of the subset is authorized for the User, t.

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

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

R(i), the Role/User mapping, which gives the subset of User, every element of the subset is authorized for the Role, i .

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.

Class Role is defined as :
- 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 which are used to check role authorization and solve role related problems, such as mutual exclusive problems.
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 shown in fig 3, we describe it by the following mapping functions:

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

$2^{Priviledge}$: represents any subset of the Priviledge .

T(l), the role/priviledge mapping, which gives the subset of Priviledge , every element of the subset is authorized for the role, l.

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

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

C(u), the priviledge /role mapping, which gives the subset of Role, every element of the subset is authorized for the priviledge , u.

Class priviledge is 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.

Class session is defined as:
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.

4.1.2 Fin and Fout

Fin and Fout are created by foreign security manager. Fin deals with foreign domain user accessing local domain resource, Fout deals with local domain user accessing foreign domain resource. A local domain user can get multiple foreign domain priviledges by Fout, a foreign domain user can get multiple local domain priviledges as well.

Class Fin is defined as:
Foreign domain user id: identify the foreign domain user.
Roles: all the role objects required by the foreign domain user.
The main function of the class is to accept foreign domain user's role requirements and evaluate them by the foreign security policy, return the authorized priviledges to the foreign domain user.

Class Fout is defined as:
Local domain user id: identify the local domain user.
Roles: all the role objects required by local domain user.
The main function of the class is to accept local domain user's role requirements and return the authorized priviledges to local domain user.

4.1.3 UR

The association class UR defines user assignment between users and roles, and class static UR and class dynamic UR describe that users can be statically or dynamically authorized during a session. In normal conditions, a user can be assigned many different static roles as it satisfied the principle of "Least Priviledge", which mean that a user can be assigned least roles to finish a certain task that is benefit for the system security. 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 but not both roles are activated at the same time. For a distributed environment, the activated roles can be dynamically assigned if they will not lead to the problem of seperation of duty. Moreover, 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.

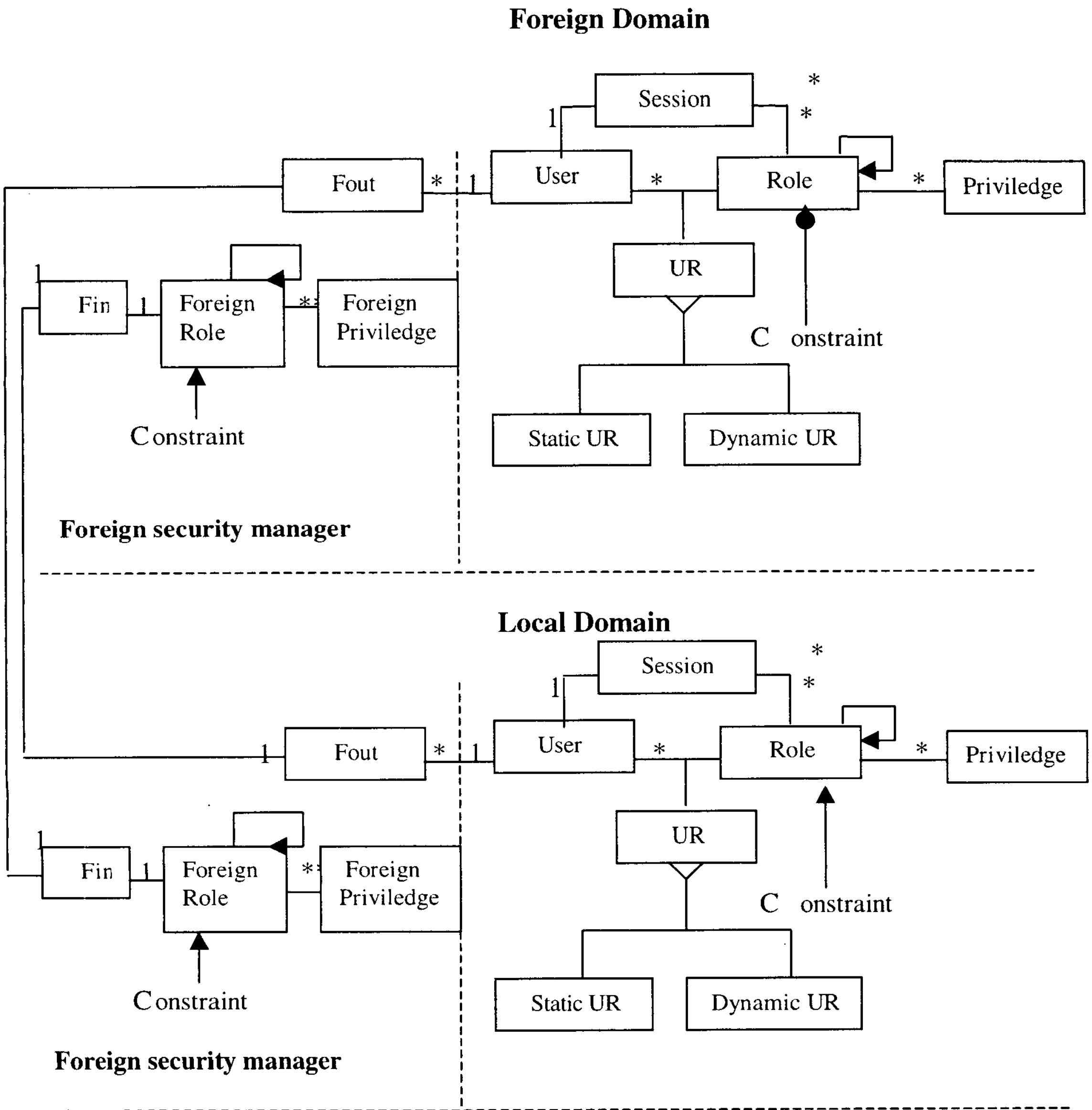

Fig 3: A block diagram of ORBAC modal

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 and priviledge authorization by calling constraints functions.

4.1.4 Constraints

ORBAC assigns constraints to user-role authorization, called constraints. Based on security policy, constraint defines which role or roles can be authorized to a valid user.

The constraint for mutual exclusive roles is a major part of the constraints. It can be used to enforce interest conflicts 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 two conflicted roles, the user is prohibited from being a member of another role. An efficient method has been presented in the next section. The constraint functions for mutual exclusive roles can be specified as follows:

(5) E : role $\times$ role

E[l, m : 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.

$(\forall\, l, m : \text{Role})(\exists t : \text{User})\ (l \neq m) \wedge E(l, m) \wedge (t \in R[l]) \Rightarrow t \notin R[m]$

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

$\forall(l, m : \text{Role})\ \exists(n)\ E(l, m) \Rightarrow \neg((l > m) \wedge (m > l))$

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

$\forall(l, m : \text{Role})\ \forall(n : \text{Role}) E(l, m) \Rightarrow (\neg\exists n)(l > n) \wedge (m > n))$

4.2 Dynamic properties

ORBAC dynamicproperties include role activation, priviledge execution and dynamic seperation of duties. Dynamic properties provide extended support for the principle of least priviledge. 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} : represents 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 : user $\times$ Priviledge $\rightarrow$ 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

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

(12) Role Assignment :

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

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

(13) Role Authorization :

Role authorization : a user's active role must be in the set of authorized roles for the user.

$(\forall t : \text{User})(\forall n : \text{Role})(n \in A[t] \Rightarrow n \in S[t])$

(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 for user t, roles below it are also included:

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

5. The general method for ORBAC implementation

The proposed ORBAC implementation diagram is shown in Fig 5. 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, UR will monitor all the active roles of each user on his sessions so that there is no mutual exclusive roles are activated simultaneously. The user object is issued to indicate all the roles (static and dynamic roles) assigned to each user by the domain security manager. The role object defines role hiearchy and constraints to present the relationship between roles and their constraints. Priviledge object defines the relationships between roles and their priviledges.
The detail implementation on ORBAC can be described as follows (see Fig 5):

- **Local user access local server**

Assume user K wants to access sever C
(1) User K logs in client A with his priviledge requirements.
(2) User K opens an application and creates a session number and sents it with his username to domain security manager B.
(3) B got it and creates a UR object such as UR1 to check user object and returns all K's allocated roles (static and dynamic 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 checks role hiearchy in the role object, search all chosen roles' child roles and their constraints, furthermore, get all the child roles which satisfy the constraints, . After checking every priviledge of the authorized child roles in priviledge object, authorized priviledges will be assigned to user K.
If there exists mutual exclusive constraints, object UR will check the session objects of K to see if there exists mutual exclusive problem after adding the chosen child roles to his session, if not, the authorized roles will be added , otherwise, this role application will be refused.
(6) After priviledges were authorized, a priviledge certificate D will be created (its format shown in fig 4) and sent to C alone with K's priviledge requirements.
(7) In server C, a proxy object will be create after the priviledge certificate and K's priviledges requirement is received, the main function of proxy is to judge whether every required priviledge is corresponded with D, if yes the required priviledge will be granted, otherwise, it will be refused.
(8) results return to K

After the application finished, session will be closed and the application session item on the session object will be deleted, also, UR and RP object will be destroyed.

- **Local User Access Foreign domain Server**

Assume user K intend to access foreign domain server P.
(9) User K provides a foreign role and priviledge requirement to its foreign security manager E.
(10) A Fout object was created and the role and priviledge requirement are sent to foreign security manager M.
(11) M create a foreign certificate R according to its security policy and sent to server P .
(12) P returns result to user K.

- **Foreign domain User access Local domain Server**

Assume user S want to access server C.
(13) User S accesses its foreign security manager M and provides role and priviledge requirement.
(14) M access to E and a Fin object is created.
(15) Fin checks constraints for foreign roles and creates a priviledge certificate T based on its security policy.
(16) T will be sent to server C along with the priviledge requirements of user S.
(17) C returns results to user S

Fromat of the priviledge certificate:

Message id	User id	Roles required	Authorized priviledges	Valid Time

Fig 4 priviledge certificate Message

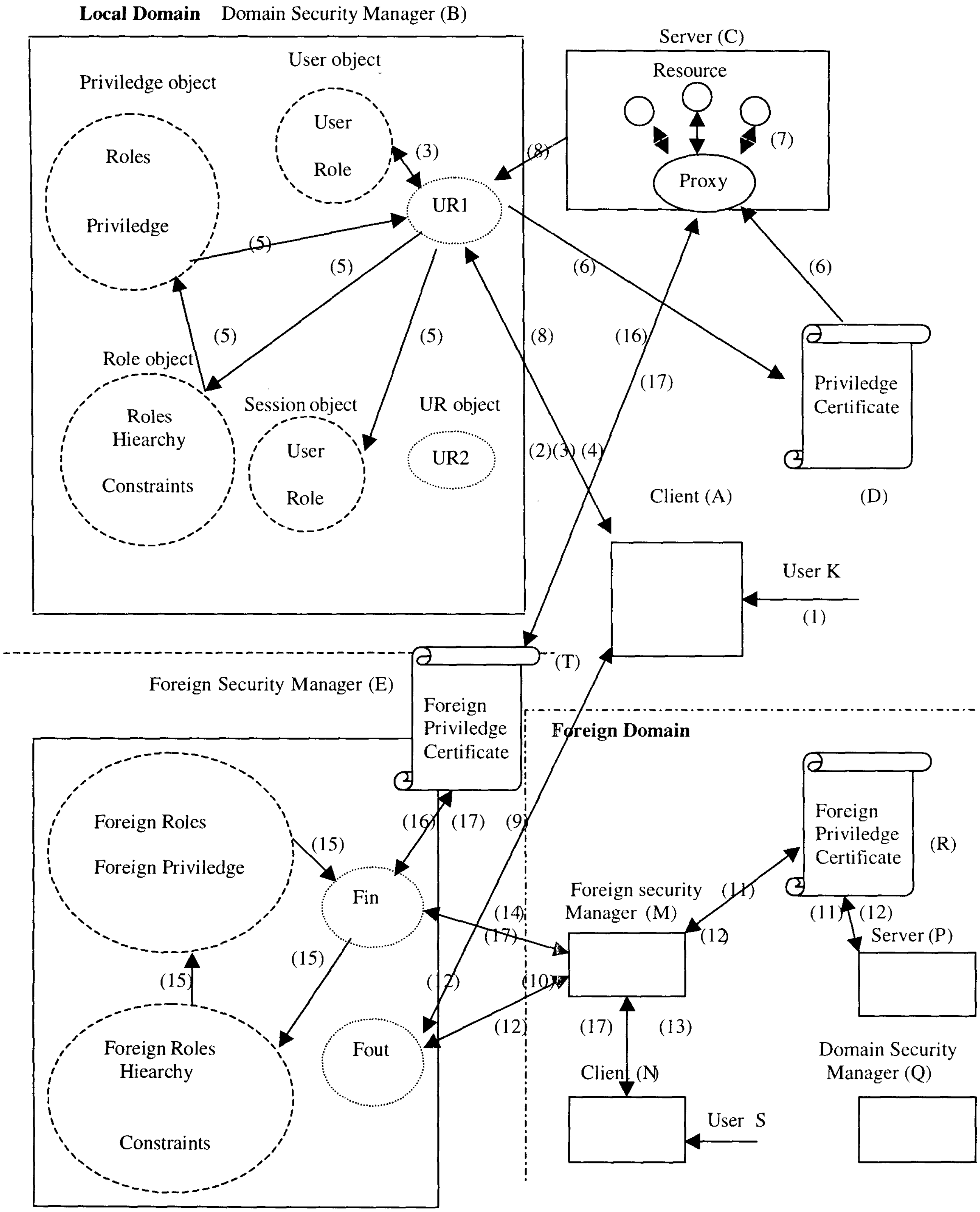

Fig 5. ORBAC implementation diagram

6. Conclusion:

In this paper we have presented an objected-oriented RBAC model (ORBAC). The driving motivation of it is to simplify security policy administration. We also proposed a decentralized security management architecture, based on it, we have realized multiple-domain access control. A new method is presented 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 duty 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.