ABDM: An Extended Flexible Delegation Model in RBAC*

Min Li Hua Wang
Department of Mathematics & Computing
University of Southern Queensland
Queensland, Australia
{limin, wang}@usq.edu.au

Abstract

Role-based access control (RBAC) is recognized as an efficient access control model and its delegation authority has been proved to be flexible and useful for information sharing on distributed environment. In today's highly dynamic distributed systems, collaboration is necessary for information sharing with others, so a user may want to delegate a collection of permissions, named an ability, to another user or all members of a group. Based on this fact, this paper builds a new ability-based delegation model (ABDM) within RBAC and develops its delegation algorithm. The framework includes both ability-based user-user delegation and user-group delegation. Further, we analyze delegation granting and revocation authorization. Compared with other existing delegation models, this ability-based delegation model provides great flexibility in authority management.

1. Introduction

Role-based access control (RBAC) is a promising access control technology for the modern computing environment [6, 4, 19]. In RBAC, a role is used to associate users and permissions. Permissions are associated with roles, and users are assigned to appropriate roles thereby acquiring the roles' permissions. As show in Figure 1, the relationships between users and roles, and between roles and permissions are many-to-many (i.e, a permission can be associated with one or more roles, and a role can be associated with one or more permissions). The security policy of the organization determines the user-role relationship, role-permission relationship and the allocation of each role's capabilities. The RBAC model supports the specification of several aspects:

a. User/role associations - the constraints specifying user authorization to perform roles;

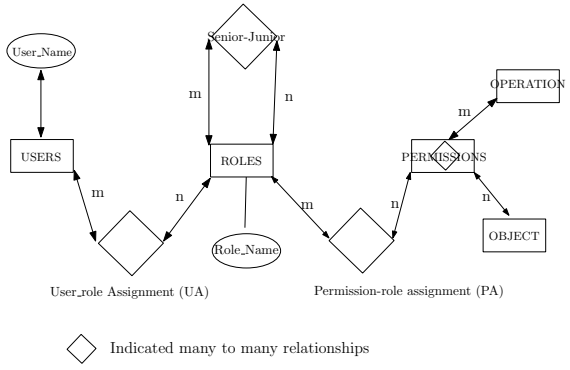

Figure 1: **RBAC relationship**

b. Role hierarchies - the constraints specifying which role may inherit all of the permissions of another role;

c. Duty separation constraints - there are user/role associations indicating conflict of interest;

c1. Static separated duty (SSD) - a constraint specifying that a user cannot be authorized for two different roles;

c2. Dynamic separation duty (DSD) - a constraint specifying that a user can be authorized for two different roles but cannot act simultaneously in both;

d. Cardinality - the maximum number of users allowed, i.e. how many users can be authorized for any particular role (role cardinality), e.g., only one manager.

The number of roles and users and permissions associated to roles in a large enterprise system can be hundreds or thousands. Managing these roles, users, permissions and their interrelationships is a vital challenge that is often highly decentralized and delegated to a small team of project groups. RBAC allows us to model security from the perspective of the organization, because we can align security modeling to the roles and responsibilities in the organization. Most large organizations have some business rules related to access control policy such as need-to-know, separation of duty, rotation of sensitive job positions, and so on.

*The research is support by an ARC Discovery Grant DP0663414.

Delegation of authority is an important one to implement the rules.

In access control systems, the delegation requirement arises when a user needs to act on another user's behalf to access resources. This might be only within a limited time, for example, a vacation, sharing resources temporarily with others, and so on. Otherwise users may perceive security as a hindrance and bypass it. With delegation, the delegated user has the privileges to react to situations or access information without referring back to the delegating user.

Role-based delegation model based on role-based access control (RBAC) has been proven to be a flexible and useful access control for information sharing on distributed collaborative environment [14]. The basic idea behind delegation is that some active entity in a system delegates authority to another active entity to carry out some functions on behalf of the former. Delegation in computer can be human to human, human to machine, machine to machine, and perhaps even machine to human. A number of models dealing with various aspects of delegation have been published [12, 10, 7, 13, 11, 8]. The RBDM0 [2, 3] and RDM2000 [17] model, in particular, are user-to-user delegation primarily based on roles. Furthermore, Wang *et al*. [15] proposed a role-based delegation model which support user-to-group delegation. Zhang *et al*. proposed permission-based delegation model (PBDM) [18], where a user can delegate a permission to the others accomplished by permission-role assignment and user-role assignment.

Actually, in many situations, a delegator wants to delegate a collection of permissions (named an ability) to delegatees. There may be some problems arising in the previous delegation models. For example, the permission of opening a bank account is composed of many different individual permissions, such as, accessing identification, social secure history, credit limits and so on. When the delegator wants to delegate his ability (opening an account) to others, it does not make sense if delegating only part of the permissions to delegatees, since the entire set is needed to do the task properly. If the number is huge, it may be a hard work to complete the delegation with PBDM, since each time only one permission can be assigned to a role in permission-based delegation model. In this paper, we propose a new delegation model, named ability-based delegation model, which can solve this problem easily. We focus exclusively on this ability-based delegation framework, which provides great flexibility in authority management.

The rest of this paper is organized as follows: In Section 2, we propose a developed assignment framework which includes group-role assignment and ability-role assignment. Our new introduced ability-based delegation model (ABDM) including delegation granting and revocation models are given in Section 3. Finally, we conclude the paper in Section 4.

2. Ability, Group and Authorization assignment

Role-based access control (RBAC) involves individual users being associated with roles as well as roles being associated with permissions (each permission is a pair of objects and operations). As such, a role is used to associated users and permissions. A user in this model is a human being. A role is a job function or job title within the organization associated with authority and responsibility. A permission is an approval of a particular operation to ba performed on one or more objects. When we want to open a bank account, many different individual permissions are involved. It does not make sense to assign only part of the permissions to a role, since the entire set is needed to do the task properly. The idea is that application developers package permissions into collections, named ability, which must be assigned together as a unit to a role. Once the notion of ability is introduced, by analogy there should be a similar concept on user side.

An *ability* is a collection of permissions that should be assigned as a single unit to a role. We denote B as the set of abilities.

A *group* is a collection of users who can accept a role assignment in the same times. Such a group can be viewed as a team. We denote G as the set of groups.

Different from the previous permission-role assignment, ability-role assignment is more difficult to be achieved. Since if there is a huge number of permissions involved in an ability, we have to do excessive jobs in order to finish the ability-role assignment, because each time we can just assign one permission to the role according to the previous permission-role assignment. So it is desired to assign all of the permissions in the ability to the role at once. The same situation happens when we want to define a group with a large number of users for role assignment. Since the function of an ability (or a group) is to collect permissions (or users) together so that administrators can treat them as a single unit. Assigning abilities to roles and groups to roles are therefore very much like permission-role assignment and user-role assignment. In this way, the problems stated above can be resolved easily.

A prerequisite condition is an expression using Boolean operators '$\wedge$' and '$\vee$' on terms of the form r and $\overline{r}$ where r is a role and '$\wedge$' means 'and', '$\vee$' means 'or'. A prerequisite condition is evaluated for a user u by interpreting r to be true if $(\exists r' \geq r), (u, r') \in UA$ and $\overline{r}$ to be true if $(\exists r' \geq r), (u, r') \notin UA$, where UA is a set of user-role assignments.

Note that the notion of a prerequisite condition is identical to that, expect the boolean expression is now evaluated for membership and nonmembership of an ability (or a

group) in specified roles. For a given set of roles R, let CR denotes all prerequisite conditions that can be formed using the roles in R. AR is the set of administrative roles. This leads to the following definitions.

Definition 1: Ability-role assignment and revocation are, respectively authorized by

$can\_assigna \subseteq AR \times CR \times 2^R$

$can\_revokea \subseteq AR \times 2^R$

The meaning of $can\_assigna(x, y, Z)$ is that a member of the administrative role x can assign an ability whose current membership satisfies the prerequisite condition y to regular roles in range Z. The meaning of $can\_revokea(x, Y)$ is that a member of the administrative role x can revoke membership of an ability from any regular role $y \in Y$.

Definition 2: Group-role assignment and revocation are, respectively, authorized by

$can\_assigng \subseteq AR \times CR \times 2^R$

$can\_revokeg \subseteq AR \times 2^R$

The meaning of $can\_assigng(x, y, Z)$ is that a member of the administrative role x can assign a group whose current membership satisfies the prerequisite condition y to regular roles in range Z. The meaning of $can\_revokeg(x, Y)$ is that a member of the administrative role x can revoke membership of a group from any regular role $y \in Y$. To identify a role range within the role hierarchy, the following closed and open intervals are used.

$[x, y] = \{r \in R | x \geq r \wedge r \geq y\}$ $(x, y] = \{r \in R | x > r \wedge r \geq y\}$
$[x, y) = \{r \in R | x \geq r \wedge r > y\}$ $(x, y) = \{r \in R | x > r \wedge r > y\}$.

3. Ability-based Delegation model (ABDM)

In this section we propose our flexible ability-based delegation model which supports role hierarchy. An intuitive overview of this model is described first, and then a formal definition will be presented.

3.1. Ability-based user-user Delegation

The central idea of this model is to create one or more delegation roles (DTR), and assign abilities to them. In RBAC, permissions are associated with roles, and users are assigned to appropriate roles thereby acquiring the roles' permissions. For example, in Figure 2 John who is in role PL acquires an ability b and a permission p. If John wants to delegate his ability b to Jenny, he can delegate according to following three phases.

1. John creates a temporary delegation role D_1.

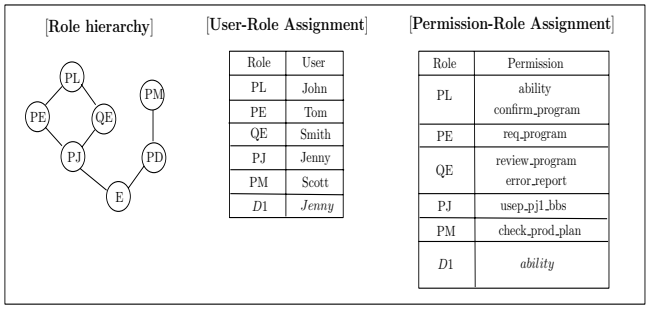

Figure 2: **Example of ability delegation**

2. John assigns the ability b to D_1 with ability-role assignment.

3. John assigns Jenny to D_1 with user-role assignment.

Roles in DTR are distinct from regular roles (RR). DTR cannot be assigned to any other roles, because it will generate invalid ability inheritance in role hierarchy. Therefore, roles in this model are partitioned into regular roles (RR) and delegation roles (DTR). This partition induces a parallel partition of UA and BA which are user-role assignment and ability-role assignment respectively. UA is separated into user-regular role assignment (UAR) and user-delegation role assignment (UAD). BA is similarly separated into ability-regular role assignment (BAR) and ability-delegation role assignment (BAD). Delegation role can be placed in the regular role hierarchy when the delegated ability includes all the permissions of a delegating role, otherwise it is isolated from the hierarchy. Delegation role cannot have any senior regular role if it is placed in the role hierarchy, since delegated abilities cannot be inherited through role-role hierarchy.

We have the following components for ability-based delegation model:

Sets: U, B, R, RR, DTR are sets of users, abilities, roles, regular roles, and delegation roles respectively.

$R = RR \cup DTR$

$UAR \subseteq U \times RR$ is a user to a regular role assignment relation.

$UAD \subseteq U \times DTR$ is a user to a delegation role assignment relation.

$UA = UAR \cup UAD$

$BAR \subseteq B \times RR$ is an ability to a regular role assignment relation.

$BAD \subseteq B \times DTR$ is an ability to a delegation role assignment relation.

$BA = BAR \cup BAD$

Abilities: $R \rightarrow 2^B$ is a function mapping a role to a set of abilities.

$Abilities(r) = Abilities\_R(r) \cup Abilities\_D(r)$

where

$Abilities\_R(r) = \{b | \exists r' < r, (b, r') \in BAR\}$

$Abilities\_D(r) = \{b|\exists r' < r, (b, r') \in BAD\}$

$senior(r) : P \rightarrow 2^R$, a function mapping a role to all its senior roles in role hierarchy.

$\forall dtr \in DTR, senior(dtr) \cap RR = \emptyset$: for each delegation role there is no senior regular role.

$own(u) : U \rightarrow 2^{DTR}$ and $\nexists(u_1, u_2 \in U, dtr \in DTR), (u_1 \neq u_2) \wedge (dtr \in own(u_1) \wedge dtr \in own(u_2))$, a function mapping a user to a set of delegation roles which he/she created.

$ability\_d(r) : DTR \rightarrow 2^B$, a function mapping a delegation role to a set of abilities.

$ability^*(u)$: a function mapping a user to a set of abilities with BAD.

$ability^*(u) = \{b \in B|\exists r \in DTR, (u, r) \in UAD \wedge (b, r) \in BAD\}$

A delegation relation in ability-based user-user delegation model is a constraint on UAD and BAD.

3.2. Ability-based User-Group delegation

Now we analyze group delegation. In some cases, we may need to define whether or not a user can delegate an ability to a group and how many times, or up to the maximum delegation depth. We only analyze one-step group delegation in this paper which means the maximum delegation path is 1. Figure 3 shows the role hierarchy structure of RBAC in an example of a problem-oriented system POS which has two projects. In Figure 4 Tony who is in role Co2 acquires all the permissions and abilities of role Co2. Now Tony wants to delegate one of his abilities to $Project$ 1, which means Tony wants to delegate the ability to all people involved in $Project$ 1. According to the ability-based user-user delegation, in the third step we have to use user-role assignment. If the number of users in $Project$ 1 is small, it may be easy to finish, otherwise, it is hard to finish the work as each time just one user can be assigned to the role. It will be time-consuming if based on user-user delegation. To solve the problem, we propose ability-based group delegation framework , in which Tony can finish the delegation according to following steps:

1. Tony creates a temporary delegation role D_1.

2. Tony assigns the ability b to D_1 with ability-role assignment.

3. Tony assigns all user of $Project$ 1 to D_1 with group-role assignment.

In fact, ability-based group delegation is achieved by ability-role assignment and group-role assignment. Same as before, a delegation role cannot have any senior roles since delegated abilities cannot be inherited. Roles in this model are partitioned into regular roles RR and delegation roles (DTR). This partition induces a parallel partition of GA and BA which are group-role assignment and ability-role assignment respectively. GA is separated into group-

Delegation Algorithm

Input: delegator u, ability b^*, delegatee.
Output: true if delegator u can delegate an ability b^* to delegatee; false otherwise.

Step 1: /*Delegator creates a temporary delegation role D_1 */
Suppose $(u, r) \in UA$ which means that delegator u is in role r,
Let $Pool_{withrole\_r} = \{p|(p, r) \in PA\} \cup \{b|(b, r) \in BA\}$,
/*$Pool_{withrole\_r}$ is the set of all permissions and abilities which are related to role r*/
If $\{b^*\} = Pool_{withrole\_r}$ /*role r only has an ability b^**/
Delegator can let role r to be the temporary delegation role D_1.
go to Step 3
If $\{b^*\} \subset Pool_{withrole\_r}$
/*role r not only has the ability b^* but also other permissions*/
Delegator creates a temporary delegation role D_1.
go to Step 2
else
return false and stop.
Step 2: /*whether the delegator can assign the abilityb^* to delegation role D_1 or not*/
Let $S = \pi_{RoleRange}(\sigma_{delegator}(can\_assigna))$
/*S is the role range where the ability b^* can be assigned to*/
If $D_1 \in S$, /* the delegation role is in the role range*/
go to Step 3.
/* the ability b^* can be assigned to D_1 by delegator*/
else
return false and stop.
Step 3: /*whether the delegator can assign the delegatee to the delegation role D_1*/
Suppose the delegatee is a user u',
Let $S = \pi_{RoleRange}(\sigma_{delegator}(can\_assign))$,
/* S is the role range where the user can be assigned to*/
If $D_1 \in S$, /* the delegation role is in the role range*/
the user u' can be assigned to the delegation role D_1.
Suppose the delegatee is a group g,
Let $S = \pi_{RoleRange}(\sigma_{delegator}(can\_assigng))$,
/* S is the role range where the group can be assigned to*/
If $D_1 \in S$, /* the delegation role is in the role range*/
the group g can be assigned to the delegation role D_1.
return true;
else
return false.

RoleName	Prereq.Condition	M
HO2	[*AP, Ho1*]	1
Co1	*CS*	2

Table 1: **Example of** $can\_delegatea$

RoleName	RoleRange
Ho1	[*Co1, CS*]
Re1	[*AP, AP*]

Table 2: **Example of** $del\_revokea$

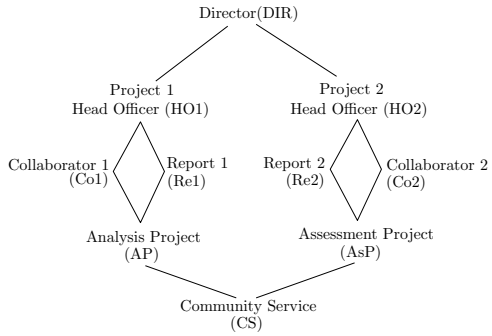

Figure 3: **Role hierarchy in** POS

[User-Role Assignment]

role	user
Co2	Tony
D1	*Group* {all users in *project 1*}

[Permission-Role Assignment]

role	permission
Co2	*ability* error_report
D1	*ability*

Figure 4: **Example of Group Delegation**

regular role assignment (GAR) and group-delegation role assignment (GAD). BA is similarly separated into ability-regular role assignment (BAR) and ability-delegation role assignment (BAD). Hence we have the following elements and functions in group delegation:

Sets: U, G, B, R, RR, DTR are sets of users, groups, abilities, roles, regular roles, and delegation roles respectively.

$R = RR \cup DTR$

$GAR \subseteq G \times RR$ is a group to a regular role assignment relation.

$GAD \subseteq G \times DTR$ is a group to a delegation role assignment relation.

$GA = GAR \cup GAD$.

$BAR \subseteq B \times RR$ is an ability to a regular role assignment relation.

$BAD \subseteq B \times DTR$ is an ability to a delegation role assignment relation.

$BA = BAR \cup BAD$

Abilities: $R \rightarrow 2^B$ is a function mapping a role to a set of abilities.

$Abilities(r) = Abilities\_R(r) \cup Abilities\_D(r)$

where

$Abilities\_R(r) = \{b | \exists r' < r, (b, r') \in BAR\}$

$Abilities\_D(r) = \{b | \exists r' < r, (b, r') \in BAD\}$

$senior(r) : P \rightarrow 2^R$, a function mapping a role to all its senior roles in role hierarchy.

$\forall dtr \in DTR, senior(dtr) \cap RR = \emptyset$: for each delegation role there is no senior regular role.

$own(u) : U \rightarrow 2^{DTR} and \nexists(u_1, u_2 \in U, dtr \in DTR), (u_1 \neq u_2) \wedge (dtr \in own(u_1) \wedge dtr \in own(u_2))$, a function mapping a user to a set of delegation roles which he/she created.

$ability\_d(r) : DTR \rightarrow 2^B$, a function mapping a delegation role to a set of abilities.

$ability^*(g)$: a function mapping a group to a set of abilities with BAD.

$ability^*(g) = \{b \in B | \exists r \in DTR, (g, r) \in GAD \wedge (b, r) \in BAD\}$

A delegation relation in ability-based user-group delegation model is a constraint on GAD and BAD.

The delegation algorithm described above provides a way for the delegator to delegate an ability to the desired delegatee.

3.3. Ability-based Delegation Authorization

In this section, we develop the delegating and revocation models. The goal of the delegation authorization is to impose restrictions on which role can be delegated to whom. Here, we partially adopt the notation of prerequisite condition from [14] to introduce delegation authorization in the delegation framework.

Definition 7: $can\_delegatea$ is a relation of $RR \times CR \times M$ where RR, CR, M are sets of regular roles, prerequisite conditions, and maximum delegation depth, respectively.

The meaning of $can\_delegatea(r, cr, m)$ means a delegator who has regular role r can delegate an ability to any user or group whose current entitlements in role satisfy the prerequisite condition cr without exceeding the maximum delegation depth m. Table 1 shows the $can\_delegatea$ relations with the prerequisite conditions in the POS example. The meaning of $can\_delegatea(Ho2, [AP, Ho1], 1)$ is that a user of role $Ho2$ can delegate his/her ability to a group in which users are members of either role AP, or $Co1$, or $Re1$, or $Ho1$. In addition, the delegated ability cannot be re-delegate to other users or groups with the maximum depth of delegation is 1. The second tuple authorizes that a user of role $Co1$ can assign an ability to another user who has CS role and the delegated ability can be re-delegate to other users or groups with the maximum depth of delegation is 2.

Definition 8: An ability-based delegation revocation is a relation $del\_revokea \subseteq RR \times 2^R$, where RR is the set of regular roles.

The meaning of $(x, Y) \subseteq del\_revokea$ is that a delegator who has regular role x can revoke relationship of a user or group from any role $y \in Y$, where Y defines the range of revocation. Table 2 gives the $del\_revokea$ relation in Figure 3. The first tuple shows that the delegator who has the

role $Ho1$ can revoke a delegation relationship of a group from any role in $[Co1, CS]$. The second tuple shows that he delegator who has the role $Re1$ can revoke a delegation relationship of a user from role in AP.

Actually, the revocation process can be finished through any of the following cases:

1. Revoke the user-delegation role assignment or group-delegation role assignment.
2. Revoke the ability-delegation role assignment.
3. Revoke delegation role.

4. Conclusion

In this paper, we popularize user to group and permission to ability. Based on this, we proposed a flexible ability-based delegation model and developed according delegation algorithms. Moreover, we have analyzed the delegating framework including delegating authorization and revocation with constraints on ability-based delegation. The work presented in this paper has significantly extended previous work, which provides a flexible and useful management of delegation authority in role-based access control environment.

References

[1] M. Abadi, M. Burrows, B. Lampson and G. Plotkin, A calculus for access control in distributed system, *ACM Trans. Program. Lang. Syst*. 15(4), 706-734, 1993.

[2] E. Barka and R. Sandhu, Framework for Role-Based Delegation Models, *Proc of 16th Annual Computer Security Application Conference (ACSAC 2000)*. December, 2000.

[3] E. Barka and Ravi Sandhu, A Role-Based Delegation Model and Some Extensions, *Proc. of 23rd National Information Systems Security Conference (NISSC 2000)*. December, 2000.

[4] E. Bertino, E. Ferrari, and Vijay Atluri. Specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security*, 2(1),February 1999.

[5] M. Blaze, J. Feigenbaum, J. Ioannidis and A. Keromytis. The role of trust management in distributed systm security, *Security Internet Programming* pp. 185-210.

[6] D. F. Ferraiolo, J. F. Barkley, and D. Richard Kuhn. A role based access control model and reference implementation within a corporate intranet. *ACM Transactions on Information and System Security*, 2(1), February 1999.

[7] M. Gasser, E. McDermott, An Architecture for practical Delegation in a Distributed System, *IEEE Computer Society Symposium on Research in Security and Privacy*. May, 1990.

[8] C. Goh and A. Baldwin, Towards a more Complete Model of Role, *Proc. of 3rd ACM Workshop on Role-Based Access Control*. October, 1998.

[9] N. Li and B. N. Grasof, A practically implementation and tractable delegetion logic, *IEEE Symposium on Security and Privacy*, pp. 27-42.

[10] J. D. Moffett. Delegation of Authority Using Domain Based Access Rules, *PhD Thesis*. Dept of Computing, Imperial College, University of London. 1990.

[11] N. Nagaratnam and D. Lea, Secure Delegation for Distributed Object Environments, *USENIX Conference on Object Oriented Technologies and Systems*. April, 1998.

[12] R. Sandhu, V. Bhamidipati and Q. Munawer, The ARBAC97 Model for Role-Based Administration of Roles, *ACM Transactions on Information and System Security*, Volume 2, Number 1, February, 1999.

[13] L. A. Stein. Delegation Is Inheritance, *Proc. of Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'87)*. October, 1987.

[14] H. Wang, J. Cao, Y. Zhang, Formal authorization approaches for permission-role assignment using relational algebra operations, *Proceedings of the 14th Australasian Database Conference*, Feb. 2-7, 2003, Adelaide, Australia, Vol. 25, No.1, pages:125-134.

[15] H. Wang, J. Li, R. Addie, S. Dekeyser and R. Watson, A framework for Role-based group delegation in distributed environment, *the 29th Australasian Computer Science Conference (ACSC2006)*, Australian Computer Society, Hobart, Australia, 2006.

[16] H. Wang, J. Cao, Y. Zhang, A Consumer Anonymity Scalable Payment Scheme with Role Based Access Control, 2nd *International Conference on Web Information Systems Engineering (WISE'2001)*, Dec. 3-6, 2001, Kyoto, Japan.

[17] L. Zhang, Gail-Joon Ahn, and Bei-Tseng Chu, A rule-based Framework for Role-Based Delegation, *Proc. 6th ACM Symposium on Access Control Models and Technologies (SACMAT 2001)*, May, 2002.

[18] X. Zhang, Sejong Oh, and R. Sandhu, PBDM: A Flexible Delegation Model in RBAC, *8th ACM Symposium on Access Control Models and Technologies (SACMAT)*, Como, Italy, June 2-3, 2003: 149-157.

[19] M. Zurko, R. Simon, and T. Sanlippo. A user-centered modular authorization service built on an rbac foundation. *IEEE Symposium on Research in Security and Privacy*, pages 57-71, Oak-land, CA, May 1999.