An Effective Role Administration Model Using Organization Structure

SEJONG OH
Dankook University
and
RAVI SANDHU and XINWEN ZHANG
George Mason University

Role-based access control (RBAC) is a well-accepted model for access control in an enterprise environment. When we apply RBAC model to large enterprises, effective role administration is a major issue. ARBAC97 is a well-known solution for decentralized RBAC administration. ARBAC97 authorizes administrative roles by means of role ranges and prerequisite conditions, where prerequisite conditions effectively work as a restricted pool for administrative roles to pick users or permissions. Although attractive and elegant in their own right, these mechanisms have significant shortcomings. In this paper, we propose an improved role administration model named ARBAC02 to overcome the weaknesses of ARBAC97. ARBAC02 introduces the concept of organization structure for defining user and permission pools independent of roles and role hierarchies, with a refined prerequisite condition specification. In addition, we present a bottom-up approach of permission-role administration in contrast to the top-down approach in ARBAC97. As a general solution, we illustrate the applications of organization structured-based security administration with other access control models, such as access control list model and lattice-based access control model.

Categories and Subject Descriptors: D.4.6 [**Operating Systems**]: Security and Protection—*Access controls*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Unauthorized access*

General Terms: Security, Human Factors

Additional Key Words and Phrases: Access control, role-based access control, RBAC, role administration

This research was partially supported by the National Science Foundation.
An earlier version of this paper appeared under the title "A Model for Role Administration Using Organization Structure" in the *Proceedings of 7th ACM Symposium on Access Control Models and Technologies*, Monterey, CA, 2002.
Authors' addresses: S. Oh, Department of Computer Science, Dankook University, San 29 Anseo-dong, Chonan, Chungnam, 330-714, South Korea; email: sejongoh@dankook.ac.kr; R. Sandhu and X. Zhang, Department of Information and Software Engineering, George Mason University, 4400 University Dr., MSN 4A4, Fairfax, VA 22030; email: {sandhu,xzhang6}@gmu.edu.

1. INTRODUCTION

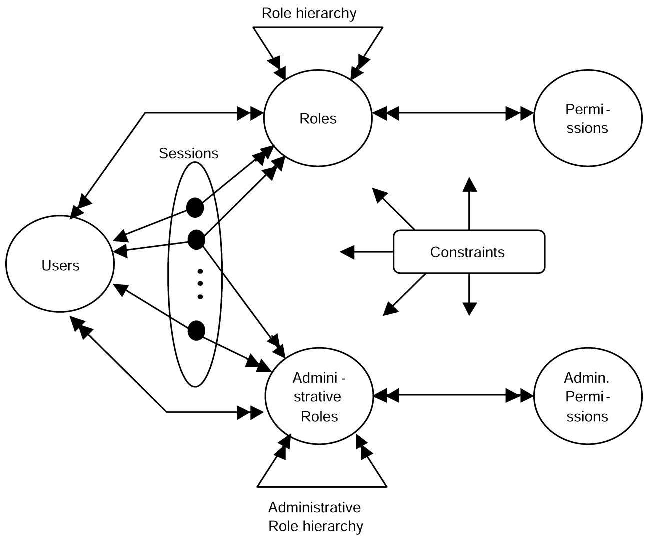
Access control is a central concern for information security in enterprises. Many access control models have been developed for different security policies, such as discretionary access control (DAC), mandatory access control (MAC), and role-based access control (RBAC). RBAC is a well-studied and increasingly commonplace technology for this purpose. In RBAC, access rights are associated with roles and users are assigned with appropriate roles, thereby acquiring the corresponding permissions. The notion of role is an enterprise or organizational concept. Therefore RBAC allows us to model security from the perspective of an organization, because we can align security modeling to roles and responsibilities. RBAC model has been shown to be "policy-neutral" in the sense that using role hierarchies and various constraints, a wide range of security policies can be expressed, including discretionary access control (DAC), mandatory access control (MAC), and user-specific access control [Osborn et al. 2000; Joshi et al. 2001].

In general, the increasing size and diversity of organizations makes for complex security problems. In large enterprise-wide systems, the number of roles can be in hundreds or thousands and users can be in tens or hundreds or thousands. Managing these roles, users, and their interrelation is a formidable task, which is often highly centralized in a small team of security administrators. A significant motivation behind RBAC is to simplify security administrations. An appealing possibility is to use RBAC itself to manage RBAC to provide further administrative convenience, especially in decentralizing administrative authority, responsibility, and tasks [Sandhu and Bhamidipati 1997b]. ARBAC97 (administrative RBAC '97), which is based on the RBAC96 model [Sandhu et al. 1996], shown in Figure 1, allows decentralized administration of user-role assignments (URA97), permission-role assignments (PRA97), and role-role assignments (RRA97).

In spite of the advantages and elegance of ARBAC97, it has some significant shortcomings and undesirable side effects. The main point of decentralized RBAC administration is to control the scope of each administrative role's administration domain (or boundary). For this purpose, ARBAC97 uses role ranges and prerequisite conditions. In particular, prerequisite roles are used as user and permission pools for administrative roles. This approach has some weaknesses because of undesirable coupling of roles and role hierarchies with user and permission pools. As we discuss in Section 2.2, URA97 includes the problems of multistep user assignments, duplicated information of user-role assignments, and restricted compositions of user pools, and PRA97 can have undesirable authorization flows, as well as the same problems as URA97.

This paper analyzes the weaknesses of ARBAC97 model and proposes an improved administration model named ARBAC02. ARBAC02 retains the main features of ARBAC97 and adds new components of organization structures as user and permission pools. We improve URA97, PRA97, and RRA97 models in ARBAC97 by using the new components. Furthermore, we show the applications of organization structure-based security administration in other access control models.

The rest of this paper is organized as follows. Section 2 presents our motivations and briefly reviews ARBAC97 and describes its weaknesses. Section 3



—sets: U, R, AR, P, AP, S for sets of users, (regular) roles, administrative roles, (regular) permissions, administrative permissions, and sessions, respectively.

—$UA \subseteq U \times A$: user-role assignments
$AUA \subseteq U \times AR$: user-administrative role assignments

—$PA \subseteq P \times R$: permission-role assignments
$APA \subseteq AP \times AR$: permission-administrative role assignments

—$RH \subseteq R \times R$: role hierarchy

—$ARH \subseteq AR \times AR$: administrative role hierarchy

—$user : S \rightarrow U$, a function mapping a session to a single user

—$roles : S \rightarrow 2^{R \cup AR}$, a function mapping a session to a set of roles:
$roles(s) \subseteq \{r : R \mid (\exists r' \geq r) \cdot [(user(s), r') \in UA \cup AUA]\}$

—$permissions : R \rightarrow 2^{P \cup AP}$, a function mapping a role to a set of permissions:
$permissions(r) = \{p : P \mid (\exists r'' \leq r) \cdot [(p, r'') \in PA \cup APA]\}$

—collection of constraints

Fig. 1. Summary of RBAC96 model.

introduces concept of organization structure as a candidate for user and permission pools. Section 4 presents URA02 and PRA02 and their advantages over ARBAC97. In Section 5 we apply organization structure concept to improve RRA97. Section 6 presents some examples of applying the organization structure concept to other access control models. Specifically, we describe how our approach is applied to access control list (ACL) and lattice-based access control (LBAC). Some related work is presented in Section 7; Section 8 concludes this paper.

2. MOTIVATION

2.1 Summary of ARBAC97 Model

ARBAC97 has three components: URA97 concerned with user-role administration, PRA97 with permission-role administration, and RRA97 with role-role

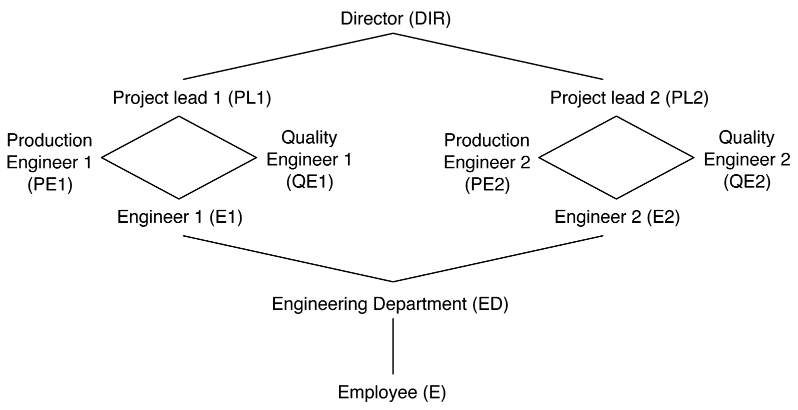


Fig. 2. An example of regular role hierarchy.

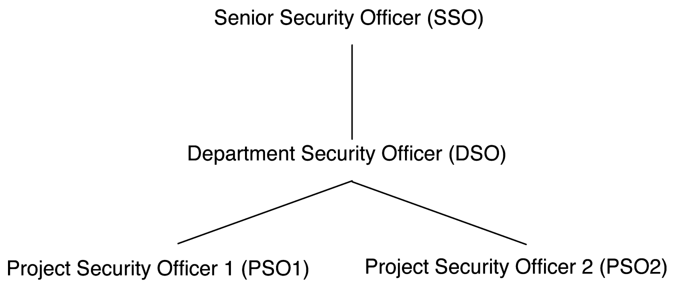

Senior Security Officer (SSO)
Department Security Officer (DSO)
Project Security Officer 1 (PSO1)
Project Security Officer 2 (PSO2)

Fig. 3. An example of administrative role hierarchy.

administration. Detailed motivations and rationale for URA97 and PRA97 are given in previous papers [Sandhu et al. 1996, 1999; Sandhu and Bhamidipati 1997a, 1999]. RRA97 is discussed in Sandhu and Munawer [1998]. Here we briefly explain each of them by an example using the regular role hierarchy and administrative role hierarchy of Figures 2 and 3, respectively, which have been frequently used in previous literatures.

2.1.1 *URA97 Model.* URA97 has two components, one dealing with the assignments of users to roles (the grant model) and the other with the revocations of user memberships (the revocation model). User-role assignments are controlled by a set of $can\text{-}assign(s, y, z)$ predicates, where x is an administrative role, y is a prerequisite condition, and z is a role range. For example,

$$can\text{-}assign(PSO1, ED, \{E1\})$$

means that a member of administrative role $PSO1$ (or a member of an administrative role senior to $PSO1$) can assign a user, who has membership of ED, to be a member of the regular role $E1$. The prerequisite condition is a boolean expression of prerequisite roles and/or constraints. For example, in prerequisite condition $E1 \wedge \overline{QE1}$, $E1$ is a prerequisite role and $\overline{QE1}$ is a constraint. The prerequisite condition $E1 \wedge \overline{QE1}$ denotes users who belong to $E1$ and do not belong to $QE1$.

Table I. Examples of *can-assign* in URA97

Admin. Role	Prereq. Condition	Role Range
$PSO1$	ED	$[E1, E1]$
$PSO1$	$E1 \wedge \overline{QE1}$	$[PE1, PE1]$
$PSO1$	$E1 \wedge \overline{PE1}$	$[QE1, QE1]$
$PSO2$	ED	$[E2, E2]$
$PSO2$	$E2 \wedge \overline{QE2}$	$[PE2, PE2]$
$PSO2$	$E2 \wedge \overline{PE2}$	$[QE2, QE2]$
DSO	$ED \wedge \overline{PL2}$	$[PL1, PL1]$
DSO	$ED \wedge \overline{PL1}$	$[PL2, PL2]$
DSO	ED	(ED, DIR)
SSO	E	$[ED, ED]$
SSO	ED	(ED, DIR)

Table II. Examples of *can-revoke* in URA97

Admin. Role	Role Range
$PSO1$	$[E1, PL1)$
$PSO2$	$[E2, PL2)$
DSO	(ED, DIR)
SSO	$[ED, DIR]$

User revocation in URA97 is controlled by a set of $can\text{-}revoke(x, z)$ predicates, where x is an administrative role and z is a role range. For example,

$$can\text{-}revoke(PSO1, \{PE1, QE1\})$$

means that a member of administrative role $PSO1$ (or a member of an administrative role senior to $PSO1$) can revoke a user, who is a member of $PE1$ or $QE1$. Table I and Table II show some examples of *can-assign* and *can-revoke* predicates in URA97. Here a role range is expressed by identifying the lower and upper boundary points of a role hierarchy, where a "(" or ")" means that the range does not include the corresponding boundary value, and "[" or "]" means that the range includes the corresponding boundary value. For example, $[E1, PL1)$ is equivalent to $\{E1, PE1, QE1\}$ in Table II with role hierarchy shown in Figure 2.

In URA97, role ranges and prerequisite roles (or conditions) are used to restrict the controlling scope of administrative roles. A role range is used as the boundary of target roles to be assigned to users and a condition is used as a domain to pick users. In other words, a prerequisite role in URA97 is used as a pool to select eligible users. Therefore we can replace "prerequisite role" with "user pool" as indicated in Figure 4 and user-role administration can be decentralized by assigning proper user pools and role ranges to administrators.

2.1.2 *PRA97 Model.* Similar to URA97, PRA97 has two components, one dealing with the assignments of permissions to roles and the other with the revocations of permissions. These two components are controlled by a set of $can\text{-}assignp(x, y, z)$ and $can\text{-}revokep(x, z)$ predicates, where x is an administrative

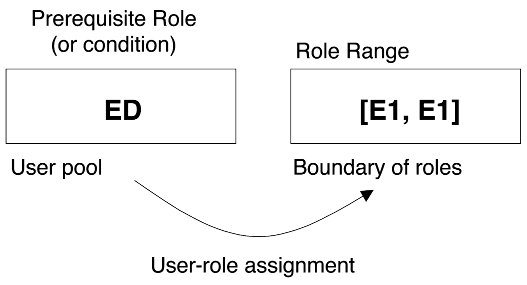


Fig. 4. Relationship between prerequisite role (or condition) and role range in URA97.

Table III. Example of *can-assignp* in PRA97

Admin. Role	Prereq. Condition	Role Range
DSO	DIR	$[PL1, PL1]$
DSO	DIR	$[PL2, PL2]$
$PSO1$	$PL1 \wedge \overline{QE1}$	$[PE1, PE1]$
$PSO1$	$PL1 \wedge \overline{PE1}$	$[QE1, QE1]$
$PSO2$	$PL2 \wedge \overline{QE2}$	$[PE2, PE2]$
$PSO2$	$PL2 \wedge \overline{PE2}$	$[QE2, QE2]$

Table IV. Example of *can-revokep* in PRA97

Admin. Role	Role Range
$PSO1$	$(E1, PL1)$
$PSO2$	$(E2, PL2)$
DSO	(ED, DIR)
SSO	$[ED, DIR]$

role, y is a prerequisite condition, and z is a role range. For example,

$$can\text{-}assignp(DSO, DIR, [PL1, PL1])$$

states that a member of administrative role *DSO* (or a member of an administrative role senior to *DSO*) can take any permission assigned to *DIR* and make it available to regular role *PL*1. Tables III and IV show some examples of *can-assignp* and *can-revokep* in the PRA97 model of our example.

Similar to URA97, a prerequisite role (or condition) in PRA97 is used as a pool to select eligible permissions. Therefore we can replace the term "prerequisite role" in PRA97 with "permission pool," as indicated in Figure 5. Permission-role administration can be decentralized by assigning proper permission pools and role ranges to administrators.

2.1.3 *RRA97 Model.* Besides URA and PRA, RRA is another way to assign access rights to users. Security administrators can modify the role hierarchy in an RBAC system, which leads to the change of the authorization structure. Therefore, modification of a role hierarchy should be managed very carefully. RRA97 uses a set of *can-modify* predicates to restrict modification of a role hierarchy by administrative roles. Table V shows an example of *can-modify*.

Because of the presence of role hierarchy, RRA has an undesirable side effect. As shown in Figure 6, suppose administrative role *PSO*1, who has control

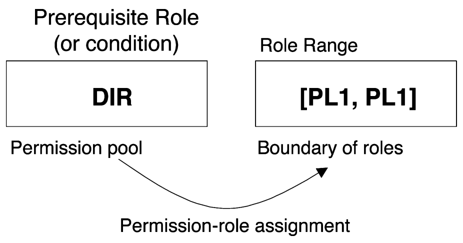


Fig. 5. Relationship between prerequisite role (or condition) and role range in PRA97.

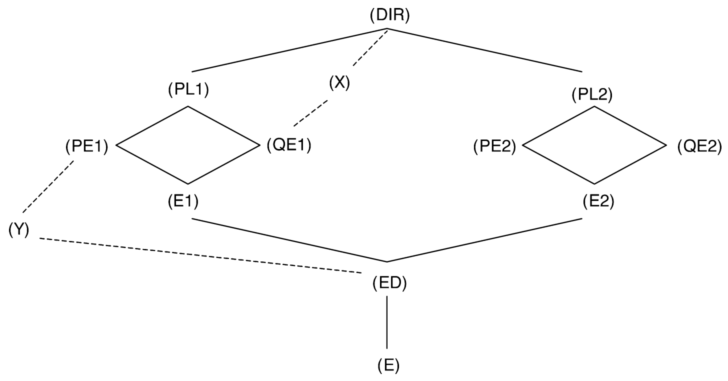


Fig. 6. An example of out of range impact.

Table V. Example of *can-modify* in RRA97

Admin. Role	Update Role Range
DSO	(ED, DIR)
$PSO1$	$(E1, PL1)$
$PSO2$	$(E2, PL2)$

authority over range $(E1, PL1)$, makes $PE1$ junior to $QE1$ by adding an edge. With the permission inheritance, this indirectly introduces a relationship between X and Y roles, which are out of the administrative range of $PSO1$.

2.2 Shortcomings of ARBAC97

The ARBAC97 model supports simple and decentralized security administrations. However, from a practical viewpoint, it has some significant shortcomings. An undesired side effect of RRA97 with regard to illegal permission flow has been shown in the previous section. In this section, we describe some weaknesses of URA97 and PRA97 with respect to prerequisite roles.

2.2.1 *Weaknesses of URA97*

2.2.1.1 *URA1 Multistep User Assignments.* Suppose that a newly employed engineer Tom is to be assigned to role $QE1$ in the environment of Figures 2 and 3 and Table I. To do so, Tom should be a member of $E1$, which is

Table VI. User-Role Assignment Information

No	Role	Assigned User
1	E	Tom
2	ED	Tom
3	$E1$	Tom
4	$QE1$	Tom

the prerequisite role of $QE1$. Before Tom can become a member of $E1$, he must be a member of the prerequisite role ED. Similarly, before Tom can be a member of ED, he should be a member of the prerequisite role E. To summarize, Tom's role assignment to $QE1$ must follow the order

> assign Tom to E $\rightarrow$ assign Tom to ED $\rightarrow$ assign Tom to $E1$ $\rightarrow$ assign Tom to $QE1$

This example shows that URA97 requires multiple steps for single user-role assignment and more assignment steps are required with higher destination roles in a role hierarchy, which may require the involvement of multiple security officers.

2.2.1.2 *URA2 Redundant User-Role Assignment (UA) Information.* Suppose Tom is a member of $QE1$ through the above assignment steps. Tom is, therefore, an explicit member of E, ED, $E1$, and $QE1$, and the corresponding information exists in the URA shown in Table VI, as the result of the multistep user assignments. In Table VI, tuples 1, 2, and 3 do not affect Tom's access rights because $QE1$ inherits the access rights of E, ED, and $E1$. From this point of view, these three tuples are redundant. They are required only for administrative purposes.

Table VI implies another problem. If DSO revokes Tom from role ED, there is no change in access rights of Tom. However, $PSO1$ may lose authority for Tom because he does not satisfy prerequisite condition in Table I. Thus, Table VI needs extra rules to manage it. Redundant user-role assignments also induce redundant user-role revocations, i.e., if we want to remove Tom from the system, we need four-step revocations according to Table VI.

2.2.1.3 *URA3 Restricted Composition of User Pools.* Suppose the organization in our example wants to maintain human resource pools H1, H2, and H3. Consider a new policy requiring that a production engineer should be selected from H1 and a quality engineer should be selected from H2. It is impossible to implement this new policy without changing the role hierarchy in ARBAC97. This is because that, in URA97, a user pool is based on some prerequisite roles and the prerequisite roles belong to a role hierarchy. This shows that a user pool in URA97 is restricted by the structure of roles and role hierarchies. Frequently a real-world application requires flexible user pools, which cause a highly complicated role hierarchy. The reason behind this problem is that URA97 has an unnecessary coupling between user pools and prerequisite roles with a role hierarchy.

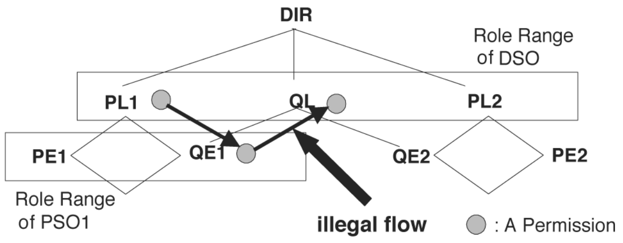


Fig. 7. Undesirable side effect in PRA97.

2.2.2 *Weaknesses of PRA97*

2.2.2.1 *PRA 1. Multistep Permission Assignments*

2.2.2.2 *PRA 2. Duplicated Permission-Role Assignment (PA) Information*

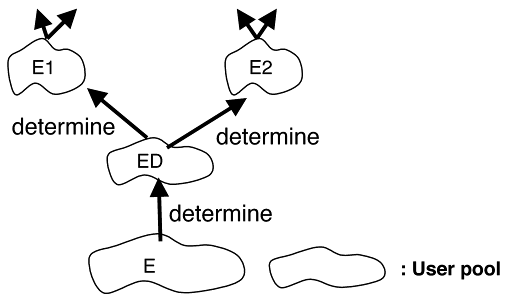
2.2.2.3 *PRA 3. Restricted Composition of Permission Pools.* We omit the explanation of problems PRA1, PRA2, and PRA3, because they are similar to URA1, URA2, and URA3, respectively.

2.2.2.4 *PRA 4. Weak Controlled Permission Assignments.* Suppose there exists a *can-assignp*$(SO1, R2, [R1, R1])$ predicate in a PRA97 model. $SO1$ can then assign any permission of $R2$ to $R1$. That is, there is no restrictions on assigning particular permissions of $R2$ to a user. PRA99 model [Sandhu and Munawer 1999] uses the concept of immobile membership to solve this problem. An immobile membership grants a user the authority to use the permissions of a role but does not make that user eligible for further role assignments. However, this approach requires additional information about permission pools.

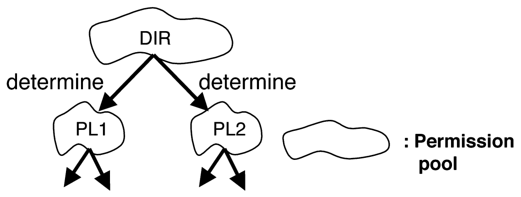
2.2.2.5 *PRA 5. Undesirable Side Effect of Permission Flows.* PRA5 is a corollary of PRA4. Consider the role hierarchy and role range shown in Figure 7. If *can-assignp*$(PSO1, PL1, [QE1, QE1])$ exists, then $PSO1$ can assign any permissions of $PL1$ to $QE1$. Consequently, QL inherits all permissions of $QE1$, since QL is a parent role of $QE1$. This means that $PSO1$ can move some permissions of $PL1$ to QL. However, QL is outside the role range of $PSO1$, so this permission flow is presumably undesirable.

The origins of these shortcomings aforementioned in ARBAC97 come from two aspects. First, user pools and permission pools in ARBAC97 are dependent on roles and the structure of role hierarchies. A prerequisite role is dependent on its lower or higher prerequisite roles. (As described above, a prerequisite role functions as a user pool or a permission pool.) As a result, all prerequisite roles form a dependency chain along the role hierarchy, as Figure 8 shows. This dependency is a strong restriction for constructing user pools and permission pools (URA3/PRA3). In addition, it results duplicate administrative work (URA1 and PRA1) and redundant data (URA2 and PRA2).

Second, because of the top-down nature of permission-role administration, security administrators can select any permissions from their prerequisite roles. This leads to the undesirable side effects of permission managements (PRA4 and PRA5).



(a) Dependency among user pools in URA97



(b) Dependency among permission pools in PRA97

Fig. 8. Dependencies in URA97/PRA97.

3. ORGANIZATION STRUCTURE AS USER/PERMISSION POOL

To overcome the weaknesses of ARBAC97 model, we use two strategies. First, we use an organization structure as a basis for defining user and permission pools instead of prerequisite roles in an organization. Second, based on the organization structure concept, we propose a bottom-up approach for permission-role assignments.

An organization is a group of employees (users) who perform business activities together in order to achieve a particular aim. For instance, a sales department includes a group of employees who perform sales works. For information systems development, organization is a good concept as a domain for analysis of business functions and activities. Most information system design methodologies use organization chart, such that shown in Figure 9 [IDS SHARE]. In spite of new types of organization structures, such as team-based, many organizations continue to have tree structures in organization charts. An organization structure is composed of organization units, each encompassing relevant users who work to achieve a mission. To achieve the given mission, each organization unit has a set of job functions or tasks. Users need to access information resources to perform the job functions or tasks. That is, job functions or tasks are related to permissions. From the perspective of access control, an organization unit works as "a group of users and permissions" to achieve a given

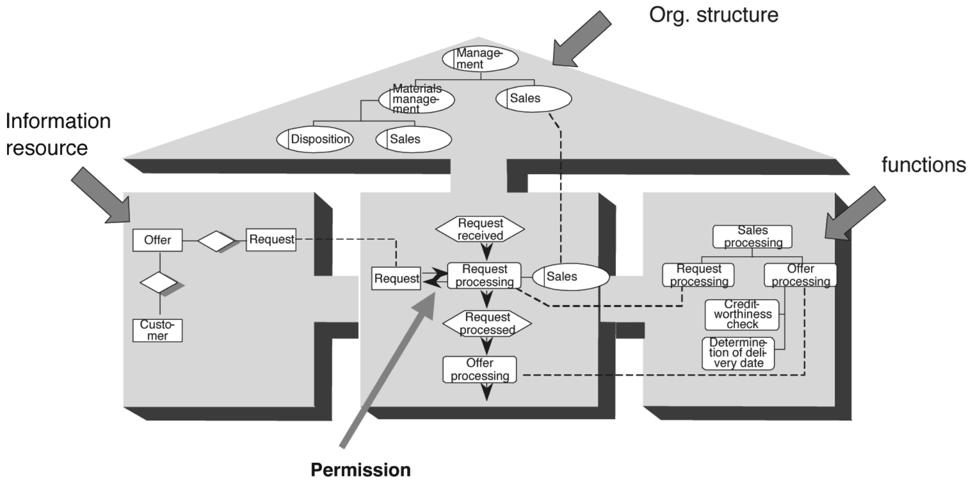


Fig. 9. An overall view of methodology using organizational chart.

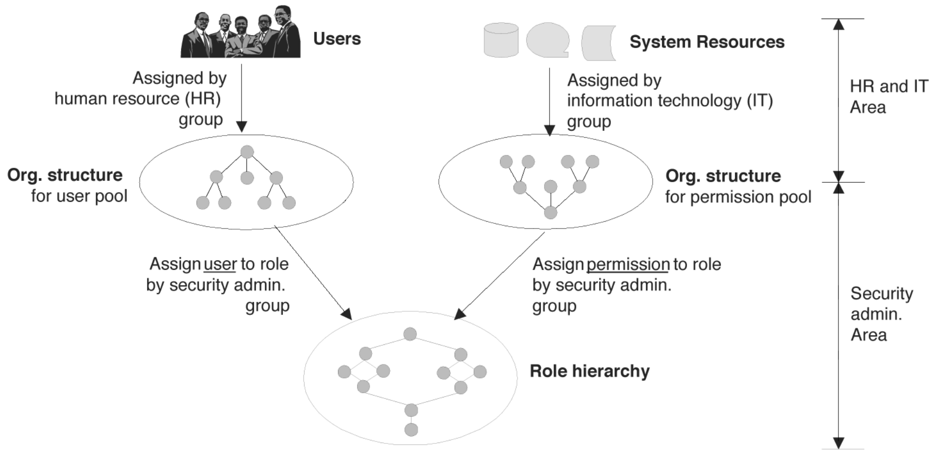

Users
System Resources
Assigned by human resource (HR) group
Assigned by information technology (IT) group
HR and IT Area
Org. structure for user pool
Org. structure for permission pool
Assign user to role by security admin. group
Assign permission to role by security admin. group
Security admin. Area
Role hierarchy

Fig. 10. Role administration concept in ARBAC02.

mission. These properties of an organization are reflected in RBAC model as roles and related role hierarchies.

We now introduce the feature of organization structure as a basis for user/permission pools. Figure 10 shows the role administration concept in the ARBAC02 model. First, users and permissions are assigned to proper organization units by the human resources (HR) group and information technology (IT) group, respectively. The security administration group then assigns the users and permissions in organization units to regular roles. We do not elaborate the functions of the HR and IT groups here, since they are outside the scope of our role-based security administration. We assume the activities of these groups are somehow accomplished in an organization. For the purpose of role administration, we can use different organization structures for user pools and permission pools. Further, we assume that there is no conflict between these

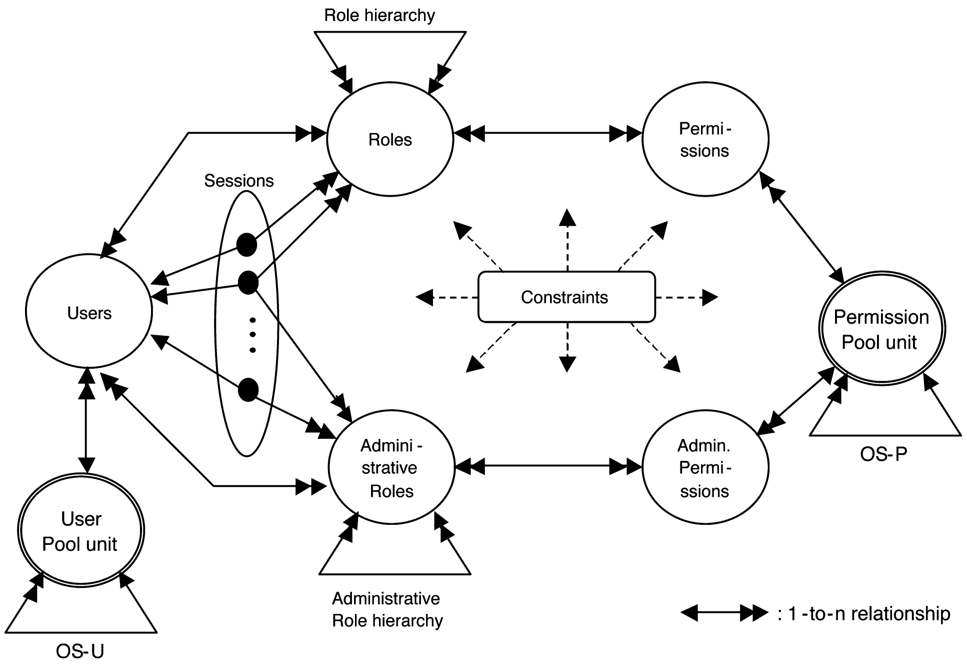


Fig. 11. Components of ARBAC02.

organization structures as they are only used for the purpose of user pool and permission pool administration. Thus, the main difference between ARBAC97 and ARBAC02 model is that ARBAC02 uses additional organizational structures as user/permission pools rather than ARBAC97 model's regular roles as user/permission pools.

4. THE ARBAC02 MODEL

4.1 Description of ARBAC02 Model

In this section, we describe the central notions of ARBAC02 model: to adopt new user and permission pools independent of roles and role hierarchies and a bottom-up method of permission-role administration. Figure 11 shows the components of ARBAC02 model, which is based on ARBAC97 model with two new components, named user pool (*OS-U*) and permission pool (*OS-P*). Both *OS-U* and *OS-P* have hierarchical structures. Unlike a role hierarchy, which can be an arbitrary partial order, the hierarchy is a rooted tree in *OS-U* and an inverted rooted tree in *OS-P*. We assume that simple and basic versions of *OS-U* and *OS-P* are given and users and permissions are preassigned to the proper positions of a given organization structure. There can exist various policies to maintain *OS-U* and *OS-P*, by collaborative efforts between the security administration group, HR group, and IT group in an organization.

4.1.1 *URA02 Model.* URA02 adopts the same notations of *can-assign* and *can-revoke* as URA97. The difference between URA97 and URA02 is that prerequisite roles in URA97 are replaced by a user organization structure (*OS-U*).

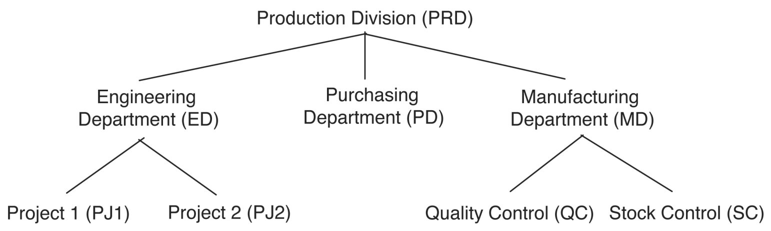

Production Division (PRD)
Engineering Department (ED)
Purchasing Department (PD)
Manufacturing Department (MD)
Project 1 (PJ1)
Project 2 (PJ2)
Quality Control (QC)
Stock Control (SC)

Fig. 12. An example of $OS\text{-}U$.

Definition 4.1. ($OS\text{-}U$) A *user organization structure* is an organization unit (OT) hierarchy represented as a user pool: $OS\text{-}U \subseteq OT \times OT$, and

- has a partially ordered tree structure;
- has a maximal organization unit, and $\forall ot \in OT$, ot has only one direct parent;
- $UUA \subseteq U \times OT$ is a set of user-organization unit assignments;
- $members : OT \to 2^U$ is a function mapping an organization unit to a set of users, and $members(ot) = \{u : U \mid (u, ot) \in UUA\}$;
- $members^* : OT \to 2^U$ is a function mapping an organization unit to a set of users with inheritance hierarchy in $OS\text{-}U$, and $members^*(ot) = \{u : U \mid (\exists ot' \leq ot), (u, ot') \in UUA\}$.

An $OS\text{-}U$ contains the users who are pre-assigned by the HR group in an organization. Figure 12 shows an example of $OS\text{-}U$. If Tom is a member of $PJ1$, it may mean that he has a job position in project 1. If John is a member of ED, it may mean that he is the director of the engineering department. As $OS\text{-}U$ has a tree structure and the characteristic of inheritance, Tom is also a member of ED and PRD.

Definition 4.2. A *prerequisite condition of URA02* is a boolean-valued expression using usual $\wedge$ and $\vee$ operators on terms of form x and $\overline{x}$, where x is a regular role or an organization unit. A prerequisite condition is evaluated for a user u by interpreting x to be true if

- $x \in R$ and $\exists x' \geq x, (u, x') \in URA$, or
- $x \in OT$ and $\exists x' \leq x, (u, x') \in UUA$;

and $\overline{x}$ to be true if

- $x \in R$ and $\forall x' \leq x, (u, x') \notin URA$, or
- $x \in OT$ and $\forall' \geq x, (u, x') \notin UUA$.

To distinguish role and organization unit names, we use an "@" in the head of an organization unit name. With the extra component, we can use mixture of organization structure and roles in prerequisite conditions, which brings enhanced expression power of condition specification. With this, a security administrator can define policies with more precision and flexibility. Following the enhanced prerequisite conditions, an example of *can-assign* in URA97

$can\text{-}assign(PSO1, E1 \wedge \overline{QE1}, [PE1, PE1])$

Table VII. Refined *can-assign* Examples

Admin. Role	Prereq. Condition	Role Range
$PSO1$	$@PJ1 \wedge \overline{QE1}$	$[PE1, PE1]$
$PSO1$	$@PJ1 \wedge \overline{PE1}$	$[QE1, QE1]$
$PSO2$	$@PJ2 \wedge \overline{QE2}$	$[PE2, PE2]$
$PSO2$	$@PJ2 \wedge \overline{PE2}$	$[QE2, QE2]$
DSO	$@ED \wedge \overline{PL2}$	$[PL1, PL1]$
DSO	$@ED \wedge \overline{PL1}$	$[PL2, PL2]$
DSO	$@ED$	(ED, DIR)
SSO	$@ED$	$[ED, ED]$

can be defined in URA02 as

$$can\text{-}assign(PSO1, @PJ1 \wedge \overline{QE1}, [PE1, PE1])$$

Table VII shows the refined *can-assign* information using the *OS-U* shown in Figure 12, according to Table I.

The integrity of administration information such as *can-assign* and *can-revoke* in URA02 is maintained by some rules, which may depend upon specific situations or organizations. In general, we have the following integrity rule in a URA02 model.

Definition 4.3. (*Integrity Rule* 1) Let $AR1$ and $AR2$ be administrative roles. If $AR1$ is senior to $AR2$ in the administrative role hierarchy, then

- (the user pool of $AR1$) $\supseteq$ (the user pool of $AR2$)
- (the role range of $AR1$) $\supseteq$ (the role range of $AR2$)

This rule is trivial, since if $AR1$ is senior to $AR2$, $AR2$'s authority range belongs to $AR1$. For all *can-assign* and *can-revoke* predicates defined in an organization, this rule must be preserved by the system administration.

4.1.2 *PRA02 Model.* PRA02 follows the same notations of *can-assignp* and *can-revokep* as PRA97. Further, PRA02 uses permission pools, where the prerequisite roles are replaced by a permission organization structure.

Definition 4.4. (*OS-P*) A *permission organization structure* is an organization unit hierarchy represented as a permission pool: $OS\text{-}P \subseteq OT \times OT$, and

- has a partially ordered inverted tree structure;
- has a maximal organization unit, and $\forall ot \in OT$, ot has only one direct child;
- $PPA \subseteq P \times OT$ is a set of permission-organization unit assignments;
- $permissions : OT \rightarrow 2^P$ is a function mapping an organization unit to a set of permissions, and $permissions(ot) = \{p : P \mid (p, ot) \in PPA\}$;
- $permission^*(ot) : OT \rightarrow 2^P$ is a function mapping an organization unit to a set of permissions with inheritance hierarchy in *OS-P* hierarchy, and $permissions^*(ot) = \{p : P \mid (\exists ot' \leq ot), (p, ot') \in PPA\}$.

An *OS-P* contains permissions that are preassigned by the IT group in an organization with an inverted tree structure, as shown in Figure 13. In *OS-P*,

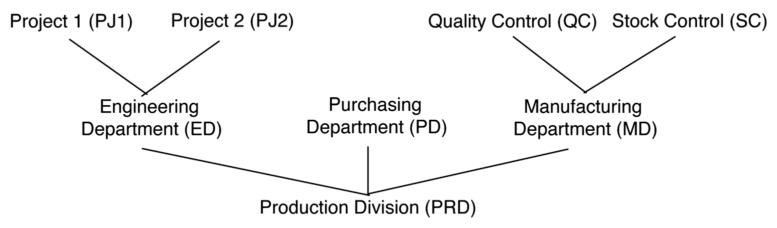

Project 1 (PJ1)
Project 2 (PJ2)
Quality Control (QC)
Stock Control (SC)
Engineering Department (ED)
Purchasing Department (PD)
Manufacturing Department (MD)
Production Division (PRD)

Fig. 13. An example of *OS-P*.

Table VIII. Refined *can-assignp* Examples

Admin. Role	Prereq. Condition	Role Range
SSO	$@ED$	$[E, DIR]$
DSO	$@ED$	$[ED, DIR)$
$PSO1$	$@PJ1$	$[E1, PL1)$
$PSO2$	$@PJ2$	$[E2, PL2)$
$PSO1$	$@PJ1 \wedge \overline{QE1}$	$[PE1, PE1]$
$PSO1$	$@PJ1 \wedge \overline{PE1}$	$[QE1, QE1]$
$PSO2$	$@PJ2 \wedge \overline{QE2}$	$[PE2, PE2]$
$PSO2$	$@PJ2 \wedge \overline{PE2}$	$[QE2, QE2]$

common permissions are assigned to lower units and special permissions are assigned to higher units. For example, access permissions for all the members of the production division are assigned to *PRD* and special permissions for the members of project 1 are assigned to *PJ*1.

From Figure 13, we might expect that users belonging to *PJ*1 inherit the permissions of *ED* and *PRD*. However, it is important that the permission inheritance is downward in *OS-P*. For example, the set of permissions of *ED* is {permissions assigned to *ED*} ∪ {permissions assigned to *PJ*1} ∪ {permissions assigned to *PJ*2}.

Definition 4.5. A *prerequisite condition of PRA02* is a boolean-valued expression using usual $\wedge$ and $\vee$ operators on terms of form x and $\overline{x}$, where x is a regular role or organization unit. A prerequisite condition is evaluated for a permission p by interpreting x to be true if

- $x \in R$ and $\exists x' \leq x, (p, x') \in PRA$, or
- $x \in OT$ and $\exists x' \geq x, (p, x') \in PPA$;

and $\overline{x}$ to be true if

- $x \in R$ and $\forall x' \geq x, (p, x') \notin PRA$ or
- $x \in OT$ and $\forall x' \leq x, (p, x') \notin PPA$.

Table VIII shows the refined *can-assign* predicates in PRA02 using the *OS-P* shown in Figure 13 according to Table III. One of the weaknesses of PRA97 is the top-down approach for permission-role administration. PRAC02 adopts a bottom-up approach. Specifically, common permissions are assigned to lower roles in the role hierarchy and higher roles inherit common permissions; while special permissions are assigned to higher roles. For example, common permissions for all users are assigned to role E, common permissions for the

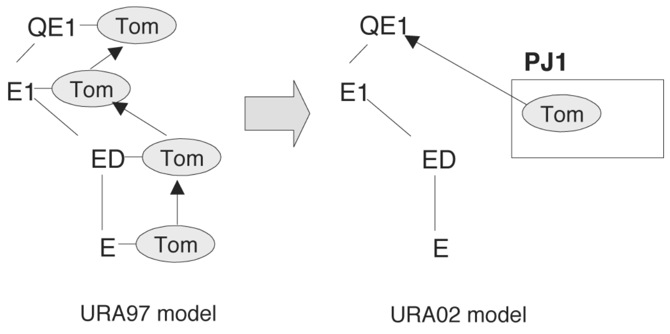


Fig. 14. Comparison of user-role assignment in URA97 and URA02.

engineering department members are assigned to ED, and common permissions for project 1 members are assigned to $E1$. The remaining special permissions are assigned to appropriate higher roles of $E1$. One advantage of this approach is that we avoid duplicate assignments of the same permission through the inheritance line of a role hierarchy. For example, a permission that is assigned to E is not required by ED, $E1$, and so on, therefore redundancy is eliminated.

Like URA02 model, PRA02 needs an integrity rule for defining *can-assignp* and *can-revokep*.

Definition 4.6. (*Integrity Rule 2*) Let $AR1$ and $AR2$ be administrative roles. If $AR1$ is senior to $AR2$ in the administrative role hierarchy, then

- (the permission pool of $AR1$) $\supseteq$ (the permission pool of $AR2$), and
- (role range of $AR1$) $\supseteq$ (role range of $AR2$)

4.2 Advantages of the ARBAC02 Model

4.2.1 *The Effects Role and Role Hierarchy Independent User/Permission Pools.* As described above, a user pool in ARBAC97 is implemented by a prerequisite role, which, in turn, depends on its prerequisite role. As a result, ARBAC97 induces multistep user assignments (URA1) and redundant user-role assignment information (URA2). Furthermore, the composition of a user pool is strongly restricted by a role hierarchy (URA3). In ARBAC02, a user pool is implemented by an organization unit, which is independent from any role or role hierarchy. A new user can be admitted into proper user pool in one step and then be assigned to a proper role from the user pool in one step. Note that assigning a user to a user pool is separated from assigning a user to a regular role in ARBAC02. As a result, a user-role assignment becomes simple and no redundant user-role assignment information exists, i.e., URA1 and URA2 are resolved. Figure 14 shows the comparison of the user-role administration in URA97 and URA02 with our aforementioned example.

Consider the example in URA3 of Section 2.2.1. A company maintains human resource pools H1, H2, and H3. A new policy requires that a production engineer should be selected from H1. In ARBAC02, new organization units H1, H2, and H3 can be added at proper positions in the organization structure. To enable the user-role assignment, predicate *can-assign*$(PSO1, @H1, [PE1, PE1])$ is defined

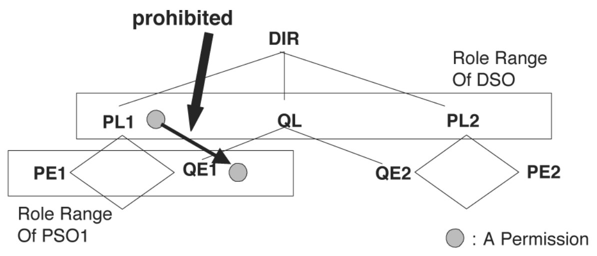


Fig. 15. Prohibition of downward permission flow.

in the URA02 model of the system. This requires no change of the role hierarchy, since the user pool is independent of the role hierarchy. Therefore, URA3 is solved. Similarly, PRA1, PRA2, and PRA3 can be solved in PRA02.

4.2.2 *The Effects of Bottom-Up Permission-Role Administrations.* In ARBAC02, common permissions are assigned to lower positions, while non-common permissions are assigned higher positions in an *OS-P*. As a result, common permissions are inherited by senior roles through the role hierarchy in the model, and permissions do not propagate downward in the role hierarchy (as that happens in ARBAC97). As a result, $PL1$ is not a prerequisite role for $PSO1$ in Figure 15, and a member of $PSO1$ cannot assign $PL1$'s permissions to his/her role range. Therefore, the undesirable side effect of PRA97 (PRA5) does not occur in PRA02. PRA4 is solved naturally, since we do not adopt top-down permission-role administration and a prerequisite role is not restricted as a permission pool.

As a summary, ARBAC02 overcomes the identified shortcomings of URA97 and PRA97. It supports flexible composition of user and permission pools. As a cost, additional components, namely organization structures, should be maintained in an organization, although they are out of the scope of RBAC administration. As organization structure is a natural notion for most organizations, we expect that this is not an extensive overhead.

5. *OS-U/OS-P* WITH ROLE-ROLE ASSIGNMENT

In this section we show how to improve role-role assignments (RRA) with *OSU/OS-P*. We claim that *OS-U/OS-P* is a useful approach to develop improved RRA model.

The purpose of an ARBAC model is to support safe and decentralized role administration. A basic requirement is to allow legal authorizations and prevent illegal authorizations for an administrative role. In particular, for user-role assignments, a member of an administrative role only can take users from the role's user pool and assign to roles in the corresponding role range. Figure 16 shows an example. For administrator $A1$, only $UA1$ and $PA1$ are legal. $UA2$ is illegal for $A1$ because he/she cannot assign users out of the role range. $UA3$ is also illegal for $A1$ because he/she cannot pick users outside of $A1$'s user pool. URA02 and PRA02 models prevent these illegal authorizations by *can-assign* and *can-assignp* relations, respectively. However, for RRA, as we have mentioned in Section 2.1.3, there may exist indirect illegal authorizations. To make one role

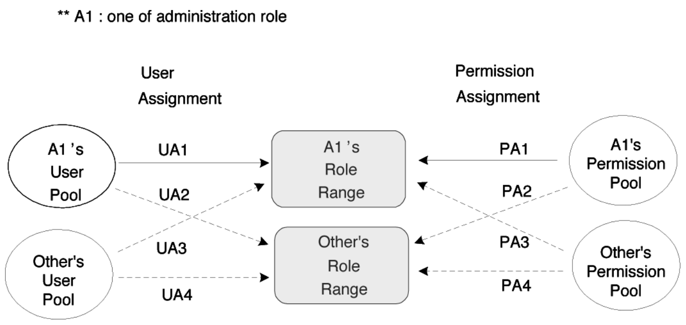


Fig. 16. Example of legal and illegal authorizations.

senior to another, illegal assignment may take place as shown in Figure 6. RRA97 model [Sandhu and Munawer 1998] and administrative scope model [Crampton and Loizou 2002] choose a topological approach to solve the problem.

We now show that *OS-U*/*OS-P* can used to improve RRA. Consider an RBAC model shown in Figure 17. Suppose *PSO*1 introduces an edge ($PL1 \rightarrow Y$) in the role hierarchy (17a). It is a legal action because both *PL*1 and Y belong to *PSO*1's role range of *can-modify* (Figure 17g enhanced to include [E1, PL1] in range of PSO1). As a result, all the permissions of Y are inherited by *DIR* through *PL*1, which is a legal flow of permissions. This case is represented by *OS-U*/*OS-P* in Figure 18, where adding ($PL1 \rightarrow Y$) makes an inheritance flow from $U4 \rightarrow U5 \rightarrow U9$. This means that the authority flows from the user-pool of *PSO*1 to the user-pool of *DSO* or from the permission-pool of *PSO*1 to the permission-pool of *DSO*. It is a legal flow because *DSO* is senior to *PSO*1, and the user and permission pools of *DSO* include the user and the permission pool of *PSO*1, respectively.

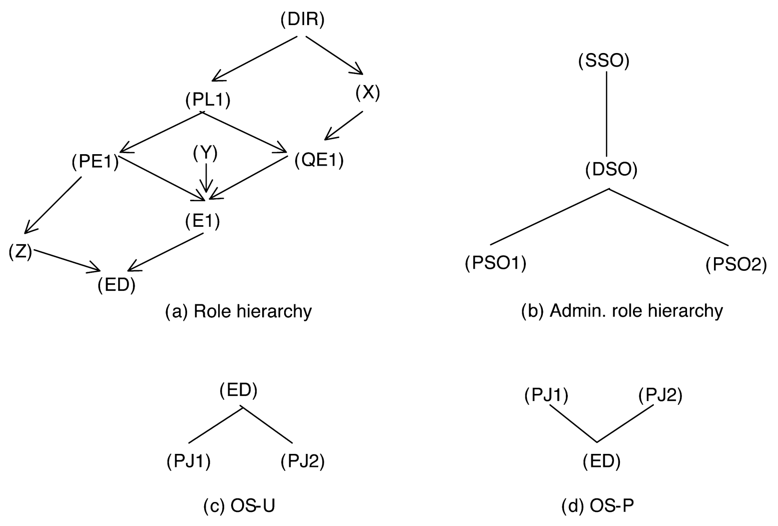
Let's consider another case. Suppose *PSO*1 introduces an edge ($QE1 \rightarrow PE1$) in the role hierarchy. This is a illegal action because the authority of Z is inherited by X through *PE*1 and *QE*1, while role Z and X do not belong to *PSO*1's role range of *can-modify* (Figure 17g). Figure19 shows the permission flow with this insertion. As it shows, the authority of $U8$ is inherited by $U7$ through $U1$ and $U2$. As we can see, $U8$ and $U7$ do not belong to the user pool of *PSO*1. Thus, the action is illegal.

With the above analysis, to prevent illegal edge insertions in a role hierarchy, some rules must be preserved in RRA with *OS-U*/*OS-P*.

Definition 5.1. (*Integrity Rule* 3) Let *UP*1 and *UP*2 be user pools, *PP*1 and *PP*2 be permission pools, and $UP1 \supset UP2$ and $PP1 \supset PP2$. If security administrator *AR*2, who has *UP*2 and *PP*2, adds a new edge in a role hierarchy, then

- $[UP2 \rightarrow (UP2 \text{ or } UP1)] \vee [PP2 \rightarrow (PP2 \text{ or } PP1)] \Rightarrow$ legal action
- $[UP1 \rightarrow UP2 \rightarrow UP1] \vee [PP1 \rightarrow PP2 \rightarrow PP1] \Rightarrow$ illegal action

where *rightarrow* indicates a permission flow because of inheritance relation. This rule indicates that if an insertion causes authority flow of $UP2 \rightarrow (UP2$ or $UP1)$ or $PP2 \rightarrow (PP2$ or $PP1)$, then it is legal insertion. While a flow of $UP1 \rightarrow UP2 \rightarrow UP1$ or $PP1 \rightarrow PP2 \rightarrow PP1$ implies illegal insertion.



(e) Can-assign

Admin. Role	User pool	Role range
PSO1	@PJ1	[E1, PL1]
DSO	@ED	[ED, DIR]

(f) Can-assignp

Admin. Role	permission pool	Role range
PSO1	@PJ1	[E1, PL1]
DSO	@ED	[ED, DIR]

(g) Can-modify

Admin. Role	Role range
PSO1	[E1, PL1]
DSO	[ED, DIR]

(h) User-OS-U assignment

Org. unit	User
PJ1	U1, U2, U3, U4,U5
PJ2	U6
ED	U7, U8, U9

(i) Permission-OS-P assignment

Org. unit	Permission
PJ1	P1, P2, P3, P4,P5
PJ2	P6, P7
ED	P8, P9, P10

(j) User-role assignment

Role	User
DIR	U9
PL1	U5
PE1	U1
QE1	U2
X	U7
Y	U4
Z	U8

(k) Permission-role assignment

Role	Permission
DIR	P10
PL1	P5
PE1	P1, P2
QE1	P2, P3
E1	P4
X	P7,P9
Y	P4
Z	P8

Fig. 17. An example of RRA with *OS-U*/*OS-P*.

Definition 5.2. (*Integrity Rule 4*) Let $UP1$ and $UP2$ be user pools, $PP1$ and $PP2$ be permission pools, and $UP1 \supset UP2$ and $PP1 \supset PP2$. If security administrator $AR1$ who has $UP1$ and $PP1$ adds a new edge in a role hierarchy, then

- $[UP1 \rightarrow UP2] \vee [PP1 \rightarrow PP2] \Rightarrow$ legal action but implies potential risk

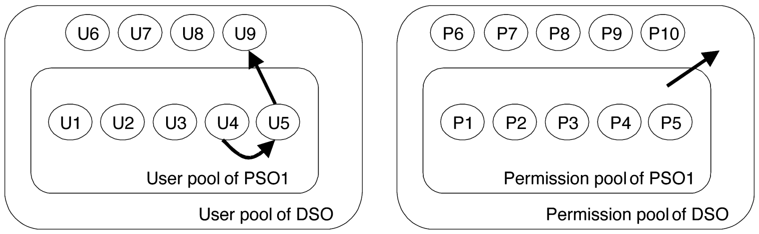


Fig. 18. Interpretation of adding $(PL1 \rightarrow Y)$ by *OS-U*/*OS-P*.

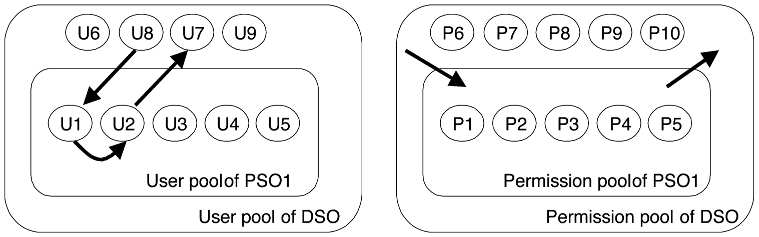


Fig. 19. Interpretation of adding $(QE1 \rightarrow PE1)$ by *OS-U*/*OS-P*.

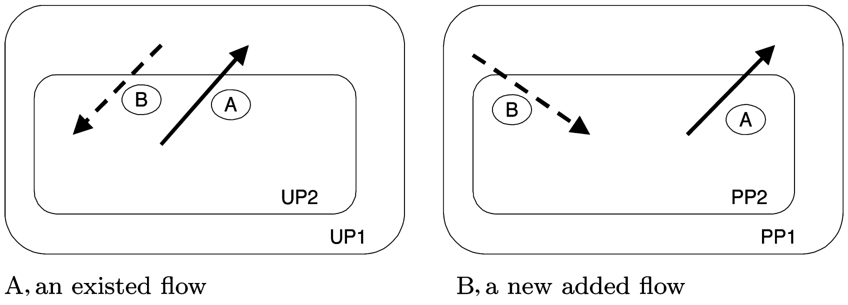


Fig. 20. An example of potential risk.

Figure 20 shows a situation of Integrity Rule 4. Suppose there is an insertion causing the authority flow A shown in the figure. Conceptually this is a legal flow. However, if there is another insertion causing flow B and B and A are linkable by these two insertions, then an illegal flow is generated, as like that in Figure 19. With this integrity rule we can find that the role hierarchy in Figure 17a is legal, but has potential risk to introduce illegal inheritance of authority.

6. APPLYING *OS-U/OS-P* TO OTHER AREAS

ARBAC02 is suitable for any areas requiring RBAC model. However, the concept of organization structured-based user pools and permission pools is not limited to RBAC systems. As shown in Figure 21, *OS-U* and *OS-P* can be supporting components in an organization and used with other access control models. In this section, we describe the applications with access control list model and lattice-based access control model.

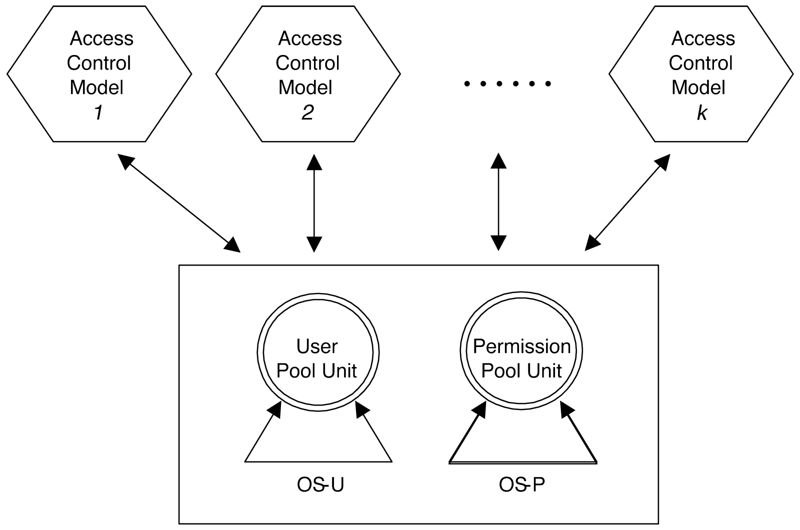


Fig. 21. Applying user/permission pool to other models.

6.1 *OS-U/OS-P* with Access Control List

As Figure 22 shows, an object's access control list (ACL) indicates the subjects with respective rights to it. Logically an ACL model can be represented as a list of triples with the form $< subject, object, rights >$.

Suppose a large organization, which adopts ACL for access control, has thousands of subjects (users) and objects. In this case, it is impossible for a single security officer to manage the ACLs, which generally needs cooperative efforts of many security officers. A problem for this purpose does arise on how to assign a security officer with proper management area of the ACLs. *OS-U/OS-P* provides a good solution for this environment.

To build a framework of ACL with *OS-U/OS-P*, we assume *OS-U/OS-P* is prebuilt in an organization. At first, we need to define the authority boundary of each security officer. We adopt an administration table (Table IX), which is based on Figures 12 and 13. For example, the first row indicates that security officer $SO1$ can assign permissions in $@PJ1$ (in permission pool) to users belonging to $@PJ1$ (in user pool). With this approach, we can control the authority of each security officer very efficiently.

6.2 *OS-U/OS-P* with Lattice-Based Access Control

In lattice-based access control (LBAC) [Bell-Lapadula 1975; Sandhu 1993], an access control decisions are made beyond the control of an individual (e.g., the owner of an object). A central authority determines what information can be accessed by whom and individual users cannot change the access rights of objects. In LBAC, each subject (user) or information object is labeled with a security level and a subject's access right is restricted according to its security level. LBAC model can be used in multilevel security systems for no-read-up and no-write-down properties, also known as Bell–Lapadula restrictions. As an

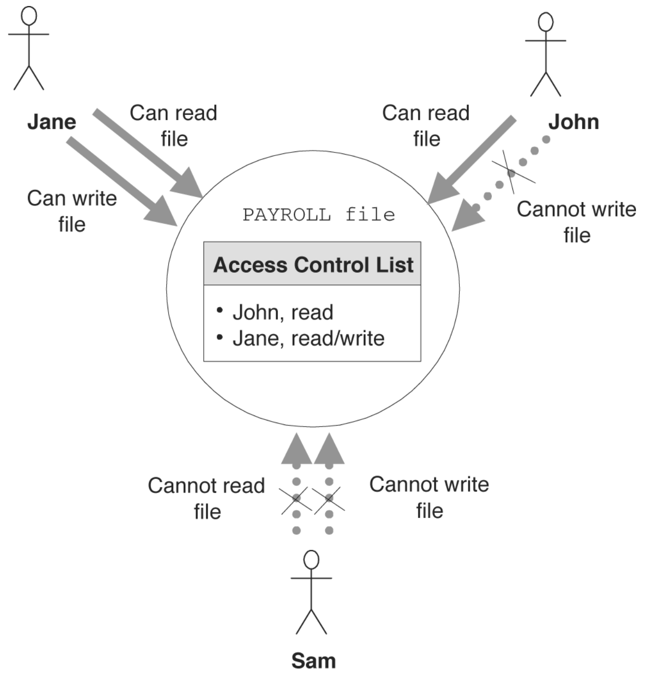


Fig. 22. Example of access control list.

Table IX. Administration for ACL

Security Officer	User Pool (subject pool)	Permission Pool (object pool)
$SO1$	$@PJ1$	$@PJ1$
$SO2$	$@PJ2$	$@PJ2$
$SO3$	$@ED$	$@ED \vee @PD$
$SO4$	$@PRD$	$@PRD$
$SO5$	$@MD$	$@QC \vee @SC$

example, shown in Figure 23, these rules are designed to ensure that information does not flow from a higher sensitivity level to a lower sensitivity level. For information integrity purposes, other access rules can be formulated, such as no read-down and no write-up [Biba 1977].

For a large organization using LBAC, the security administration is also a problem, since the number of subjects and objects is large, and the administration requires cooperation between many security officers. In LBAC, a security officer assigns security levels to subjects or objects. It is reasonable that a security officer itself has a security level. Again, organization structure can be used in LBAC administration, as Table X shows. This table has the same structure as Table IX. But here a security officer may only have user pool or permission pool. For example, a member of $SO1$ has the authority of assigning security levels to users in $@PJ1$, but has no authority of assigning security levels to objects.

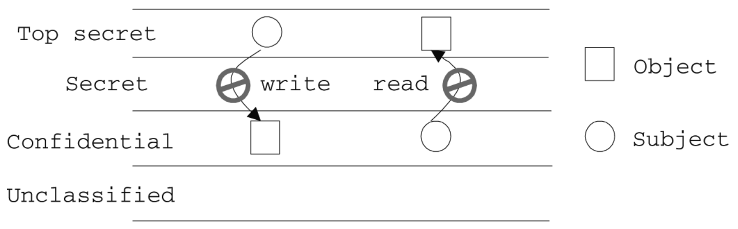


Fig. 23. Example of LBAC.

Table X. Administration for LBAC

Security Officer	User Pool (subject pool)	Permission Pool (object pool)
*SO*1	@*PJ*1	
*SO*2	@*PJ*2	@*PJ*1
*SO*3	@*ED*	
*SO*4	@*PRD*	@*PRD*
*SO*5		@*PJ*2

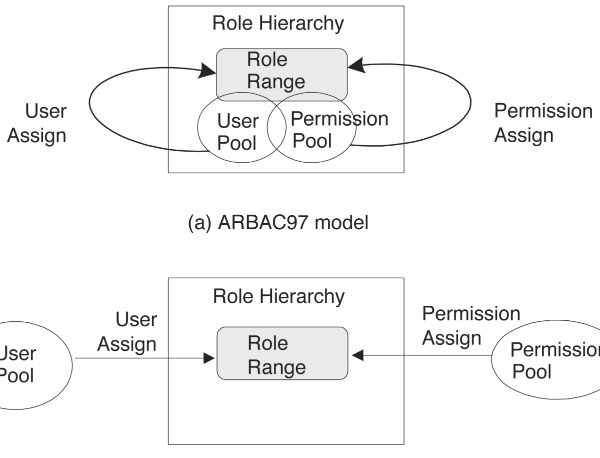
7. RELATED WORK

Moffett [1998] and [Moffett and Lupu 1999] discuss the meaning of organization and role hierarchy. Perwaiz and Sommerville [2001] show how to manage permission-role relationship using organization units. Oh and Park [2001] propose a method to derive access control information from an enterprise model. All these work present similar concepts and notation of organization units and structures, but not in the context and purpose of role administration for RBAC, as this paper does.

Osborn and Guo [2000a] introduce groups and group graph in the administration of RBAC. Users belong to some groups and roles can be assigned to groups. The key difference between *OS-U* and user group graph is that there is no role assigned to users in *OS-U*, while group has role assignments. This is because *OS-U* is generated and managed by departments (for example, the human resource department in an organization) other than security administrators. For group graph, it is managed by security administrators. Therefore, we can consider *OS-U* as an external user-pool for security administrators to pick users for user-role assignments.

8. CONCLUSIONS

In this paper, we described ARBAC02, an improved administrative RBAC model. Our motivation is based on shortcomings of ARBAC97 caused by unnecessary coupling between user/permission pools with roles and role hierarchies. To overcome the shortcomings, we introduce organization structure-based user and permission pools independent from the roles and role hierarchy in an organization. Figure 24 shows the main difference between ARBC97 and ARBAC02. In addition, we use a bottom-up inheritance for permission-role administration, instead of the top-down manner in ARBAC97. Independent user and permission pools give strong flexibility for URA and PRA administrations and overcome the identified weaknesses in RRA. At the same time, we illustrate the applications



(b) ARBAC02 model

Fig. 24. Main difference between ARBAC97 and ARBAC02.

of *OS-U*/*OS-P* in other access control models, such as ACL and LBAC. This shows that *OS-U*/*OS-P* is a comprehensive solution of security administration for different access control models.

REFERENCES

BIBA, K. J. 1977. Integrity Considerations for Secure Computer Systems. Mitre Corp. Report No.TR3153, Bedford, MA. (Also available through Nat'l Technical Information Service, Springfield, Va., Report No. NTIS AD–A039324.)

BELL, D. E. AND LAPADULA, L.J. 1975. Secure Computer Systems: Mathematical Foundations and Model. Mitre Corp. Report No. M74-244, Bedford, MA. (Also available through Nat'l Technical Information Service, Springfield, VA, Report No. NTIS AD-771543.)

CRAMTON, J. AND LOIZOU, G. 2002. Administrative scope and role hierarchy operations. In *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT2002)*. Monterey, CA.

IDS SHARE. Aris house. *http://www.ids-scheer.com*

JOSHI, J. B. D., AREF, W. G., GHAFOOR, A., AND SPAFFORD, E. H. 2001. Security models for web-based applications. *Communications of the ACM, 44*, 2.

MOFFETT, J. D. 1998. Control principles and role hierarchies. In *Proceedings of the 3rd ACM Workshop on Role-Based Access Control*. Fairfax, VA.

MOFFETT, J. D. AND LUPU, E. C. 1999. The use of role hierarchies in access control. In *Proceedings of the 4th ACM Workshop on Role-Based Access Control*. Fairfax, VA.

NYANCHAMA, M. AND OSBORN, S. 1999. The role graph model and conflict of interest. *ACM Transactions on Information and System Security, 2*, 1, 3–33.

OH, S. AND PARK, S. 2001. An improved administration method on role-based access control in the enterprise environment. *Journal of Information Science and Engineering 17*, 921–944.

OSBORN, S. AND GUO, Y. 2000. Modeling users in role-based access control. In *Proceedings of Fifth ACM Workshop on Role-Based Access Control*, 2000.

OSBORN, S., SANDHU, R., AND MUNAWER, Q. 2000. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and System Security, 3*, 2, 85–106.

PERWAIZ, N. AND SOMMERVILLE, I. 2001. Structured management of role-permission relationships. In *Proceedings of 6th ACM Symposium on Access Control Models and Technologies*. Chantilly, VA.

SANDHU, R. 1993. Lattice-Based Access Control Models. *IEEE Computer, 26*, 11.

Sandhu, R. and Bhamidipati, V. 1997a. The URA97 model for role-based user-role assignment. In *Proceedings of IFIP WG 11.3 Workshop on Database Security*. Lake Tahoe, CA.

Sandhu, R. and Bhamidipati, V. 1997b. The ARBAC97 model for role-based administration of Roles: Preliminary description and outline. In *Proceedings of second ACM Workshop on Role-Based Access Control*. Fairfax, VA.

Sandhu, R. and Munawer, Q. 1998. The RRA97 model for role-based administration of role hierarchy. In *Proceedings of the Annual Computer Security Applications Conference*. Phoenix, AZ.

Sandhu, R., Coyne, E., Feinstein H., and Youman, C. 1996. Role-based access control models. *IEEE Computer, 29*, 2, 38–47.

Sandhu, R. and Bhamidipati, V. 1999. Role-based administration of user-role assignment: The URA97 model and its Oracle implementation. *Journal of Computer Security, 7*.

Sandhu, R. and Munawer, Q. 1999. The ARBAC99 model for administration of roles. In *Proceedings of the Annual Computer Security Applications Conference*. Phoenix, AZ.

Sandhu, R., Bhamidipati V., and Munawer, Q. 1999. The ARBAC97 model for role-based administration of roles. *ACM Transactions on Information and System Security, 2*, 1, 105–135.

Received January 2003; revised July 2005; accepted February 2006