# Handbook of Research on Social and Organizational Liabilities in Information Security

Manish Gupta
*State University of New York at Buffalo, USA*

Raj Sharman
*State University of New York at Buffalo, USA*

Information Science REFERENCE

INFORMATION SCIENCE REFERENCE

Hershey · New York

Director of Editorial Content: Kristin Klinger
Director of Production: Jennifer Neidig
Managing Editor: Jamie Snavely
Assistant Managing Editor: Carole Coulson
Typesetter: Jeff Ash
Cover Design: Lisa Tosheff
Printed at: Yurchak Printing Inc.

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue, Suite 200
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

and in the United Kingdom by
Information Science Reference (an imprint of IGI Global)
3 Henrietta Street
Covent Garden
London WC2E 8LU
Tel: 44 20 7240 0856
Fax: 44 20 7379 0609
Web site: http://www.eurospanbookstore.com



Library of Congress Cataloging-in-Publication Data

Handbook of research on social and organizational liabilities in information security / Manish Gupta and Raj Sharman, editors.

p. cm.

Includes bibliographical references and index.

Summary: "This book offers insightful articles on the most salient contemporary issues of managing social and human aspects of information security"--Provided by publisher.

ISBN 978-1-60566-132-2 (hardcover) -- ISBN 978-1-60566-133-9 (ebook)

1. Computer security--Management--Handbooks, manuals, etc. 2. Data protection--Management--Handbooks, manuals, etc. 3. Computer crimes--Prevention--Handbooks, manuals, etc. 4. Human computer interaction--Handbooks, manuals, etc. I. Gupta, Manish, 1978- II. Sharman, Raj.

QA76.9.A25.H365 2008

658.4'78--dc22

2008035140

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

# Chapter VI
# Towards a Scalable Role and Organization Based Access Control Model with Decentralized Security Administration

**Zhixiong Zhang**
*The College Board, USA*

**Xinwen Zhang**
*Samsung Information System America, USA*

**Ravi Sandhu**
*University of Texas at San Antonio, USA*
*TriCipher Inc., USA*

## ABSTRACT

*This chapter addresses the problem that traditional role-base access control (RBAC) models do not scale up well for modeling security policies spanning multiple organizations. After reviewing recently proposed Role and Organization Based Access Control (ROBAC) models, an administrative ROBAC model called AROBAC07 is presented and formalized in this chapter. Two examples are used to motivate and demonstrate the usefulness of ROBAC. Comparison between AROBAC07 and other administrative RBAC models are given. We show that ROBAC/AROBAC07 can significantly reduce administration complexity for applications involving a large number of organizational units. Finally, an application compartment-based delegation model is introduced, which provides a method to construct administrative role hierarchy in AROBAC07. We show that the AROBAC07 model provides convenient ways to decentralize administrative tasks for ROBAC systems and scales up well for role-based systems involving a large number of organizational units.*

## INTRODUCTION

With the wide Internet usage in our society, the security and privacy issues become more important than ever. In the last decade, role-based access control (RBAC) had been generating considerable interests among the researchers and practitioners. In RBAC, roles are defined based on job functions, permissions are associated with roles, and users are made members of appropriate roles, thereby acquiring the roles' permissions. This indirect association between users and permissions greatly simplifies users' permission management. RBAC has several attractive features, such as policy neutrality, principle of least privilege, and ease of management. Several well-known RBAC models, such as RBAC96 (Sandhu et al, 1996), the role graph model (Nyanchama & Osborn, 1999), and NIST model (Ferraiolo et al., 2001), have been developed during the last decade. Those models form the basis for the ANSI RBAC standard (ANSI INCITS 359-2004). As a powerful alternative to traditional discretionary and mandatory access control, the adoption of RBAC in commercial software and enterprises has rapidly increased in recent years (RTI International, 2002).

The complexity of an RBAC system can be defined on the basis of the number of roles, the number of permissions, the size of the role hierarchy, the constraints on user-role and permission-role assignments, etc. (Sandhu et al, 2000). For existing large-scale RBAC systems, the number of roles and the number of permissions are in the order of 1000s. Beyond that magnitude, the performance of RBAC may degrade and its management becomes too difficult to handle correctly. Several approaches (Giuri & Iglio, 1997; Thomas, 1997; Perwaiz & Sommerville, 2001; Park et al, 2004) have been proposed to scale up RBAC systems by extending the RBAC model from various perspectives. To achieve decentralized administration of RBAC, some role-based administrative models have been proposed (Sandhu et al, 1999; Crampton & Loizou, 2003; Oh et al, 2006; Bhatti et al, 2004). Most of the previous work address RBAC in the context of a single organization and are mainly motivated by B2E (Business to Employee) applications. On the other hand, B2B (Business to Business) and B2C (Business to Consumer) applications often involve a large number of organizations such as corporations, schools, families, etc. Typically, users from different organizations with the same role name have slightly different access privileges due to privacy consideration. For example, a user with parent role in family A has permission to view the progress records of Family A's kids but not the progress recodes of other families' kids. Using standard RBAC naively in these situations can result in an enormous number of roles and permissions, well into the order of millions.

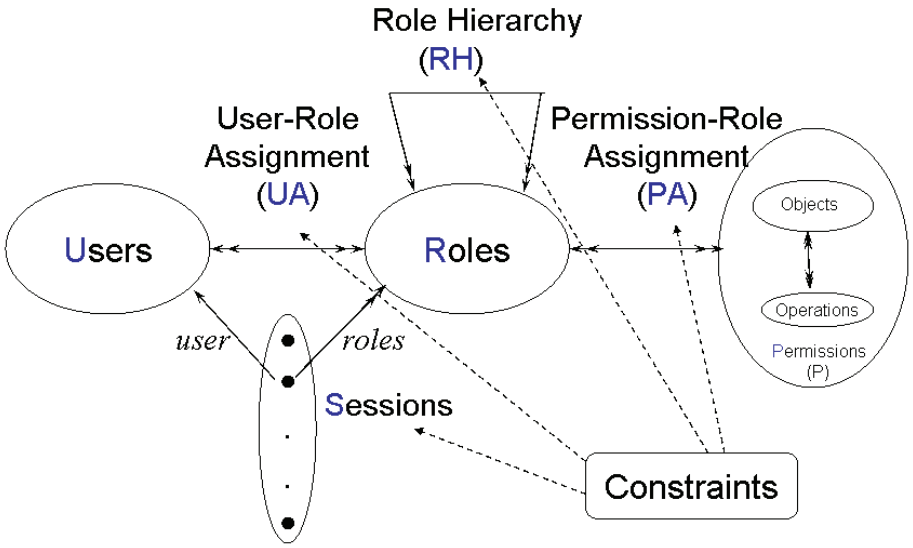
This chapter tries to address the scalability problem when applying RBAC to applications involving many organizational units. The rest of this chapter is organized as follows. Section 2 gives background and two motivating examples. Section 3 reviews Role and Organization Based Access Control (ROBAC) models. Section 4 presents a decentralized administrative ROBAC model called AROBAC07 (administrative ROBAC '07) to control administrative tasks in ROBAC systems. Section 5 discusses the implementation perspective of ROBAC. Section 6 concludes the chapter.

## BACKGROUND

ANSI RBAC reference model includes core RBAC (no role hierarchy), hierarchy RBAC (has role hierarchy), and constrained RBAC (has Separation of Duty constraints). Figure 1 shows a classic (standard) RBAC which is based on the well-known RBAC96 and permission definition from ANSI RBAC.

Here we use the term classic RBAC to refer the typical RBAC models proposed in (Sandhu et al, 1996; Nyanchama & Osborn, 1999; Ferraiolo et al,

*Figure 1. Classic (standard) RBAC*




2001). As you can see, permissions are assigned to roles and users are assigned to roles. Users acquire permissions via their memberships in roles during session. The permissions in standard RBAC are defined as operations over objects. Here objects represent any resources need to be protected in the system. Assigning permissions to roles and assigning roles to users are two separate administrative tasks. How to define roles and permissions depends on desired security policy in an organization. RBAC models have been extended from various aspects (temporal, spatial, or context-aware) to better meet the needs in real world (Bertino et al, 2001; Joshi et al, 2005; Bertino et al, 2005, Covington et al, 2001; Kumar et al, 2002). Due to indirect assignment between users and permissions, RBAC based system is more flexible and scales up better than traditional MAC and DAC based system with respecting to the number of users. But RBAC does not scale up well with respecting to the number of roles and permissions. Beyond the magnitude of thousand roles, the management of RBAC is very error prone. Several approaches have been proposed to scale up RBAC systems. Giuri and Iglio (1997) extend RBAC by introducing the concept of role templates with parameterized privileges to achieve content-based access control. Thomas (1997) proposes Team Based Access Control (TMAC) to scale up permission assignment with fine-grained run-time permission activation at the level of individual users and objects. Perwaiz and Sommerville (2001) describe a mechanism for viewing role-permission relationships in the context of organizational structures, which reduces the number of roles in an RBAC implementation. Park et al. (2004) propose a composite RBAC for large and complex organizations. Most of these existing models address problem in the context of one organization. Many B2B and B2C applications involve a large number of organizations and often have some privacy requirements such as users in one organization are only allowed to access the resources related to the organization and are not allowed to access other organizations' resources. We discuss two examples from B2B and B2C context, respectively, to show the motivation of our new models. These are abstracted from our experience with similar real-world applications.

**B2B example:** Consider access control policies for a web based report delivery system, which only allows authorized users to access specific

reports. Users are educational professionals from schools, districts, and states in USA. There are on the order of 10,000 schools participating in the system. Reports are classified into types based on the sensitivity and nature of the content. Because some report types include privacy-sensitive data such as student test results and personal information, only an authorized user, say, School_1's official, can view School_1's reports but cannot view School_2's reports. There are many different types of reports, each of which may have up to three different levels of information (state level, district level, and school level). Some sample report types are listed in Table 1.

States, districts, and schools usually form a management hierarchy. Figure 2 shows an example of a possible management hierarchy among states, districts, and schools.

Informal security policies of the system may include:

- Users from a school are only allowed to access the reports related to this school.
- Users from a district education office are allowed to access the reports related to this district and the schools under it.
- Users from a state education office are allowed to access the reports related to this state and the districts and schools under it.
- School principles can view type A and type B reports.
- School teachers can view type B and type E reports.
- Officials from a district's board of education offices can view type A and type B reports but cannot view type D reports

Under the above policies, an authorized school level user (say School_1 teacher) can only access certain types of the user's own school's reports, but is not allowed to access other types of reports, and, further, cannot access other school's or any district or state level reports. Here we assume that an access not explicitly allowed by the stated policies is denied. An authorized district level user can access certain types of the user's own district's reports (district level) and may also access the same types of its subordinate schools' reports. For example, an authorized District_1 official can access District_1's district level Type_A reports and school level Type_A report for School_1 and School_2 since the School_1 and School_2 are under District_1.

Assuming there are 10,000 organizations and 10 types of reports, if we use standard RBAC to model this problem directly, we have to define about 100,000 (10,000 x 10) permissions because viewing School_1's Type_A report is different from viewing School_2's Type_A report. We also need to define 100,000 different roles because a role that can view a School_1_Type_A report is different from a role that can view a School_2_Type_A report. Table 2 and Table 3 show some samples of the possible permissions and roles in this example.

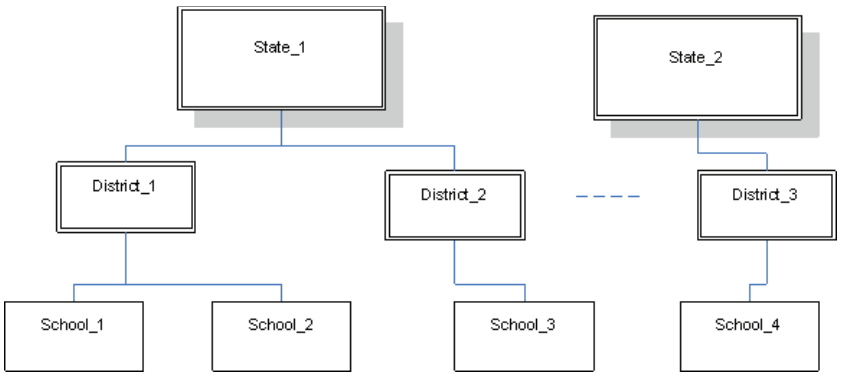
The goal here is not to define a complete and coherent policy for this example but rather to illustrate the issues and complexities involved.

**B2C example:** Consider an online subscription-based tutoring system, where customers are families that have children in elementary schools. Parents pay subscription fees for their children and are authorized to create/update the family's profile and view their children's progress reports. Students that have subscribed to the service can take classes on the web and view their progress

*Table 1. Sample report types in B2B example*

| |
|---|
| Type A Report (school level, district level, and state level) |
| Type B Report (school level only) |
| Type C Report (school level only) |
| Type D Report (school level only) |
| Type E Report (school level and district level) |
| … |

*Figure 2. A sample organization hierarchy*




reports and family profiles. Here, family profiles and student's progress reports need to be protected against unauthorized access. There are potentially millions of families, and even 10s or 100s of millions. The informal description of some security policies of this system may include:

- Parents can only view their own children's progress reports.
- Parents can create/update/view their family's profile.
- A student can view his/her own progress report and view his/her family's profile.

Suppose we use standard RBAC to model these policies. Because Family_1's parent is only allowed to access Family_1's profile and Family_1's children's progress reports, the Family_1's parents have slightly different permissions from that of Family_2's parents. Table 4 and Table 5 show some samples of the possible permissions

*Table 2. Sample permissions in B2B example (with RBAC)*

| |
|---|
| p1: View School_1 Type A Report |
| p2: View School_2 Type A Report |
| p3: View District_1 Type A Report |
| … |

*Table 3. Sample roles in B2B example (with RBAC)*

| |
|---|
| r1: School_1 Type A Report Viewer with permission p1. |
| r2: School_2 Type A Report Viewer with permission p2. |
| r3: District_1 Type A Report Viewer with permission p3. |
| … |

*Table 4. Sample permissions in B2C example (with RBAC)*

| |
|---|
| p1: Update Family_1's Profile |
| p2: View Family_1's Kids' Progress Reports |
| p3: View Family_1's Profile |
| p4: Update Family_2's Profile |
| p5: View Family_2's Kids' Progress Reports |
| p6: View Family_2's Profile |
| …… |

*Table 5. Sample roles in B2C example (with RBAC)*

| |
|---|
| r1: Family_1 Parents with permission p1 and p2. |
| r2: Family_1 Student with permission p2 and p3. |
| r3: Family_2 Parents with permission p4 and p5. |
| r4: Family_2 Student with permission p5 and p6. |
| … |

and roles when using classic RBAC in this B2C example.

We can see that the administrative complexity is very high in applying RBAC directly to the above two examples. These scenarios are quite typical for B2B and B2C applications. In practice, security and application engineers usually work around this problem by combining RBAC with other access control mechanisms such as context-based or attribute-based access control. The result is an ad hoc access control model with a specialized administrative tool for each application (Georgiadis et al, 2001; Schaad et al, 2001).

## ROBAC MODELS

To address the issue that classic RBAC does not scale up well for applications involving multiple organizations where privacy issue is the main concern, a family of extended RBAC models called Role and Organization Based Access Control (ROBAC) models has been proposed by the authors (Zhang et al, 2006).

The central idea behind ROBAC is quite simple. Instead of only using role related information, ROBAC utilizes both the role information and the organization information during the authorization process. Specifically, in ROBAC, a user is assigned to role and organization pairs instead of roles only. The permissions in ROBAC are defined as operations over object types instead of operations over objects. A user can access an object if and only if the user is assigned to a role and organization pair, and the role has the right to access the object's type and the object is related to the organization. In the following sections, we show that the number of roles and permissions for the above B2B and B2C examples can be reduced significantly if we use ROBAC to model them.

ROBAC models consist of four models ($ROBAC_0$, $ROBAC_1$, $ROBAC_2$, $ROBAC_3$) based on the increasing security functionality in direct correspondence with the four models of well-known RBAC96 family ($RBAC_0$, $RBAC_1$, $RBAC_2$, $RBAC_3$). $ROBAC_0$ is a base model. $ROBAC_1$ is $ROBAC_0$ plus role hierarchy and organization hierarchy. $ROBAC_2$ is $ROBAC_0$ plus constraints. $ROBAC_3$ is $ROBAC_0$ plus role hierarchy, organization hierarchy and constraints. Figure 3 shows their essential characteristics.

To make the chapter concise, we only review the formal definitions for $ROBAC_0$ and $ROBAC_1$ here.

**Definition 1:** $ROBAC_0$ has the following components:

- U -- a set of users (same as U in RBAC96);
- S -- a set of sessions (same as S in RBAC96);
- R -- a set of roles (same as R in RBAC96);
- O -- a set of organizations;
- Op -- a set of operations;
- A -- a set of assets;
- At -- a set of asset types;
- $P \subseteq Op \times At$ -- a set of permissions;
- $RO \subseteq R \times O$ -- a set of applicable role and organization associations;
- $PA \subseteq P \times R$ -- a many-to-many permission-to-role assignment relation;
- $UA \subseteq U \times RO$ -- a many-to-many user-to-role-and-organization assignment relation;
- *user*: $S \rightarrow U$ -- a function mapping a session $s_i$ to a single user *user*($s_i$) (same as *user* in RBAC96);
- *atype*: $A \rightarrow At$ -- a function mapping an asset to its type;
- *aorg*: $A \rightarrow O$ -- a function mapping an asset to the organization is related to;
- *assigned_role-orgs*: $U \rightarrow 2^{RO}$ -- a function mapping a user to a set of role-organization pairs assigned to the user; formally: *assigned_role-orgs*(*u*) = { (*r*,*o*) | (*u*, (*r*,*o*)) $\in$ UA };

- *active_role-orgs*: S → $2^{RO}$ -- a function mapping a session $s_i$ to a set of active role-organization pairs such that *active_role-orgs*($s_i$) ⊆ *assigned_role-orgs*(*user*($s_i$));
- *can_access*: S × Op × A → {true, false} -- a predicate defined as *can_access(s, op, a)* is true iff ∃ (*r, o*) ∈ *active_role-orgs(s)* ∧ *aorg(a)* = *o* ∧ ((*op*, *atype(a)*), *r*) ∈ PA ;

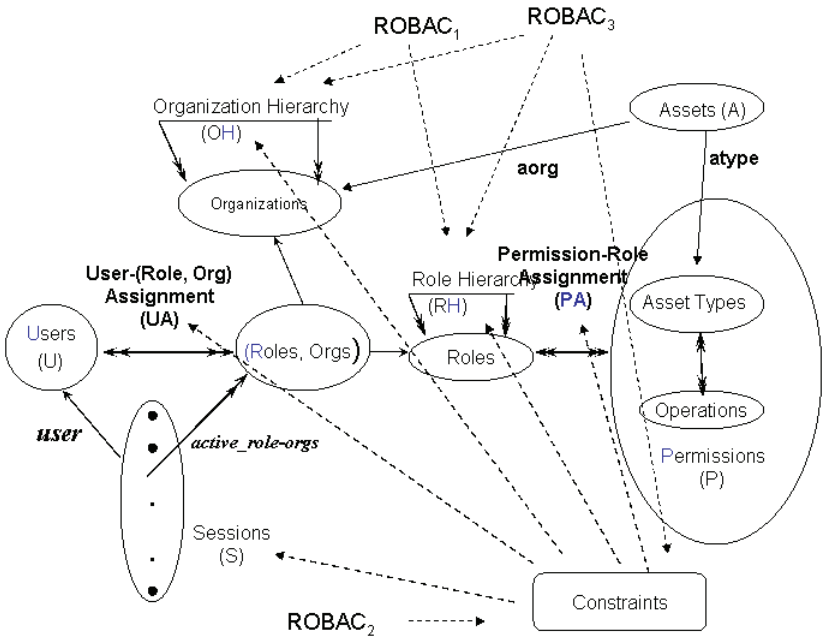
where the operations (Op) are similar to the operations or actions in classic RBAC; the assets (A) are similar to objects; the *active_role-orgs* is used to model activation of role-organization pairs inside a session and it returns a subset of the role and organization pairs that the *assigned_role-orgs* returns. Because certain roles are only meaningful for certain organizations, such as School Principle role is only meaningful for school type organizations, we introduce the set of applicable role and organization pairs (RO) to model that requirement. Briefly, $ROBAC_0$ extends $RBAC_0$ by:

- introducing new sets: Organizations(O), Asset Types (At), and Role-Organization pairs (RO);
- introducing new functions: *atype, aorg;*
- extending *assigned_role* and *roles* (*session_role*) to *assigned_role-orgs* and *active_role-orgs*;
- redefining permissions (P) and user to role assignment (UA);
- introducing a predicate *can_access(s, op, a).*

Any access control system needs to answer the following question: Can a subject perform an operation over an object?

The newly introduced predicate *can_access* serves this purpose in ROBAC. The definition of *can_access* in $ROBAC_0$ indicates that a user (*user(s)*) in a session *s* can perform an operation *op* over an asset *a* if and only if that the user has an active role and organization pair (*r, o*) in that

*Figure 3. A family of ROBAC models*



session and the *r* has a permission to perform the *op* over *a*'s type and *a* is related to *o*.

**Definition 2:** $ROBAC_1$ has the following components:

- U, S, R, O, Op, A, At, P, RO, PA, UA, *user, atype, aorg* are same as those from $ROBAC_0$.
- $OH \subseteq O \times O$ -- a partial order relation on O called organization hierarchy;
- $RH \subseteq R \times R$ -- role hierarchy (same as RH in RBAC96);
- *assigned_role-orgs*: $U \rightarrow 2^{RO}$ -- a function mapping a user to a set of role-organization pairs assigned to the user; formally: $assigned\_role\text{-}orgs(u) = \{ (r,o) \mid \exists r' \geq r \wedge \exists o' \geq o \wedge (u, (r',o')) \in UA \}$;
- *active_role-orgs*: $S \rightarrow 2^{RO}$ -- a function mapping each session $s_i$ to a set of active role-organization pairs such that $active\_role\text{-}orgs(s_i) \subseteq assigned\_role\text{-}orgs(user(s_i))$;
- *can_access*: $S \times Op \times A \rightarrow$ {true, false} – a predicate defined as *can_access(s, op, a)* is true iff $\exists (r, o) \in active\_role\text{-}orgs(s) \wedge aorg(a) \leq o \wedge ( \exists r' \leq r, ((op, atype(a)), r') \in PA )$ ;

$ROBAC_1$ adds OH (organization hierarchy) and RH (role hierarchy) and changes *assigned_role-orgs* function and *can_access* predicate from $ROBAC_0$.

The definition of *can_access* in $ROBAC_1$ means that a user *user(s)* in a session *s* can perform an operation *op* over an asset *a* if and only if that the user has an active role and organization pair (*r, o*) in that session and the role *r* or any of its junior roles has a permission to perform the operation *op* over the asset *a*'s type, and the asset *a* is related to the organization *o* or any of its subordinate organizations.

For the aforementioned B2B example, we can use $ROBAC_1$ to model it very conveniently. We show some ROBAC elements differing from those in RBAC as follows.

- O= {State_1, State_2, District_1, District_2, District_3, School_1, School_2, School_3, School_4, …}
- OH = {(State_1, District_1), (State_1, District_2), (District_1, School_1), (District_1, School_2), (District_2, School_3), (State_2, District_3), (District_3, School_4), …}
- At = {Type_A_Report, Type_B_Report, …}
- RO = { (r1, District_1), (r2, District_1), (r1, School_1), (r2, School_2), (r3, School_1), (r4, School_1), … }

Possible permissions and roles are listed in Table 6 and Table 7.

In this B2B example, ROBAC only creates one role called Type_A_Report_Viewer which has one permission called View_Type_A_Report for viewing type A report, but a classic RBAC needs to

*Table 6. Sample permissions in B2B example (with ROBAC)*

| |
|---|
| p1: View Type A Report |
| p2: View Type B Report |
| p3: View Type C Report |
| p4: View Type D Report |
| … |

*Table 7. Sample roles in B2B example (with ROBAC)*

| |
|---|
| r1: Type A Report Viewer which has permission p1. |
| r2: Type B Report Viewer which has permission p2. |
| r3: Type C Report Viewer which has permission p3. |
| r4: Type D Report Viewer which has permission p4. |
| … |

create a role for each organization's type A report viewer, such as School_1_Type_A_Report_Viewer which has View_School_1_Type_A_Report permission, and School_2_Type_A_Report_Viewer which has View_School_2_Type_A_Report permission, etc.

Based on the security policies, r1 and r2 can have role-organization pairs with all levels of organizations but r3 and r4 can only have role-organization pairs with school level organizations.

For the aforementioned B2C example, we can use $ROBAC_0$. Possible permissions and roles are listed in Table 8 and Table 9.

In this B2C example, ROBAC only creates two roles (parent and student) instead of a parent role and a student role for each family in the classic RBAC.

Permissions in ROBAC are defined as a subset of Op × AT while permissions in classic RBAC are defined as a subset of Op × A. Usually, |AT| is much smaller than |A|. In the above B2B example, $|A| \approx 10{,}000 \times |AT|$

Comparing to RBAC, the number of roles and permissions in RODBC are reduced dramatically in the above B2B and B2C examples. The set of applicable role and organization pairs (RO) is a newly introduced concept in ROBAC. The size of RO may become large when there are a large number of organizations involved. Instead of creating RO explicitly, we can define RO implicitly by using some rules. For example, in the aforementioned B2B example, we use the following rules to establish RO implicitly:

- r1 (Type A Report Viewer) and r2 (Type B Report Viewer) can associate with any organizations.
- r3 (Type C Report Viewer) and r4 (Type D Report Viewer) can only associate with school type organizations.

For the B2C example, we allow any role associate to any organization. While the size of RO may be large, the administrative work for RO is small.

Because the number of roles and permissions in ROBAC is much smaller than that in RBAC under the situations similar to the above two examples, the administrative complexity in Permission-to-Role assignment is significantly reduced. Therefore, using ROBAC to model the problems similar to the above B2B and B2C examples is more succinct and intuitive than using RBAC.

The organization concept in ROBAC introduces a powerful abstraction that can be coupled quite naturally with the traditional concept of roles. For example, we can treat the divisions or project teams in an enterprise as organizations. So ROBAC can also be used in many B2E applications. A user with an assigned role and organization pair *(r, o)* indicates that the user can act as role *r* within the organization *o* and its subordinate organizations. Because ROBAC performs access control based on both roles and association relations between users and protected resources (assets), it is suitable to model privacy-related policies in applications involving a large number of similar organizational units. A detail comparison between ROBAC and other RBAC extensions can be found in Zhang et al, 2006.

*Table 8. Sample permissions in B2C example (with ROBAC)*

| |
|---|
| p1: Update Family Profile |
| p2: View Kid's Progress Reports |
| p3: View Family Profile |
| … |

*Table 9. Sample roles in B2C example (with ROBAC)*

| |
|---|
| r1: Parent which has permission p1 and p2. |
| r2: Student which has permission p2 and p3. |
| … |

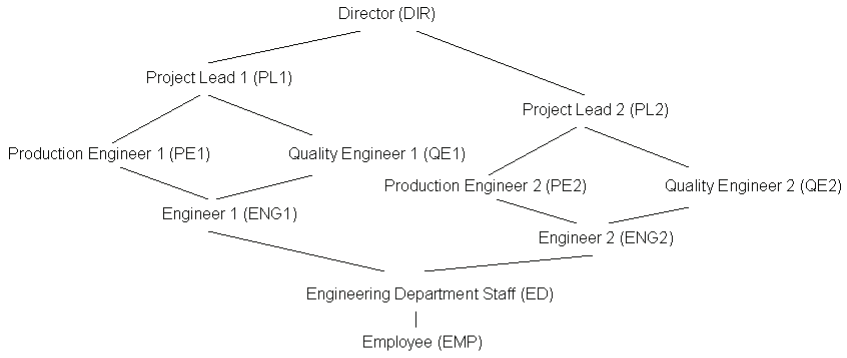
## ADMINISTRATIVE ROBAC

In any security systems, administrative actions need to be controlled. Using role-based method to control RBAC administrative tasks is often a preferred way because it can share the underline authorization mechanism. There are two main approaches to perform RBAC administration. One is centralized such as Gavrila and Barkley's NIST model (1998) and Nyanchama and Osborn's role graph model (1999), where one or more security administrators perform all administrative tasks. Another is decentralized such as Sandhu et al's ARBAC97 model (1999), Crampton and Loizou's SARBAC model (2003), Oh et al's ARBAC02 model (2006), and Bhatti et al's X-GTRBAC admin (2004), where administrative tasks are distributed among many different administrators in a controlled manner. Those role-based decentralized approaches usually add a separate administrative role hierarchy in the original RBAC model. Figure 4 shows an example of regular role hierarchy and administrative role hierarchy created in ARBAC97 model for an engineering department within an organization.

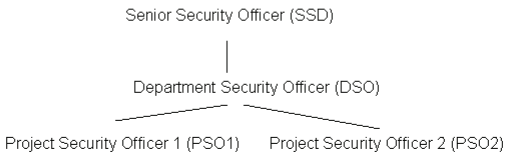
Department Security Officer (DSO) can perform administrative tasks on the department level and Project Security Officer (PSO) can perform administrative tasks on the project level. Each project not only has its own instance of Project Leader role, Production Engineer role, and Quality Engineer role in the regular role hierarchy, but also has its own instance of Project Security Officer role in the administrative role hierarchy. If there are a large number of projects, say in the degree of 100s or more, in an enterprise, both the regular role hierarchy and administrative role hierarchy will become very clumsy and hard to manage correctly. Many classic administrative RBAC models, such as ARBAC97, do not scale up well when a large numbers of similar organizational units are involved.

As we mentioned earlier, the organization concept in ROBAC should not be treated literally.

*Figure 4. Examples of role hierarchy using classic administrative RBAC*




**(a) An example of regular role hierarchy**




**(b) An example of administrative role hierarchy**

For business to employee (B2E) applications, we may treat the divisions or project teams in an enterprise as organizations. ROBAC model is suitable to be used in large enterprises where there are many similar organizational units. A decentralized administrative approach is preferred for large enterprises. Based on the observation that administrative tasks are very similar in many enterprises, we believe that it is an effective approach to utilize the role and organization based access control concept to manage administrative tasks in ROBAC. The main topic of this section is to present a comprehensive model for role and organization based administration of ROBAC.

In classic RBAC, major administrative tasks include assigning users to roles, assigning permissions to roles, and adjusting role hierarchy. So some role-based administrative model, such as ARBAC97, has three separate sub-models: URA97 (user-role assignment), PRA97 (permission-role assignment), and RRA97 (role-role assignment), to deal with these three major administrative tasks. There are more components in ROBAC than those in RBAC, making the administration of ROBAC more multifaceted than RBAC. Following the ARBAC97 approach, we divide administrative tasks into the following categories: assigning users to role-organization pairs, assigning permissions to roles, managing roles and role hierarchy, managing organizations and organization hierarchy, and managing role and organization association. The reason is that these administrative activities in ROBAC affect user's access rights in different ways. Our administrative model is called AROBAC07 (administrative ROBAC '07). It has five components.

1. UROA07 (user to role and organization pair assignment '07) is concerned with user to role and organization pair assignment;
2. PRA07 (permission to role assignment '07) deals with permission-role assignment;
3. RRA07 (role to role assignment '07) manages roles and role hierarchy;
4. OOA07 (organization to organization assignment '07) handles organizations and organization hierarchy; and
5. ROA07 (role to organization assignment '07) controls applicable association between roles and organizations.

The development of AROBAC07 was heavily influenced by ARBAC97, SARBAC, and ARBAC02. Our AROBAC07 model is presented in the context of $ROBAC_1$. Its interpretation for $ROBAC_0$, $ROBAC_2$, and $ROBAC_3$ is straightforward.

AROBAC07 adds some additional sets, relationships, and functions to ROBAC model. Similar to ROBAC, an administrative user is assigned to administrative role and organization pairs instead of administrative role only. An administrative decision is made based on both role and organization information. The common elements of AROBAC07 are described in Definition 3 and Figure 5 shows some relationship among the elements of AROBAC07.

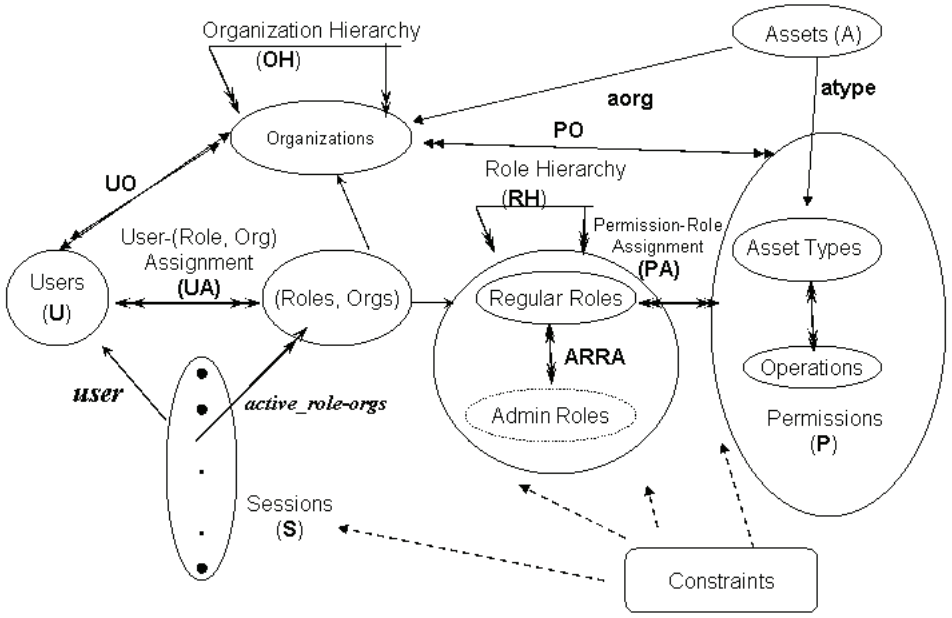
**Definition 3:** AROBAC07 has the following components:

- U, S, O, OH, Op, A, At, P, RO, PA, UA, *user, atype, aorg, assigned_role-orgs*, *active_role-orgs, can_access* are same as those from ROBAC;
- RR -- a set of regular roles (renamed R in ROBAC);
- $RRH \subseteq RR \times RR$ – regular role hierarchy (renamed RH in ROBAC);
- AR -- a set of administrative roles (same as AR in ARBAC97), where $RR \cap AR = \varnothing$.
- $ARH \subseteq AR \times AR$ -- administrative role hierarchy (same as ARH in ARBAC97);
- $R = RR \cup AR$ -- the set of all roles;
- $ARRA \subseteq AR \times RR$ -- a many-to-many administrative role to regular role assignment;

- RH = RRH ∪ ARH -- a combined role hierarchy;
- UO ⊆ U × O -- a set of user-organization affiliations;
- PO ⊆ P × O -- a set of applicable permission-organization associations;
- CRU – a set of applicable prerequisite condition for users;
- CRP – a set of applicable prerequisite condition for permissions;
- CAN_ASSIGN_USER ⊆ ARRA × CRU - an association defines the constraints when assigning users to role-organization pairs;
- CAN_REVOKE_USER ⊆ ARRA × CRU - an association defines the constraints when revoking users from role-organization pairs;
- *can_assign_user*: S × U × RO → {true, false} – a predicate which indicates that if *can_assign_user(s, u, (r,o))* is true then user *u* can be assigned to the role-org pair *(r,o)* within the session *s* (the definition is described in UROA07);
- *can_revoke_user*: S × U × RO → {true, false} – a predicate which indicates that if *can_revoke_user(s, u, (r,o), c)* is true then user *u* can be revoked from role-org pair *(r,o)* within the session *s* (the definition is described in UROA07);
- CAN_ASSIGN_PERMISSION ⊆ ARRA × CRP - an association defines the constraints when assigning permissions to roles;
- CAN_REVOKE_PERMISSION ⊆ ARRA × CRP - an association defines the constraints when revoking permissions from roles;
- *can_assign_permission*: S × P × RR → {true, false} – a predicate which indicates that if *can_assign_permission(s, p, r)* is true then the permission *p* can be assigned to the regular role *r* within the session *s* (the definition is described in PRA07);
- *can_revoke_permission*: S × P × RR → {true, false} – a predicate which indicates that if *can_revoke_permission(s, p, r)* is true then the permission *p* can be revoked from the regular role *r* within the session *s* (the definition is described in PRA07);
- *can_modify_R*: S × $2^{RR}$ → {true, false} -- a predicate which indicates that if *can_modify_R*(*s, rset*) is true then the user *user(s)* can modify the roles and their relationship inside the role set *rset* within the session *s* (the definition is described in RRA07);
- *can_modify_O*: S × $2^{O}$ → {true, false} -- a predicate which indicates that if *can_modify_O*(*s, oset*) is true then the user *user(s)* can modify the organizations and their relationship inside the organization set *oset* within the session *s* (the definition is described in OOA07);
- *can_modify_RO*: S × R × O → {true, false} -- a predicate which indicates that if *can_modify_RO*(*s, r, o*) is true then the user *user(s)* can associate or disassociate role *r* with organization *o* within the session *s* (the definition is described in ROA07);

In AROBAC97, UO defines user's organization affiliation. A user may be affiliated with multiple organizations. UO is usually pre-determined by Human Resource (HR) departments of individual organizations. The applicable permission-organization association set PO defines permissions applicable to organizations. Similar to permission set P, PO is pre-determined via joined efforts between HR and Information Technology (IT) departments. So we do not include the management of UO, P, and PO in our model. The set ARRA can be considered as the administrative role's permission over the regular roles. Some constraints need to be enforced when creating or modifying the role hierarchy (RH) or organization hierarchy (OH) such as no circular reference. The detailed descriptions of the prerequisite condition sets CRU and CRP, the predicates *can_assign_user, can_revoke_user, can_assign_permission, can_revoke_permission, can_modify_R, can_modify_O,* and *can_modify_RO* are discussed in the following corresponding subsections.

*Figure 5. AROBAC07 model*




## The UROA07 Model

The UROA07 model deals with managing user to role-organization pair assignment. It provides two predicates to determine whether the current session can grant a user membership in a role-organization pair (or simply role-org pair) or revoke a user membership in a role-org pair. Before introducing the details of UROA07 model we need some definitions.

*user prerequisite condition (upc)* - a *upc* is a boolean expression using the usual $\wedge$ and $\vee$ operators on terms of form of *(r, ?), ¬(r, ?), (r, o),* and *¬(r, o)* where *(r, o)* is a role-org pair belongs to RO. A *user prerequisite condition* is evaluated for a user *u* by interpreting *(r, o)* to be true if ($\exists r' \geq r$, $\exists o' \geq o$) (*u, (r', o')*) $\in$ UA and *¬(r, o)* is true if *(r, o)* is not true. Here "*?*" is a place holder for any $o \in$ O, and *(r, ?)* is true for user *u* if ($\exists r' \geq r$ , $\exists o' \geq ?$, (*u, (r', o')*) $\in$ UA ) and *¬(r, ?)* is true if *(r, ?)* is not true. CRU is a set including all applicable *upc*s plus a *null* element. The *null* is interpreted as true for any user.

**Note:** The *(r, ?)* expression represents a condition template where the value of "*?*" is set to *o* when the system is asked whether a user can be assigned to a role-organization pair *(r, o)*. We will explain it in an example later.

*omembers*: O $\rightarrow 2^{U}$, is a function mapping an organization to a set of users who affiliated with the organization; formally, *omembers*(*o*) = { *u* | *(u, o)* $\in$ UO }; *omembers**(*o*) = { *u* | $\exists o' \leq o$, *(u, o')* $\in$ UO }

**Note:** *omembers*(*o*) is the set of all users affiliated with organization *o* and *omembers**(*o*) is the set of all users affiliated with organization *o* or its subordinate organizations.

*apermissions*: AR $\rightarrow 2^{RR}$, is a function mapping an administrative role to a set of regular roles which the administrative role has administrative privilege over;

formally, $apermissions(ar) = \{ r \mid (ar, r) \in ARRA \}$; $apermissions^*(ar) = \{r \mid \exists ar' \leq ar, (ar', r) \in ARRA \}$

**Note:** *apermisions*(*ar*) is the set of regular roles which the administrative role *ar* has administrative privilege and *apermissions**(*ar*) is the set of regular roles which administrative role *ar* or its junior administrative roles has administrative privilege.

*may_manage_user*: AR × U × RO × CRU → {true, false} - a predicate defined as *may_manage_user(ar, u, (r,o), c)* is true iff $(r \in apermissions^*(ar)) \wedge c \wedge (u \in omembers^*(o))$.

**Note:** The definition of *may_manage_user(ar, u, (r,o), c)* indicates that a user with administrative role *ar* may manage the user *u* with respect to the role-org pair *(r, o)* if and only if the user *u* satisfies the user prerequisite condition *c* and is affiliated to the organization *o* or its subordinate organizations and the administrative role *ar* or its junior administrative roles can perform administrative tasks on role *r*. The *may_manage_user* predicate is used as a sub-routine in the following UROA07 Grant Model and UROA07 Revoke Model.

**Definition 4:** The UROA07 Grant Model – *can_assign_user* predicate controls whether a user can be assigned to a role-org pair within a session. Formally, *can_assign_user(s, u, (r,o))* is true iff $(\exists o' \geq o, \exists (ar, o') \in active\_role\text{-}orgs(s)) \wedge (\forall c \in CRU, ((ar, r), c) \in CAN\_ASSIGN\_USER \wedge may\_manage\_user(ar, u, (r,o), c))$.

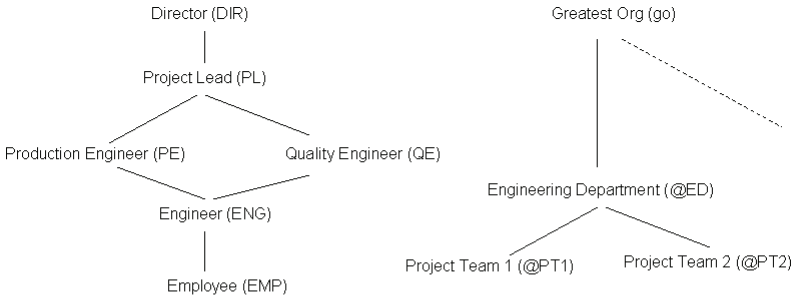
The definition of *can_assign_user(s, u, (r,o))* in UROA07 indicates that a user (*user(s)*) in a session *s* can assign a user *u* to a role-org pair *(r, o)* if and only if *user(s)* has an active role-org pair *(ar, o)* (explicitly or implicitly via organization hierarchy) in that session and user *u* satisfies all related user prerequisite conditions defined in CAN_ASSIGN_USER and is affiliated to the organization *o* or its subordinate organizations and the administrative role *ar* or its junior administrative roles can perform administrative tasks on role *r*.

The user prerequisite condition in UROA07 is likely to be empty in most cases. It may be used to model some complex policy, so we include it in the model.

To appreciate the benefit behind the UROA07 model, let us remodel the aforementioned engineering department example in Figure 4 using AROBAC07. Figure 6 shows the AROBAC07 model for the engineering department problem in Figure 4. Here we treat a project team as an organization. We further assume that roles in different project teams within the engineering department perform similar tasks and a team member only can access the resource related to the team he/she is in. This assumption will usually hold because most enterprises need to enforce unified security policy across multiple teams.

We can see that both the regular role hierarchy (Figure 6(a)) and the administrative role hierarchy (Figure 6(c)) in AROBAC07 are simpler. For the moment please ignore the *Greatest Administrative Role* (*gar*) and the *Greatest Organization* (*go*). We will explain these later. Here we prefix "@" in the front of organizations to distinguish them from roles. For example, a user with an active role-org pair (PSO, @PT1) is a security administrator in project team 1. *may_manage_user*(PSO, *u, (*PE, @PT1*),* ¬(QE, ?)) is true if user *u* is affiliated with project team 1 (@PT1) and *u* is not a QE inside the project team 1. Based on the UROA07 grant-model, the user with active role-org pair (PSO, @PT1) can assign membership of roles: PL, PE, QE, and ENG within project team 1, to users affiliated with the project team 1 but he/she cannot assign these users to roles within project team 2 and cannot assign users not affiliated to project team 1 to any roles. He/she also cannot assign both (PE, @PT1) and (QE, @PT1) to the same user because of the user prerequisite conditions, ((PSO, PE), ¬(QE, ?)) and ((PSO, QE),

*Figure 6. Role and organization hierarchies using administrative ROBAC*




**ARRA**

| Admin Role | Regular Role |
|---|---|
| DSO | DIR |
| PSO | PL |
| PSO | PE |
| PSO | QE |
| PSO | ENG |
| gar | EMP |

**CAN_ASSIGN_USER**

| Admin Role | Regular Role | User Prereq. Cond. |
|---|---|---|
| DSO | DIR | null |
| PSO | PL | null |
| PSO | PE | ¬(QE, ?) |
| PSO | QE | ¬(PE, ?) |
| PSO | ENG | null |
| gar | EMP | null |

(d) ARPA and CAN_ASSIGN_USER Relations

¬(PE, ?)) , defined in CAN_ASSIGN_USER at Figure 6(d), which represents a global separation of duty constraint in ROBAC.

If there are more project teams in the engineering department or there are more engineering departments, we only need to add them in the organization set O and organization hierarchy OH but do not need to change other settings in ROBAC. Even with 100s or more project teams, the UROA07 model will scale up very nicely whereas previous models will become incomprehensible.

To complete the definition of the UROA07 model we define the Revoke Model as follows.

**Definition 5:** The UROA07 Revoke Model – *can_revoke_user* predicate controls whether a user can be revoked from a role-org pair within a session. Formally, *can_revoke_user(s, u, (r,o))* is true iff ($\exists o' \geq o, \exists (ar, o') \in$ *active_role-orgs*(*s*) ) $\wedge$ ($\forall c$, *((ar, r), c)* $\in$ CAN_REVOKE_USER $\wedge$ *may_manage_user(ar, u, (r,o), c)*).

## The PRA07 Model

The PRA07 model deals with managing permission to role assignment. Similar to UROA07, it also provides two predicates to determine whether a session can assign or revoke permission to or from a role. We give the following definitions in analogy to similar definitions for UROA07.

*permission prerequisite condition (ppc)* – a *ppc* is a boolean expression using the usual $\wedge$ and $\vee$ operators on terms of form of *r* and $\neg r$ where *r* is a role in RR. A *permission prerequisite condition* is evaluated for a permission *p* by interpreting *r* to be true if ($\exists r' \leq r$, $(p, r') \in$ PA) and $\neg r$ is true if ($\forall r' \geq r$, $(p, r') \notin$ PA. CRP is a set which includes all applicable *ppc*s plus a *null* element. The *null* is interpreted as true for any permission.

*opermissions*: O → $2^P$, is a function mapping an organization to a set of permissions which applicable to the organization;

formally, *opermissions*(*o*) = { *p*: P | *(p, o)* ∈ PO }; *opermissions**(*o*) = { *p*: P | ∃*o'* ≤ *o, (p, o')* ∈ PO }

**Note:** *opermissions*(*o*) is the set of all permissions applicable to the organization *o* and *opermissions**(*o*) is the set of all permissions applicable to the organization *o* or its subordinate organizations.

*can_manage_permission*: RO × P × RR × CRP → {true, false} – a predicate defined as *can_manage_permission((ar, o), p, r, c)* is true iff (*r* ∈ *apermissions**(*ar*)) ∧ *c* ∧ (*p* ∈ *omembers**(*o*)).

**Note:** The definition of *can_manage_permission((ar, o), p, r, c)* indicates that a user who is a member of administrative role-org pair (*ar, o)* can manage the permission *p* for the role *r* if and only if permission *p* satisfies the permission prerequisite condition *c* and is applicable to the organization *o* or its subordinate organizations and the administrative role *ar* or its junior administrative roles can perform administrative tasks on the regular role *r*.

This leads to the following Grant Model.

**Definition 6:** The PRA07 Grant Model – *can_assign_permission* predicate controls whether a permission can be assigned to a role within a session. Formally, *can_assign_permission(s, p, r)* is true iff ∃*(ar, o)* ∈ *active_role-orgs*(*s*) ∧ (∀*c*, *((ar, r), c)* ∈ CAN_ASSIGN_PERMISSION ∧ *can_manage_permission((ar, o), p, r, c)*.

The definition of *can_assign_permission(s, p, r)* in PRA07 indicates that a user (*user(s)*) in a session *s* can assign a permission *p* to a role *r* if and only if *user(s)* has an active role-org pair *(ar, o)* in that session, and the administrative role *ar* or its junior administrative roles have administrative right over the regular role *r,* and the permission *p* is applicable to the organization *o* or its subordinate organizations, and the permission *p* satisfies all specified permission prerequisite conditions defined in CAN_ASSIGN_PERMISSION. The permission prerequisite condition in PRA07 is optional.

Finally we have the following Revoke Model.

**Definition 7:** The PRA07 Revoke Model – *can_revoke_permission* predicate controls whether a permission can be revoked from a role within a session. Formally, *can_revoke_permission(s, p, r)* is true iff (∃*(ar, o)* ∈ *active_role-orgs*(*s*) ∧ (∀*c*, *((ar, r), c)* ∈ CAN_REVOKE_PERMISSION ∧ *can_manage_permission((ar, o), r, c)*).

## The RRA07 Model

The RRA07 model deals with managing roles and role hierarchy. It provides one predicate called *can_modify_R* to determine whether the current session can add/remove a role or change role hierarchy during the session. To define *can_modify_R* predicate, we need to introduce some definitions.

*rjuniors*: R → $2^R$, is a function mapping a role to its junior roles; formally, *rjuniors*(*r*) = { *r'*: R | *r'* < *r* }

*rseniors*: R → $2^R$, is a function mapping a role to its senior roles; formally, *rseniors*(*r*) = { *r'*: R | *r'* > *r* }

*rfamily*: R → $2^R$, is a function mapping a role to a set of roles including itself and its junior role and senior roles; formally, *rfamily*(*r*) = {*r*} ∪ *rjuniors*(*r*) ∪ *rseniors*(*r*)

*rfamilies*: $2^R$ → $2^R$, is a function mapping a set of roles to a set of roles including all families of its members; formally, *rfamilies*({$r_1$, $r_2$, ... $r_n$}) = *rfamily*($r_1$) ∪ *rfamily*($r_2$) ∪ … ∪ *rfamily*($r_n$)

It is worth noting that *rfamily* and *rfamilies* only include recursive direct family members and do not include siblings.

*permissible administrative role set*, *parset*: AR → $2^{RR}$, is a function mapping an administrative role to a set of regular roles in which the administrative role can modify the regular role hierarchy. Formally,

$parset(ar) = \{ r : RR \mid (ar, r) \in ARRA \wedge rfamily(r) \subseteq apermissions^*(rfamily(ar)) \}$.

The above definition indicates that *parset* for an administrative role *ar* includes all of the regular roles it has administrative privilege such that the regular role's family is a part of the regular roles the *ar*'s family has administrative privilege over. For example *parset*(PSO) = { PL, PE }.

Because modifying role hierarchy affects all organizations in ROBAC, we should only allow the users at the highest organization level to perform these actions. We introduce an artificial organization called *greatest organization* (*go*) which is the ancestor for all organizations in O (see Figure 3). Now let us define the *can_modify_R* predicate.

**Definition 8:** *can_modify_R*: $S \times 2^{RR} \rightarrow$ {true, false} -- a predicate defined as *can_modify_R*(*s, rset*) is true iff $\exists$*(ar, go)* $\in$ *active_role-orgs*(*s*) $\wedge$ *rset* $\subseteq$ *parset*(*ar*).

The definition of *can_modify_R* means that a user *user(s)* in a session *s* can modify the relationship within the role set *rset* if and only if that the user has an active administrative role *ar* and paired with the greatest organization *go* in that session and the role set *rset* is a subset of the *permissible administrative role set* of *ar*. Here the modification within a set of roles means adding/deleting an edge or adding/removing a role. For example, according to Figure 6, PSO can remove the edge between PL and PE but cannot remove the edge between ENG and QE because QE and ENG are not in its *permissible administrative role set*.

## The OOA07 Model

The OOA07 model deals with managing organizations and organization hierarchy. Similar to RRA07, it provides one predicate called *can_modify_O* to determine whether the current session can add/remove an organization or change organization hierarchy during the session. To define *can_modify_O* predicate, we also need to introduce some definitions.

*ojuniors*: O → $2^O$, is a function mapping an organization to its subordinate organizations.

Formally, $ojuniors(o) = \{ o' : O \mid o' < o \}$

*oseniors*: O → $2^O$, is a function mapping an organization to its parent organizations.

Formally, $oseniors(o) = \{ o' : O \mid o' > o \}$

*ofamily*: O → $2^O$, is a function mapping an organization to a set of organizations including itself and its subordinate organizations and parent organizations. Formally, $ofamily(o) = \{o\} \cup ojuniors(o) \cup oseniors(o)$

*permissible administrative organization set*, *paoset*: O → $2^O$, is a function mapping an organization to a set of organizations. Formally, $paoset(o) = \{ o' : O \mid o' < o \wedge ofamily(o') \subseteq ofamily(o) \}$

For example, *paoset*(@ED) = { @PT1, @PT2 } according to Figure 6(b).

Because modifying organization hierarchy has some global effects on the access right in ROBAC, we should only allow the most senior administrative role to modify the organization hierarchy. We can pre-define a most senior administrative role called *gar* (*greatest administrative role*) in

AR. Here is the definition for *can_modify_O* predicate.

**Definition 9:** *can_modify_O*: $S \times 2^O \rightarrow$ {true, false} -- a predicate defined as *can_modify_O*(*s, oset*) is true iff $\exists$*(gar, o)* $\in$ *active_role-orgs*(*s*) $\wedge$ *oset* $\subseteq$ *paroet*(*o*).

The definition of *can_modify_O* means that a user *user(s)* in a session *s* can modify the relationship within the organization set *oset* if and only if that the user has the greatest administrative role *gar* in an organization *o* in that session and the organization set *oset* is a subset of the *permissible administrative organization set* of *o*. For example, a user assigned (gar, @ED) can remove @PT1 from the organization hierarchy or add a new organization say @PT3 under it.

## The ROA07 Model

The ROA07 model deals with managing role and organization association in ROBAC. Similar to other models, it provides one predicate called *can_modify_RO* to determine whether the current session can associate / disassociate a role with an organization.

**Definition 10:** *can_modify_RO*: $S \times R \times O \rightarrow$ {true, false} -- a predicate defined as *can_modify_RO*(*s, r, o*) is true iff ($\exists o' \geq o$, $\exists$*(ar, o')* $\in$ *active_role-orgs*(*s*)) $\wedge$ $r \in$ *apermissions**(*ar*).

The definition of *can_modify_RO* means that a user *user(s)* in a session *s* can associate or disassociate the role *r* with the organization *o* if and only if *user(s)* has an active role and organization pair *(ar, o')* in that session and *o* is *o'* or a subordinate of *o'* and the administrative role *ar* or its junior administrative roles can perform administrative tasks on the role *r*. For example, a user assigned (PSO, @PT1) can associate PE with @PT1 but cannot disassociate PE from @PT2.

The five sub-models in AROBAC07 decentralize the administrative tasks along the administrative role hierarchy and organization hierarchy. They control administrative tasks based on both administrative role permissions and organization hierarchy. This is a ROBAC approach to perform administrative work on ROBAC systems.

## Discussion and Related Work

ROBAC models require that organizations have similar roles. Administrator roles across organizations tend to have greater similarity than the underlying regular roles may have. So the ROBAC concept is particularly well suited to administrative tasks. A user assigned an administrative role and organization pair *(ar, o)* can perform administrative tasks the *ar* allowed within the organization *o* and its subordinate organizations. Administrative tasks in AROBAC07 can be delegated not only along the administrative role hierarchy but also along the organization hierarchy.

As we mentioned earlier, AROBAC07 model is inspired by the following three role-based administrative models: ARBAC97, ARBAC02, and SARBAC.

ARBAC97 is one of the most comprehensive role-based administrative models. It uses *role range (encapsulated range)* concept to define the scope of administrative permission. The *user prerequisite condition (upc)* and *permission prerequisite condition (ppc)* in AROBAC07 are extended versions of the corresponding prerequisite conditions in ARBAC97.

ARBAC02 enhances ARBAC97 by incorporating two external organization structures: *user organization structure* (OS-U) and *permission organization structure* (OS-P). URA02 and PRA02 sub-models in ARBAC02 modify the prerequisite conditions in URA97 and PRA97 of ARABC97 by using memberships of organizations to avoid some weakness, such as multi-step assignments, redundant assignment information, restricted

hierarchy etc., in ARBAC97. ARBAC02 uses the organization structures only for constructing user pool and permission pool in prerequisite conditions but does not use it to control administrative permissions. UROA07 and PRA07 in AROBAC07 use the build-in organization hierarchy and implicitly enforce the user pool and permission pool constraints with respect to *can_assign_user* and *can_revoke_user* and *can_assign_permission* and *can_revoke_permission* predicates. So UROA07 and PRA07 have similar benefits of user pool and permission pool concepts used in URA02 and PRA02 models without putting the user pool or permission pool as a part of the prerequisite conditions. Similar to ARBAC02, AROBAC07 avoids the weakness of ARBAC97 by using the build-in organization hierarchy, the user-organization affiliation information, and the applicable permissions and organization association information.

SARBAC introduces a concept called *administrative scope*, which can be calculated for each role-based on the role hierarchy. SARBAC tends to be simpler, more flexible, and more permissible than ARBAC97. The ARRA relation in AROBAC07 is similar to the **can_admin** relation in SARBAC. The current construction of *permissible administrative role set (parset)* concept in RRA07 is similar to the *administrative scope* concept in SARBAC. From that perspective, RRA07 is very similar to $RH_4$ sub-model in SARBAC. So RRA07 has some similar benefits, such as flexibility, that SARBC enjoys. Unlike $RH_4$, RRA07 enforces strict separation between regular roles and administrative roles because we believe the separation of administrative duty from regular duty is a desired security policy in most cases. It is possible that we can construct *parset* differently, such as using the *encapsulated range* concept in RRA97 or other criteria. The OOA07 applies the similar concept on the role hierarchy.

X-GTRBAC admin (Bhatti et al, 2004) is a decentralized administrative model for the XML-based Generalized Temporal Role-based Access Control (X-GTRBAC) framework (Bhatti et al, 2003). It uses a concept called *admin domain* to tire roles (admin roles and regular roles), permissions (admin permissions and regular permissions), and users (admin users and regular users) together. It uses hard-coded Eligible Role (ER) for users in admin domain to put constraint on user to role assignment task. It uses Admin Permissions to model the possible administrative tasks. An administrator in an Admin Role is authorized to handle assignment of users to regular roles within a given domain. Both administrative roles and admin domains in X-GTRBAC admin are flat. If considering an admin domain as an organization, X-GTRBAC admin's impact on X-GTRBAC framework is similar to the UROA07's impact on $ROBAC_0$ based system. It is hard for X-GTRBAC admin to achieve similar functionality AROBAC07 has when organization hierarchy or/and administrative hierarchy exist.

The first three of aforementioned role-based administrative models control administrative tasks based on roles only while AROBAC07 controls administrative tasks based on both roles and organizations. From the perspective of administrative units, the admin domain concept in X-GTRBAC admin is similar to the organization concept in UROA07 for $ROBAC_0$ based systems, but AROBAC07 has more functionality that the X-GTRBAC admin seems hard to achieve.

In AROBAC07, we use the same ARRA relation across all sub-models. It may be desirable that we use a different version of ARRA in each sub-model when finer-grain control is needed. The administrative role hierarchy in AROBAC07 tends to be simpler than these existing models when there are many organizational units, such as lots of branches or project teams, in an enterprise. With ROBAC/AROBAC07, security policies can be defined in a small scope first and then applied to all organizations.

## Practical Consideration when Applying ROBAC to Web Applications

In real world, an authorization service needs to support many applications in an enterprise. AROBAC07 model not only can decentralize the administrative tasks along organization hierarchy, but also can decentralize the tasks along the administrative role hierarchy. Creating appropriate administrative roles and administrative role hierarchy is critical to achieve decentralized administration in ROBAC system.

## ACom Based Delegation Model

Each application normally has its own set of roles and these roles are only applicable to the application. So the regular role set may be partitioned based on which application the roles belong to. Here we introduce the concept of application compartment. We denote **App** as a set of all applications controlled by a ROBAC system.

**Definition 11:** An Application Compartment (ACom) for an application $app_i \in$ **App** in a ROBAC system is a tuple: ($U_i$, $R_i$ $O_i$, $P_i$, $RO_i$, $RH_i$, $OH_i$, $PA_i$, $UA_i$), where:

- $U_i = U$ -- the same set of users as in ROABC;
- $R_i \subseteq R$ -- a subset of roles applicable to this application;
- $O_i \subseteq O$ -- a subset of organizations applicable to this application;
- $P_i \subseteq P$ -- a subset of permissions applicable to this application;
- $RO_i \subseteq R_i$ x $O_i$ –- a set of applicable role and organization association in this application;
- $RH_i \subseteq R_i$ x $R_i$ -- a partial order relation on $R_i$ called role hierarchy;
- $OH_i \subseteq O_i$ x $O_i$ -- a partial order relation on $O_i$ called organization hierarchy;
- $PA_i \subseteq P_i$ x $R_i$ -- a many-to-many permission-to-role assignment relation;
- $UA_i \subseteq U_i$ x $RO_i$ -- a many-to-many user-to-role-and-organization assignment relation;

Let's use $acom(app_i)$ to represent the ACom corresponding to application $app_i$ and denote ACOM as a set of all application compartments in the ROBAC system. We can think of $acom(app_i)$ as being controlled by a sub ROBAC system. Here we define a dominate relation DOM in ACOM.

**Definition 12:** DOM $\subseteq$ ACOM $\times$ ACOM -- is a partial order relation on ACOM such that ( $acom(app_i)$, $acom(app_j)$) $\in$ DOM iff $U_i \subseteq U_j \wedge R_i \subseteq R_j \wedge O_i \subseteq O_j \wedge P_i \subseteq P_j \wedge RO_i \subseteq RO_j \wedge RH_i \subseteq RH_j \wedge OH_i \subseteq OH_j \wedge PA_i \subseteq PA_j \wedge UA_i \subseteq UA_j$.

For administration purpose, we can create an administrative role $ar_i$ for each $acom(app_i)$. That requires adding $ar_i$ in AR, adding $\{ ar_i \} \times R_i$ into ARRA, and adding $(ar_i , ar_j)$ in ARH iff ( $acom(app_j)$, $acom(app_i)$ ) $\in$ DOM.

The partitioning process may continue within an application. In general, we can form a hierarchy in AR based on the DOM relationship. The senior administrative role can delegate the administrative tasks to its junior administrative roles safely. ACom concept not only provides a way to partition the regular role set (and construct administrative role hierarchy) in AROBAC07, but also can be used to enforce application boundary. For example, when a user *u* enters an application, say $app_i$, the ROBAC system only actives the role-org pairs applicable to this application. That is,

if a user *u* is inside $app_i$ then *active_roles-orgs(u)* $\subseteq$ *assigned_role-orgs(u)* $\cap$ $acom($app$_i).RO_i$.

where $acom($app$_i).RO_i$ represents the role-org pairs applicable to the application $app_i$. You may notice

that AROBAC07 does not explicitly talk about how to assign / revoke users to/from administrative role and organization pairs. In practice, a user with administrative role and organization pair *(ar, o)* can assign (or revoke) users to (or from) administrative role and org pair *(ar', o')* such that $ar' \leq ar$ and $o' \leq o$. It is not hard to develop a formal sub-model similar to the UROA07 model if more complex constraints are needed. So a user assigned with *(gar, @go)* can act like a super user in a ROBAC system. The syntax of assigning user to role and organization pair is same regardless a role is a regular role or an administrative role. This feature makes AROBAC07 easy to implement in practice.

A partial implementation of a $ROBAC_1$ model has been successfully used as an authorization engine, which provides authorization services for multiple web applications (https://epl.collegeboard.com/epl/goHome.do).

## CONCLUSION

A family of extended RBAC models called Role and Organization Based Access Control (ROBAC) models is reviewed and its corresponding administrative model called AROBAC07 is presented and formalized in this chapter. The motivation behind ROBAC is to scale up RBAC for B2B and B2C applications where a large number of organizational units are involved. The advantages of ROBAC models over traditional RBAC models are shown via two examples. We claim that ROBAC is more intuitive and succinct than many RBAC variants when used for scenarios involving a large number of similar organizational units and the privacy issue is a major concern.

The AROABC07 is a decentralized role and organization based administrative model for ROBAC. It has five sub-models: UROA07 is concerned with user to role and organization pair assignment; PRA07 deals with permission-role assignment; RRA07 manages roles and role hierarchy; OOA07 handles organizations and organization hierarchy; and ROA07 controls applicable association between roles and organizations. UROA07 and PRA07 obtain the benefits of user pool and permission pool concepts in ARBAC02 without specifying user pool and permission pool explicitly. RRA07 has the similar advantage of $RHA_4$ in SARBAC over RRA97 in ARBAC97 without sacrificing the separation of duty between administrative roles and regular roles. OOA07 and ROA07 provide ways to decentralize the administrative tasks on organization hierarchy and applicable role and organization association. We claim that AROBAC07 scales up well and is better than using existing role-based administrative models by providing more controlled and decentralized approaches to perform administrative tasks on ROBAC.

Many serious security breaches are due to internal users. So it is very important to restrict and control administrative actions on access control systems. ROBAC/AROBAC07 scales up classic RBAC systems for situations where many similar organizational units are involved. It inherits the RBAC's good features and provides a way to restrict access control within specified organizational units without introducing too much administrative burden on access control systems. It is quite suitable for modeling privacy related security policy.

The implication of *can_modify_R*, *can_modify_O*, and *can_modify_RO* predicates on different administrative tasks such as add/delete nodes or edges need to be detailed and studied further. How to manage other aspects of ROBAC such as integrating general constraints and defining *atype* and *aorg* functions may also merit further study.

## REFERENCES

American National Standard Institute. ANSI INCITS 359-2004 for Role-based Access Control. 2004

Bertino, E., Bonatti, P. A., & Ferrari, E. (2001, August). TRBAC: A temporal role-based access control model. ACM Transactions on Information & System Security, 4(3), 191-233.

Bertino, E., Catania, B., Damiani, M. L., & Perlasca, P. (2005, June). Access control model I: GEO-RBAC: a spatially aware RBAC. *Proceedings of the tenth ACM symposium on Access control models and technologies*.

Bhatti, R. (2003). *X-GTRBAC: An XML-based Policy Specification Framework and Architecture for Enterprise Wide Access Control*. Masters thesis, Purdue University, May 2003.

Bhatti, R., Joshi, J., Bertino, E., & Ghafoor, A. (2004, June). Role administration: X-GTRBAC admin: a decentralized administration model for enterprise wide access control. *Proceedings of the ninth ACM symposium on Access control models and technologies.*

Crampton, J., & Loizou, G. (2003, May). Administrative Scope: A Foundation for Role-Based Administrative Models. *ACM Transactions on Information and System Security, 6*(2), 201-231.

Covington, M. J., Long, Srinivasan, S., Dev, A. K., Ahamad, M., & Abowd, G. D. (2001, May). Securing context-aware applications using environment roles. *Proceedings of the sixth ACM symposium on Access control models and technologies*.

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

Ferraiolo, D., Sandhu, R., Gavrila, S., Kuhn, D., & Chandramouli, R. (2001). Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC), 4*(3).

Gavrila, S., & Barkley, J. (1998, October). Formal Specification for RBAC User/Role and Role/Role Relationship Management. *Proceedings of Third ACM Workshop on Role-Based Access Control*. Fairfax, VA, USA.

Giuri, L., & Iglio, P. (1997). Role Templates for Content-Based Access Control. *Proceedings of Second ACM Workshop on Role-Based Access Control*, Fairfax, VA, USA, November 1997

Georgiadis, C. K., Mavridis, I., Pangalos, G., & Thomas, R. K. (2001). Flexible Team-Based Access Control Using Contexts. *Proceedings of Sixth ACM Symposium on Access Control Models*, May 2001, Fairfax, Virginia, USA.

Joshi, J. B. D., Bertino, E., Latif, U., & Ghafoor, A. (2005). A generalized temporal role-based access control model (GTRBAC). *IEEE Transaction on Knowledge and Data Engineering 17*,1 (Jan. 2005).

Kumar, A., Karnik, N., & Chafle, G. (2002, July). Context sensitivity in role-based access control, ACM SIGOPS Operating Systems Review, 36(3),.

Nyanchama, M., & Osborn, S. (1999, February). The Role Graph Model and Conflict Of Interest, ACM Transactions on Information and System Security, 2(1), 3-33.

Oh, S., Sandhu, R., & Zhang, X. (2006, May). An Effective Role Administration Model Using Organization Structure. *ACM Transactions on Information and System Security, 9*(2), 113-137.

Park, J. S., Costello, K. P., Neven, T. M., & Diosomito, J. A. (2004). A Composite RBAC Approach for Large, Complex Organizations. *Proceedings of Ninth ACM Symposium on Access Control Models*, June 2004, Yorktown Heights, New York, USA.

Perwaiz, N., & Sommerville, I. (2001). Structured Management of Role-Permission Relationships. *Proceedings of Sixth ACM Symposium on Access Control Models*, May 2001, Fairfax, Virginia, USA.

RTI International. (2002). The Economic Impact of Role-Based Access Control. March 2002. Retrieved from http://www.nist.gov/director/progofc/report02-1.pdf

Sandhu, R., Bhamidipati, V., & Munawer, Q. (1999, February). The ARBAC97 Model for Role-Based Administration of Roles, *ACM Transactions on Information and Systems Security, 2*.

Sandhu, R., Coyne, E., Feinstein, H., & Youman, C. (1996, February). Role-Based Access Control Models. *IEEE Computer, 29*(2).

Sandhu, R., Ferraiolo, D., & Kuhn, R. (2000). The NIST Model for Role-Based Access Control: Towards A Unified Standard. National Institute of Standards and Technology, December 2000, http://csrc.nist.gov/rbac/sandhu-ferraiolo-kuhn-00.pdf

Schaad, A., Moffett, J., & Jacob, J. (2001). The Role-Based Access Control System of a European Bank: A Case Study and Discussion. *Proceedings of Sixth ACM Symposium on Access Control Models*, May 2001, Fairfax, Virginia, USA.

Thomas, R. K. (1997). Team-Based Access Control (TMAC): A Primitive for Applying Role-Based Access Controls in Collaborative Environments. *Proceedings of the Second ACM workshop on Role-based Access Control*, Fairfax, VA, USA, 1997.

Zhang, Z., Zhang, X., & Sandhu, R. (2006). ROBAC: Scalable Role and Organization Based Access Control Models. *Proceedings of CollaborateCom-2006/TrustCol-2006*, Atlanta, Georgia, USA, November 2006.

## KEY TERMS

**Administrative RBAC:** Refers to approaches of controlling administrative tasks in RBAC. It usually provides ways to control the following major administrative tasks: assigning users to roles (user to role assignment), assigning permissions to roles (permission to role assignment), and adjusting role hierarchy (role to role assignment).

**Administrative ROBAC:** Refers to approaches of controlling administrative tasks in ROBAC. It usually provides ways to control the following major administrative tasks: assigning users to role-organization pairs, assigning permissions to roles, managing roles and role hierarchy, managing organizations and organization hierarchy, and managing role and organization association.

**Application Compartment (ACom):** An *ACom* of an application is a subset of a ROBAC where only the users, permissions, roles, and organizations applicable to the application are included.

**Permissible Administrative Organization Set (PAOSET):** A *paoset* of an organization is a set of organizations in which the greatest administrative role (gar) in the organization can modify the organization hierarchy.

**Permissible Administrative Role Set (PARSET):** A *parset* of an administrative role is a set of regular roles in which the administrative role can modify the regular role hierarchy.

**Permission Prerequisite Condition (PPC):** A condition a permission needs to be met before the permission can be assigned to roles.

**Role-Based Access Control (RBAC):** A method to restrict user's access to system resources based on the user's roles. In RBAC, roles are defined based on job functions, permissions are associated with roles, and users are made members of appropriate roles, thereby acquiring the roles' permissions.

**Role and Organization Based Access Control (ROBAC):** An extension of RBAC. In ROBAC, access is based on user's roles and the indirect association between users and system resources via organizations.

**User Prerequisite Condition (UPC):** A condition a user needs to be met before the user can be assigned to roles or role-organization pairs.