Role-based Administration of User-role Assignment and Its Oracle Implementation

Lilong Han, Qingtan Liu, and Zongkai Yang
Department of Information and Technology&Engineer Research Center on Education Infromation Technology,Huazhong Normal University,Wuhan,China
Hanlilong2001@yahoo.com.cn
WWW home page: http://eitec.ccnu.edu.cn/

Abstract. In role-based access control (RBAC) permissions are associated with roles, and users are made members of appropriate roles thereby acquiring the roles' permissions. The principal motivation behind RBAC is to simplify administration. An appealing possibility is to use RBAC itself to manage RBAC, to further provide administrative convenience. In this paper we investigate one aspect of RBAC administration concerning assignment of users to roles. We define a role-based administrative model, called URA (User-Role Assignment), for this purpose and describe its implementation in the Oracle database management system. Although our model is quite different from that built into Oracle, we demonstrate how to use Oracle stored procedures to implement it.

1 Introduction

Role-based access control (RBAC) has recently received considerable attention as a promising alternative to traditional discretionary and mandatory access controls [1-3]. In RBAC permissions are associated with roles, and users are made members of appropriate roles thereby acquiring the roles' permissions. This greatly simplifies management of permissions. Roles are created for the various job functions in an organization and users are assigned roles based on their responsibilities and qualifications. Users can be easily reassigned from one role to another. Roles can be granted new permissions as new applications and systems are incorporated, and permissions can be revoked from roles as needed. Role–role relationships can be established to lay out broad policy objectives.

Figure 1 illustrates the most general model in this family. In Fig.1 a single headed arrow indicates a one to one relationship and a double headed arrow indicates a many to many relationship. For simplicity we overload the term RBAC to refer to the family of models as well as its most general member.

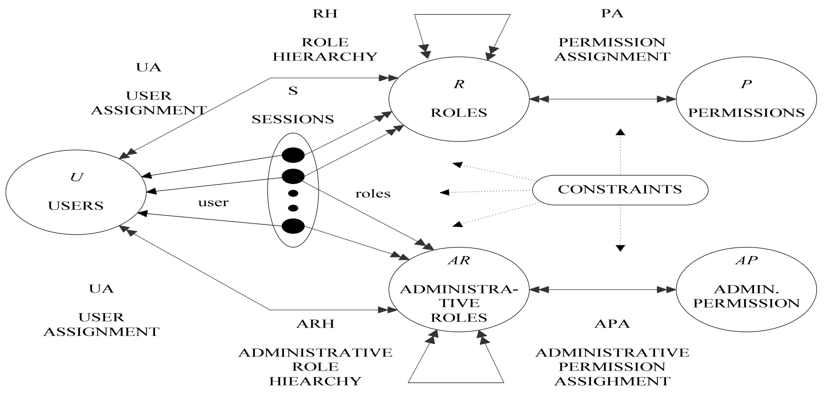

Fig. 1 RBAC model

In this paper we propose a model for the assignment of users to roles by means of administrative roles and permissions. For ease of reference we call this model as URA. URA imposes strict limits on individual administrators regarding which users can be assigned to which roles. We then describe an implementation of URA in the Oracle database management system [4-5]. Oracle's administrative model for user-role assignment is very different from URA. Nevertheless, we show how to use Oracle's stored procedures to implement URA.

2 URA Administrative Model

Administration of RBAC can be partitioned into several areas for which administrative models can be separately and independently developed to be later integrated. Our focus is exclusively on user-role assignment. As discussed in Section 1 this is likely to the first and most widely decentralized administrative task in RBAC.

2.1 URA Grant Model

In several systems, including Oracle, it is possible to designate a role, say, junior security officer (JSO) whose members have administrative control over one or more regular roles, say, A, B and C. Thus limited administrative authority is delegated to the JSO role. Unfortunately these systems typically allow the JSO role to have complete control over roles A, B and C. A member of JSO can not only add users to A, B and C but also delete users from these roles and add and delete permissions. Moreover, there is no control on which users can be added to the A, B and C roles by

JSO members. Finally, JSO members are allowed to assign A, B and C as junior to any role in the existing hierarchy (so long as this does not introduce a cycle). All this is consistent with classical discretionary thinking whereby member of JSO are effectively designated as "owners" of the A, B and C roles, and therefore are free to do whatever they want to these roles.

In URA our goal is to impose restrictions on which users can be added to a role by whom, as well as to clearly separate the ability to add and remove users from other operations on the role [6]. The notion of a prerequisite condition is a key part of URA.

Definition 1. A prerequisite condition is a boolean expression using the usual $\wedge$ and $\vee$ operators on terms of the form x and $\bar{x}$ where x is a regular role (i.e., $x \in R$). A prerequisite condition is evaluated for a user u by interpreting x to be true if $(\exists x' \geq x)\ (u, x') \in UA$ and $\bar{x}$ to be true if $(\forall x' \geq x)\ (u, x') \notin UA$. For a given set of roles R let CR denotes all possible prerequisite conditions that can be formed using the roles in R.

In the trivial case a prerequisite condition can be a tautology which is always true. The simplest non-trivial case of a prerequisite condition is test for membership in a single role, in which situation that single role is called a prerequisite role. User-role assignment is authorized in URA by the following relation.

Definition 2. The URA model controls user-role assignment by means of the relation *can-assign* $\subseteq AR \times CR \times 2^R$.

The meaning of *can-assign*(x, y, {a, b, c}) is that a member of the administrative role x (or a member of an administrative role that is senior to x) can assign a user whose current membership, or non-membership, in regular roles satisfies the prerequisite condition y to be a member of regular roles a, b or c.

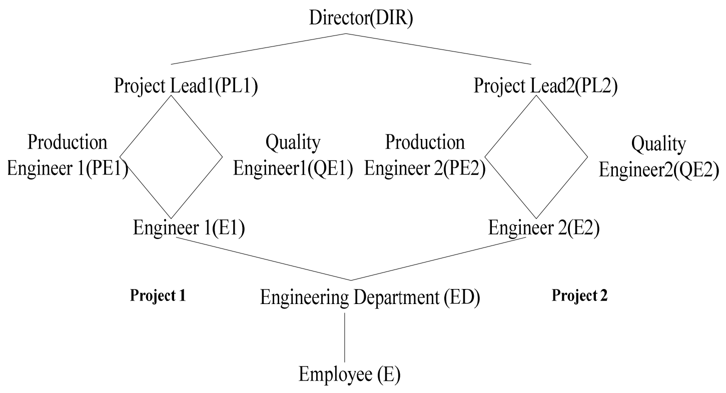

Fig. 2. An example role hierarchy

To appreciate the motivation behind the *can-assign* relation referring to the role hierarchy of Fig. 2 and the administrative role hierarchy of Fig. 3. Figure 2 shows the regular roles that exist in an engineering department. There is a junior-most role E to which all employees in the organization. Within the engineering department there is a junior-most role ED and senior-most role DIR. In between there are roles for two projects within the department, project 1 on the left and project 2 on the right. Each project has a senior-most project lead role (PL1 and PL2) and a junior-most engineer role (E1 and E2). In between each project has two incomparable roles, production engineer (PE1 and PE2) and quality engineer (QE1 and QE2).

Figure 2 suffices for our purpose but this structure can be extended to dozens and even hundreds of projects within the engineering department. Moreover, each project could have a different structure for its roles. The example can also be extended to multiple departments with different structure and policies applied to each department.

Figure 3 shows the administrative role hierarchy which co-exists with Fig. 2. The senior-most role is the senior security officer (SSO). Our main interest is in the administrative roles junior to SSO. These consist of two project security officer roles (PSO1 and PSO2) and a department security officer (DSO) role with the relationships illustrated in the figure.

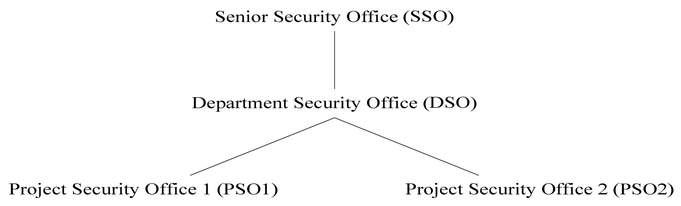
Senior Security Office (SSO)
Department Security Office (DSO)
Project Security Office 1 (PSO1)
Project Security Office 2 (PSO2)

Fig. 3. An example administrative role hierarchy

The role structure shown in Fig. 2 becomes a project oriented if the users are assigned to roles in a single project. If the users are assigned to roles from multiple projects then the structure is of matrix-form and if the users are assigned to same functional role in different projects then the structure is of functional oriented.

2.2 Prerequisite Roles of URA

For sake of illustration we define the *can-assign* relation shown in Table 1. This example has the simplest prerequisite condition of testing membership in a single role known as the prerequisite role [7].

The PSO1 role has partial responsibility over project 1 roles. Let Alice be a member of the PSO1 role and Bob a member of the ED role. Alice can assign Bob to any of the E1, PE1 and QE1 roles, but not to the PL1 role. Also if Charlie is not a member of the ED role, then Alice cannot assign him to any project 1 role. Hence, Alice has authority to enroll users in the E1, PE1 and QE1 roles provided these users are already members of ED. Note that if Alice assigns Bob to PE1 he does not need to be explicitly assigned to E1, since E1 permissions will be inherited via the role hierarchy. The PSO2 role is similar to PSO1 but with respect to project 2. The DSO role inherits the authority of PSO1 and PSO2 roles but can further add users who are members of ED to the PL1 and PL2 roles. The SSO role can add users who are in the E role to the ED role, as well as add users who are in the ED role to the DIR role.

This ensures that even the SSO must first enroll a user in the ED role before that user is enrolled in a role senior to ED. This is a reasonable specification for *can-assign*. There are, of course, lots of other equally reasonable specifications in this context. This is a matter of policy decision and our model provides the necessary flexibility.

In general, one would expect that the role being assigned is senior to the role previously required of the user. That is, if we have *can-assign* (a, b, C) then b is junior to all roles $c \in C$. We believe this will usually be the case, but we do not require it in the model. This allows URA to be applicable to situations where there is no role hierarchy or where such a constraint may not be appropriate.

The notation of Table 1 has benefited from the administrative role hierarchy. Thus for the DSO we have specified the role set as {PL1, PL2} and the other values are inherited from PSO1 and PSO2. Similarly for the SSO. Nevertheless explicit enumeration of the role set is unwieldy, particularly if we were to scale up to dozens or hundreds of projects in the department. Moreover, explicit enumeration is not resilient with respect to changes in the role hierarchy. Suppose a third project is introduced in the department, with roles E3, PE3, QE3, PL3 and PSO3 analogous to corresponding roles for projects 1 and 2.

Table 1. Example of *can-assign* with prerequisite roles

Administrative role	Prerequisite role	Role set
PSO1	ED	{E1,PE1,QE1}
PSO2	ED	{E2,PE2,QE2}
DSO	ED	{PL1,PL2}
SSO	E	{ED}
SSO	ED	{DIR}

3 Oracle RBAC and Related Features

The Oracle database management [4-5] system provides support for RBAC including support for hierarchical roles. However, Oracle does not directly support the URA model. In particular, Oracle has a strong discretionary flavor to its administrative model for user-role assignment. We will see in the next section how it is possible to use Oracle’s stored procedures to implement URA. In this section we briefly review relevant features of Oracle access control.

3.1 Privileges

Oracle has two kinds of privileges, system privileges and object privileges. System privileges authorize actions on a particular type of object for example create table, create user, etc. There are over 60 distinct system privileges. Object privileges authorize actions on a specific object (table, view, procedure, package, etc.). Typical examples of object privileges are select rows from a table, delete rows, execute procedures, etc.

Who can grant or revoke privileges from users or roles? The answer depends on various issues such as whether it is a system or an object privilege, and whether the object is owned by the user, etc. In order to grant or revoke a system privilege the user should have the admin option on that privilege or the user should have GRANT_ANY_PRIVILEGE system privilege. In order to grant or revoke an object privilege a user should own that particular object or the user should have grant option on the object if it is owned by someone else.

3.2 Roles in Oracle

Oracle provides roles (from Oracle 7.0 onwards) for ease of management of privilege assignment. System and object privileges can be granted to a role. A role can be granted to any other role (circular granting is not allowed). Any role can be granted to any user in the database. A role can either be enabled or disabled during a session. This includes both explicit and implicit roles that a user is a member of. Enabling a role will implicitly enable all the roles granted to it directly or transitively. The system privileges related to role management are CREATE_ROLE, GRANT_ANY_ROLE, DROP_ROLE, and DROP_ANY_ROLE. Information about privileges assigned to a role can be obtained from Oracle's built-in views ROLE_SYS_PRIVILEGES, ROLE_TAB_PRIVILEGES, and ROLE_ROLE_PRIVS. When a regular user performs query on these views these views only show information pertaining to the roles granted to that user. However, the Oracle internal user SYS will see information about all the roles through these views. The view SESSION_ROLES provides information about roles that are enabled in a session. The view ROLE_ROLE_PRIVS shows information about which roles are directly assigned to another role. Roles inherited transitively are not shown. For example, if role C was granted to role B and role B to role A the ROLE_ROLE_PRIVS view will show that B has been granted to A and C to B, but will not show the implied transitive C to A grant.

4 Implementing URA in Oracle

To implement URA we define Oracle relations which encode the *can-assign* URA. The *can-assign* relation of URA is implemented in Oracle as per the entity-relation diagram of Fig. 4. We assume that the prerequisite condition is converted into disjunctive normal form using standard techniques. Disjunctive normal form has the following structure [8-9].

$$(\cdots\wedge\cdots\wedge\cdots)\vee(\cdots\wedge\cdots\wedge\cdots)\vee\cdots\vee\cdots(\cdots\wedge\cdots\wedge\cdots)$$

Each $\cdots$ is a positive literal or a negated literal $\bar{x}$. Each group $(\cdots\wedge\cdots\wedge\cdots)$ is called a disjunction. For a given prerequisite condition *can-assign2* has a tuple for each disjunction. All positive literals of a single disjunction are in *can-assign3*, while negated literals are in *can-assign4.*

The *can-assign, can-assign2, can-assign3* and *can-assign4* relations are owned by the DBA who also decides what their content should be. In addition we have three accompanying procedures and a package to support these. Execute privilege on these procedures is given to all administrative roles. We achieve this by introducing a junior-most administrative role, say GSO (generic security officer), and assigning it the permission to execute these procedures.

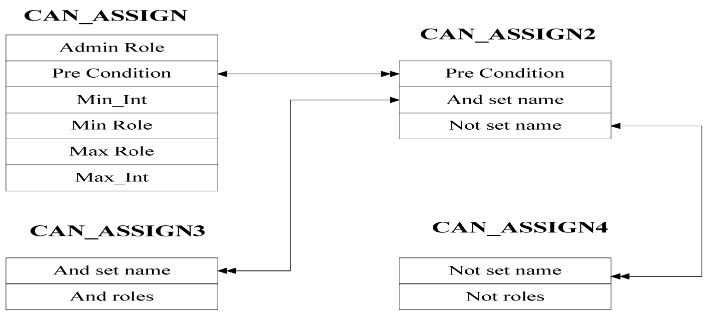

Fig. 4. Entity-relation diagram for *can-assign* relation

Oracle does not provide convenient primitives for testing whether or not a user is an implicit member of a particular role. Testing explicit membership is straightforward since explicit membership is encoded as a tuple in Oracle's system relations. To test implicit membership, however, we need to chase the role hierarchy.

In our implementation of URA a user invokes the stored procedure to grant a role from or to another user. The procedure calls are then as follows.

ASSIGN(user, trole, arole)

The parameters user and trole (target role) specify which user is to be added to trole. The arole parameter specifies which administrative role should be applied (with respect to the user who is invoking the URA procedure). The procedure code will check whether or not the user who calls the procedure has turned on the arole.

All the three procedures follow three basic steps.

1. If the user executing the procedure is an explicit or implicit member of arole then proceed to step 2, else stop execution and return an error message indicating this is not an authorized operation.

2. The tuple(s) from *can-assign* (for assign procedure) are obtained where AR role value equals or is junior to the arole parameter specified in the procedure call.

3. If trole is in the specified range for any one of the tuples selected in step 2, then assign the trole else return an appropriate error message. In case of ASSIGN also check whether the user being assigned to trole satisfies the prerequisite condition specified in the authorizing *can-assign* tuple or not.

Our implementation is convenient for the DBA since the stored procedures and packages we provide are generic and can be reused by other databases. The DBA only needs to define the roles and administrative roles, and configure the *can-assign* relations. Our implementation is available in the public domain for other researchers and practitioners to experiment with.

5 Conclusion

Authorization to assign to and from roles is controlled by administrative roles. The model requires users must previously satisfy a designated prerequisite condition before they can be enrolled via URA into additional roles. URA applies only to regular roles. Control of membership in administrative roles remains entirely in hands of the chief security officer.

The paper has also described an implementation of URA using Oracle stored procedures [10]. Oracle's built in primitives are cumbersome to use for determining indirect membership in roles. We have implemented suitable functions and packages to enable this conveniently. These should be of use to other researchers and practitioners and are available in the public domain.

Acknowledgement

This work was supported by doctor degree point in National Science Foundation under Grant No:20050511002 and supported by important project plan of education under Grant No:705038.

References

1. R.S. Sandhu, E.J. Coyne, H.L. Feinstein and C.E. Youman, Role-based access control models, IEEE Computer 29(2) (1996), 38–47.
2. S.H. von Solms and I. van der Merwe, The management of computer security profiles using a roleoriented approach, *Computers & Security* 13(8) (1994), 673–680.
3. C. Youman, E. Coyne and R. Sandhu, eds, *Proceedings of the 1st ACM Workshop on Role-Based Access Control,* Nov. 31–Dec. 1, 1995, ACM, 1997.
4. S. Feuerstein, Oracle PL/SQL Programming, O'Reilly & Associates, Inc., 1995.
5. G. Koch and K. Loney, *Oracle The Complete Reference*, Oracle Press, 1995.
6. L. Guiri and P. Iglio, A formal model for role-based access control with constraints, in: *Proceedings of IEEE* Computer Security Foundations Workshop 9, Kenmare, Ireland, June 1996, pp. 136–145.
7. R. Sandhu, Rationale for the RBAC96 family of access control models, in: *Proceedings of the 1st ACM Workshop on Role-Based Access Control, ACM,* 1997.
8. R.S. Sandhu, E.J. Coyne, H.L. Feinstein and C.E. Youman, Role-based access control models, IEEE Computer 29(2) (1996), 38–47.
9. C. Youman, E. Coyne and R. Sandhu, eds, *Proceedings of the 1st ACM Workshop on Role-Based Access Control, Nov*. 31–Dec. 1, 1995, ACM, 1997.
10. I. Mohammed and D.M. Dilts, Design for dynamic user-role-based security, Computers & Security 13(8) (1994), 661–671.