XML for RBAC Administration in Enterprise Environment

Cungang Yang and C. N. Zhang

Department of Computer Science

University of Regina, TRLabs

Regina, Saskatchewan, S4S 0A2

Phone: 1-306-5693716
cungang@cs.uregina.ca

Abstract: We have proposed an object-oriented RBAC (ORBAC) model to efficiently represent the real world. Though ORBAC is a good model, administration of ORBAC including creating and maintaining an access control security policy still remains a challengeable problem. In this paper, we present a practical method that can be employed in an enterprise environment to manage security policies using eXtensible Markup Language (XML). Based on ORBAC security policy expressed in XML, a role assignment algorithm is presented, the computation complexity of the algorithm is \(O(N)\) where \(N\) is the number of position roles in user’s assigned position role scope.

1. Introduction

To protect the resources in an enterprise environment, it is necessary to secure the data through the user authentication and access control policy. Nowadays, there are three basic access control techniques: mandatory access control (MAC), discretionary access control (DAC) and role-based access control (RBAC) [1][2][3]. The central notion of RBAC is that users do not directly get access to enterprise objects; instead, access privileges of the objects are associated with roles, each user is assigned to one or multiple members of appropriate roles. As a result, an organization not only preserves access control policy appropriate to its characteristics consistently, but it also maintains access control relationships between users and objects independently.

In the last few years, the fundamentals of RBAC policies have been identified [1] and a number of RBAC models have been proposed to satisfy security requirements in different areas [2][3][4]. In [5][6][7], we have proposed a variation of RBAC model called Object Oriented Role-based Access Control model (ORBAC). We used object technology to model application-level users access control because (1) object oriented technol-
ogy has been widely used in analyses and in designs of large and complex applications in which access control and security management are significant and complicated components. (2) While different applications are diversified, their access control has some generalities. Object oriented technology is good to model these generalities in order to reduce the burden of repeated development.

In an enterprise environment, there are thousands of users and objects, managing these roles and users, and their interrelationships is a formidable task for security manager. Administrative RBAC (ARBAC) [8] is an alternative solution. Though ARBAC relaxes the complexity of administration, it cannot solve the fundamental problems of administration.

Realizing the need for an enterprise environment that is flexible and manageable, researchers have proposed several frameworks such as OMG Resource Authorization Decision (RAD) specification [9], CORBA Security Service [10], and Secure European System in a Multi-vendor Environment (SESAME) [11] as a solution. These proposed frameworks separate security logics from application logics. Other researches such as [12][13][14] provide notation, logics, and calculi for expressing and reasoning about security policies. But these works mainly concern modelling policies and enforcement mechanisms and put little emphasis on managing security policies.

In this paper, we propose a novel concept that facilitates managing security policies in an enterprise environment. Our research employs XML-based technology for the policies definition and representation. The advantages of using XML are that as a meta language, XML can well define ORBAC security policies and is able to extend and modify easily. XML can precisely and effectively represents desired security policies and offers an additional degree of flexibility. In addition, XML allows a security manager to build and administrate a security policy for the entire enterprise.

The rest of the paper is organized as follows. Section 2 introduces the object oriented role-based access control model (ORBAC) on an enterprise environment. Section 3 concerns about the building and administration of the XML-based ORBAC security policy. Moreover, a role assignment algorithm is introduced in section 3.2.3. Finally, section 4 concludes the paper.

2. Object Oriented Role-Based Access Control Model (ORBAC) on an Enterprise Environment

The proposed ORBAC model [5][6][7] intends to capture the essential features of RBAC and fully realizes the original RBAC functions. Roles are divided into two groups: position roles and task roles. A position role is a collection of tasks (task roles) performed by a certain position in an enterprise, such as sales manager, sales clerk, vice president of the sales department, etc. In ORBAC, each position role takes on different tasks according to its responsibility and authority. For instance, a position role such as sales manager may carry out tasks like “approve order” or “grant loan extension”. We define those tasks as task roles because each task has one or multiple privileges. On the other hand, each task can be assigned to one or multiple position roles. A task role may have many privileges, and the same privilege can be associated to different task roles. A user can be assigned a number of position roles, a position role also can be assigned to multiple users. The class diagram of ORBAC d-
scribed by Unified Modeling Language (UML) [15] is shown in Fig. 1 and the definition of each element of the diagram is shown as follows.

(1) Class User:
A user is a human being or an autonomous agent, a position role set is assigned to each user according to user’s responsibility and authority called assigned position role set.

Formally, class User is described as:
User ID: identifies the user.

Fig. 1. Class Diagram of the Object Oriented Role-based Access Control Model

Position roles: refer to every position role in the user’s assigned position role set.
Class user has several member functions such as assigning a value to User ID, adding, deleting or modifying position roles to the user.

(2) Class Privilege:
A privilege is an action that can be exercised on an object. A task role can have one or multiple privileges directly. The formal definition of class Privilege is described as:
Privilege ID: identifies the privilege.
Task Roles: refer to all task roles that have the privilege directly.
Class privilege has a member function that assigns a value to Privilege ID, adding, deleting or modifying task roles to the privilege.

(3) Class Position Role: In ORBAC model, a position role can have multiple task roles and assign to multiple users. Also, a position role hierarchy is introduced to reflect inheritance of authority and responsibility among the position roles and defined by: If \( r_i \rightarrow r_j \), then position role \( r_i \) inherits the task roles of position role \( r_j \). Position role \( r_i \) is called a direct parent position role of position role \( r_j \) and position role \( r_j \) is called a direct child position role of position role \( r_i \). Furthermore, the inheritance relationship is transitive, that is, if \( r_i \rightarrow r_j \) and \( r_j \rightarrow r_k \) then \( r_i \rightarrow r_k \). Formally, class Position Role is described as:
Position Role ID: identifies the position role.
Task Roles: refer to all task roles that the position role invokes directly.
Users: refer to every user of the position role.
Direct parent position roles: refer to all its direct parent position roles.
Direct child position roles: refer to all its direct child position roles.
Position Role Constraints: refer to all its position role constraints.
Class Position Role has several member functions such as assigning a value to Position Role ID, adding, deleting, modifying its direct parent position roles, direct child position roles or task roles.

(4) Class Position Role Constraint:
ORBAC assigns constraints to user-position role authorization. An example of the user assignment constraint is that a position role may have a limited number of members. We call these constraints as cardinality constraints. The concept of prerequisite roles is based on competency and appropriateness, whereby a user can be assigned to position role A only if the user is already a member of position role B. Moreover, there exist many other position role constraints besides the above three types of constraints such as time constraint which concerns about how long the position roles can be activated.
Formally, class Position Role Constraint is described as:
Constraint ID: identifies the constraint.
Position Role: refer to the position role that related to the constraint.
Parameter values: parameter values of the constraint.
Class Position Role Constraint has several constraint functions that deal with different position role constraints such as prerequisite roles, time constraint, cardinality constraint, etc. The general procedure for those major types of position role constraints has been described in [6][7] and class Position Role Constraint has multiple access functions such as assigning a value to Constraint ID or assigning parameter values to constraint functions.

(5) Class Task Role:
A task role can have one or more than one privileges and can be invoked by one or multiple position roles. Each task role is used to implement a specific task, such as initiating a payment or authorizing a payment by calling its specific task function. Formally, class Task Role is defined as:
Task Role ID: identifies the task role.
Position Roles: refer to the position roles that invoke the task role.
Privileges: refer to the privileges that the task role invokes in order to carry out the task.
Class task role has several member functions such as assigning a value to Task Role ID, adding, deleting, modifying privileges of the task role. Also, class task role has the task function that is used to implement for the specific task.

(6) Class UR (User-Role):
In ORBAC, users carry out tasks by obtaining task roles. UR deals with authorizing task roles to a user.
Formally, class UR is defined as:
User: identifies the user who applies for task roles.
Task Roles: refer to the task roles for which the user applies.
The main function of UR is to realize task role authorization by calling a role assignment algorithms that will be described in section 3.2.3.
3. Implementing ORBAC Security Policy Using XML Technology

3.1 XML-based ORBAC Domain Security Policy

Our research employs XML for syntactic representation of the security policies. The XML specification [16] is the work of the World Wide Web Consortium (W3C) Standard Generalized Markup Language (SGML) Working Group. Being a meta-language, it provides accessible notations and means to describe an ORBAC conceptual model. Each ORBAC component is represented by an XML element and each XML-based ORBAC security policy is comprised of the following three parts:

Part 1: Basic elements of the XML-based ORBAC security policy.

- A Privilege object is represented by:
  ```xml
  <!–Privilege set definition–>
  <PRIVILEGE ID=privilege-id></PRIVILEGE>
  ```
The above syntax defines a new XML tag of type PRIVILEGE with a required ID attribute value `privilege-id`.

- A Position Role object is represented by:
  ```xml
  <!–Position Role definition–>
  <POSITION ROLE ID=role-id></POSITION ROLE>
  ```
The above syntax defines a new XML tag of type ROLE with a required ID attribute value `role-id`.

- A Constraint object is represented by:
  ```xml
  <!–Constraint set definition–>
  <CONSTRAINT ID=constraint-id></CONSTRAINT>
  <PARAMETER VALUE=parameter-values></PARAMETER>
  ```
The above syntax defines a new XML tag of type CONSTRAINT with a required ID attribute value `constraint-id`. The parameter values for the constraint are defined by a PARAMETER tag with a required VALUE attribute value `parameter-values`. PARAMETER VALUE is different for different kinds of constraints.

- A Task Role object is represented by:
  ```xml
  <!–Task Role definition–>
  <TASK ROLE ID=task-role-id></TASK ROLE>
  ```
The above syntax defines a new XML tag of type TASK ROLE with a required ID attribute value `task-role-id`.

Part 2: the relationships between different elements.

Part 2 defines the relationships between different elements of the ORBAC model.

- Position role hierarchy is represented as a set of INHERITES elements, each of which associates a position role with its direct child position role. For instance, a position role hierarchy is represented by:
  ```xml
  <!–Role hierarchy definition–>
  <INHERITES FROM = ri TO rj></INHERITES>
  ```
The above syntax defines a new XML tag of type INHERITES with a required FROM (position role `ri`) and TO (position role `rj`) attribute values which indicate position role `ri` is a direct parent position role of position role `rj`.
• A privilege assignment assigns a set of privileges to a task role and it is represented as:

```xml
<PRIV-ASSIGN TASK ROLE = ti PRIVILEGE= pj,…… pk.></PRIV-ASSIGN>
```

The above syntax defines a new XML tag of type PRIV-ASSIGN with ROLE and PRIVILEGE attributes in which task role `ti` has privileges of `pj,……pk`.

• A task role assignment assigns a set of task roles to a position role and it is represented as:

```xml
<TASK ROLE-ASSIGN POSITION ROLE = ri TASK ROLE= pj,…… pk.></TASK ROLE-ASSIGN>
```

The above syntax defines a new XML tag of type TASK-ASSIGN with POSITION ROLE and TASK ROLE attributes in which position role `ri` has task roles of `pj,……pk`.

• A constraint assignment assigns a set of constraints to a role and it is represented by:

```xml
<CONS-ASSIGN ROLE= r1 CONSTRAINTS= c1,……cm. ></CONS-ASSIGN>
```

The above syntax defines a new XML tag of type CONS-ASSIGN with a required ROLE and CONSTRAINTS attributes in which role `r1` has constraints of `c1,……cm`.

Part 3: Separation of duty constraints.

Part 3 defines an important theme of ORBAC: separation of duty or conflict of interest constraints. The goal of separation of duty is to improve security by requiring collision among two or more users. Some task roles cannot be assigned to same user, otherwise, one user would be sufficient to perform some illegal activities. A most well known example is the separate task roles which task functions are initiating a payment and authorizing a payment, no single position role which assigned to a user should be capable of executing both payments task roles. If a task role `ri` is defined to have a separation of duty constraint with task role `rj`, we call them as conflicted task roles. Any position role in a position role hierarchy cannot directly or indirectly inherit conflicted task roles. Therefore, when a position role hierarchy is created or modified, each position role should be checked.

Separation of duty constraint is represented by:

```xml
<SSOD-CONSTRAINT SOD= ri, rj ></SSOD-CONSTRAINT>
```

The above syntax defines a new XML tag of type SSOD-CONSTRAINT with a required SOD attributes in which task role `ri` and task `rj` are conflicted task roles.

Based on the definitions of each element of XML based ORBAC security policy, an example of security policy is shown as below:

Example 1:

```xml
<?xml version= "1.0" >
<ORBAC-MODEL TYPE= "RBAC1_POLICY">
```
<! — Basic Elements -- >
<! — Privilege set definition-- >
<PRIVILEGE ID= "p1" > </PRIVILEGE>
<PRIVILEGE ID= "p2" > </PRIVILEGE>
<PRIVILEGE ID= "p3" > </PRIVILEGE>
<PRIVILEGE ID= "p4" > </PRIVILEGE>
<PRIVILEGE ID= "p5" > </PRIVILEGE>
<PRIVILEGE ID= "p6" > </PRIVILEGE>
</ — Privilege set definition-- >
<! — Position Role definition-- >
<POSITION ROLE ID= " A "  > </POSITION ROLE>
<POSITION ROLE ID= " B "  > </POSITION ROLE>
<POSITION ROLE ID= " C "  > </POSITION ROLE>
<POSITION ROLE ID= " D "  > </POSITION ROLE>
<POSITION ROLE ID= " E "  > </POSITION ROLE>
<POSITION ROLE ID= " F "  > </POSITION ROLE>
<POSITION ROLE ID= " G "  > </POSITION ROLE>
<POSITION ROLE ID= " H "  > </POSITION ROLE>
</ — Position Role definition-- >
<! — Constraint set definition -- >
<CONSTRAINT ID= " C1 "  > </CONSTRAINT>
<PARAMETER VALUE= " h1 h2"></PARAMETER>
<CONSTRAINT ID= " C2 "  > </CONSTRAINT>
<PARAMETER VALUE= " h3"></PARAMETER>
<CONSTRAINT ID= " C3 "  > </CONSTRAINT>
<PARAMETER VALUE= " h4 h5"></PARAMETER>
<CONSTRAINT ID= " C4 "  > </CONSTRAINT>
<PARAMETER VALUE= " h6"></PARAMETER>
<CONSTRAINT ID= " C5 "  > </CONSTRAINT>
<PARAMETER VALUE= " h7 h8"></PARAMETER>
<CONSTRAINT ID= " C6 "  > </CONSTRAINT>
<PARAMETER VALUE= " h9 h10"></PARAMETER>
<CONSTRAINT ID= " C7 "  > </CONSTRAINT>
<PARAMETER VALUE= " h11"></PARAMETER>
<CONSTRAINT ID= " C8 "  > </CONSTRAINT>
<PARAMETER VALUE= " h12"></PARAMETER>
<CONSTRAINT ID= " C9 "  > </CONSTRAINT>
<PARAMETER VALUE= " h13"></PARAMETER>
</ — Constraint set definition -- >
<! — Task Role definition-- >
<TASK ROLE ID= " t1 " > </TASK ROLE>
<TASK ROLE ID= " t2 " > </TASK ROLE>
<TASK ROLE ID= " t3 " > </TASK ROLE>
<TASK ROLE ID= " t4 " > </TASK ROLE>
<TASK ROLE ID= " t5 " > </TASK ROLE>
<TASK ROLE ID=“t6”/>
</TASK ROLE>

<!-- Task Role definition-- >
</-- Basic Elements -- >

<!-- Relationships of Elements -->

<!-- Role hierarchy definition-- >

<INHERITES FROM = “A” To “B” > </INHERITES>
<INHERITES FROM = “A” To “C” > </INHERITES>
<INHERITES FROM = “B” To “E” > </INHERITES>
<INHERITES FROM = “B” To “F” > </INHERITES>
<INHERITES FROM = “C” To “F” > </INHERITES>
<INHERITES FROM = “C” To “G” > </INHERITES>
<INHERITES FROM = “D” To “G” > </INHERITES>
<INHERITES FROM = “D” To “H” > </INHERITES>

</-- Role hierarchy definition-- >

<! -- Privilege assignment definition -- >

<PRIV-ASSIGN TASK ROLE= “t1” PRIVILEGE= “p1” > </PRIV-ASSIGN>
<PRIV-ASSIGN TASK ROLE= “t2” PRIVILEGE= “p2 p4” > </PRIV-ASSIGN>
<PRIV-ASSIGN TASK ROLE= “t3” PRIVILEGE= “p3” > </PRIV-ASSIGN>
<PRIV-ASSIGN TASK ROLE= “t4” PRIVILEGE= “p4” > </PRIV-ASSIGN>
<PRIV-ASSIGN TASK ROLE= “t5” PRIVILEGE= “p4” > </PRIV-ASSIGN>
<PRIV-ASSIGN TASK ROLE= “t6” PRIVILEGE= “p5 p6” > </PRIV-ASSIGN>

</-- Privilege assignment definition-- >

<! -- Constraint assignment definition -- >

<CONS-ASSIGN ROLE= “A” CONSTRAINTS= “C9” > </CONS-ASSIGN>
<CONS-ASSIGN ROLE= “B” CONSTRAINTS= “C8” > </CONS-ASSIGN>
<CONS-ASSIGN ROLE= “C” CONSTRAINTS= “C6” > </CONS-ASSIGN>
<CONS-ASSIGN ROLE= “D” CONSTRAINTS= “C5” > </CONS-ASSIGN>
<CONS-ASSIGN ROLE= “E” CONSTRAINTS= “C1 C4” > </CONS-ASSIGN>
<CONS-ASSIGN ROLE= “F” CONSTRAINTS= “C7” > </CONS-ASSIGN>
<CONS-ASSIGN ROLE= “G” CONSTRAINTS= “C2” > </CONS-ASSIGN>
<CONS-ASSIGN ROLE= “H” CONSTRAINTS= “C3” > </CONS-ASSIGN>
3.2 Administration of the XML-based ORBAC Security Policy

3.2.1 Creating the Object Model of an XML-based ORBAC Security Policy

The object model of an XML-based ORBAC security policy is automatically created by using an object model translator. An example of the created object model of the XML-based ORBAC security policy in example 1 is shown in Fig. 2 and the general procedure for the creation of the object model presented in Fig.3 is described as follows.

**Step 1:** The object model translator parses each line in part 1 of the security policy in example 1 and creates objects of each element (privilege, position role, task role and position role constraint) according to their class definitions described in section 2, and calls function of the privilege objects to assign the value of `privilege-id` fetched from the security policy to each created privilege object. In the same way, assigns `position-role-id` to each position role object. Also, it assigns `constraint-id` to each position role constraint object and assigns `parameter-values` fetched from the security policy to the position role constraint object. For instance, the value of `constraint-id` for position role constraint object C6 is “C6” and the value of `parameter-values` of the object is “h9 h10”.

**Step 2:** The object model translator parses each line in part 2 of the security policy in example 1, establishes the relationships between position roles and position roles, position roles and task roles, task roles and privileges, position roles and position role constraints. Finally, an object model of the sample 1 XML-based ORBAC domain security policy is created and shown in Fig.2.
Step 3: The object model translator parses each line in part 3 of the security policy in example 1, a separation of duty constraint table which lists all pairs of conflicted task roles of example 1 is created.

Step 4: Based on the separation of duty constraints table, the object model translator checks each position role of the created object model and make sure that each of them is

![Diagram](https://via.placeholder.com/150)

**Fig. 2** An example object model of XML-based ORBAC security policy in example 1
secure. We call a position role in an object model is secure if it does not inherit conflicted task roles. For instance, in the object model of sample 1 XML-based ORBAC security policy, there is only one pair of conflicted task roles, t1 and t6, exists in the separation of duty constraints table. Apparently, each position role in Fig. 2 is secure.

**Step 5:** Each position role in an object model is secure cannot guarantee every assigned position role set is also secure. For instance, though each position role in the object model of Fig. 2 is secure, an assigned position role set like {B, D} is not secure, we call an assigned position role set is secure if position role of the set does not inherit conflicted task roles. Therefore, the object model translator should check and guarantee each assigned position role set is secure and then create an assigned position role set table which lists pairs of user and his/her assigned position role set.

Based on the created object model of XML-based security policy shown in Fig. 2 and the assigned position role set table, a user-task role table and a user-assigned position role scope table are created and will be used for the role assignment algorithm in section 3.2.3. **User-task roles table** lists pairs of user and his/her assigned task roles. **Position role scope table** lists pairs of user and his/her assigned position role scope. Position role scope of a user is comprised of the position roles of his/her assigned position role set and all child position roles of each element of the set. For instance, in Fig. 2, if the assigned position role set for user U1 is A, then the corresponding task roles of user U1 in the user-task roles table is {t1, t2, t3, t5} and the assigned position role scope corresponding with user U1 in the user-assigned position role scope table is {A, B, C, E, F, G} which is enclosed by a triangular dashed line shown in Fig. 2.

![Fig. 3. The diagram of creating an object model by security manager](image-url)
User-task roles table and user-assigned position roles scope table are created and maintained by security manager, if user’s assigned position role set table changed, then his/her use-task roles table and user-assigned position roles scope table will be automatically recreated by the object model translator.

3.2.2 Modifying the Object Model of XML-based ORBAC Security Policy
In an enterprise environment, the number of position roles, task roles, position role constraints and privileges can be very large. Maintaining the secure status of an security policy if basic elements of ORBAC, such as position roles, task roles, privileges and position role constraints are required to be added, modified or deleted time by time is a formidable task. Our method simplified the maintenance procedure by using the proposed XML-based ORBAC security policy. If the security manager manually adds, deletes or modifies basic elements in an XML-based security policy, then the object model translator automatically recreate the new object model and based on the separation of duty table to check and make sure that each position role after the modification, addition or deletion in the object model is still secure. Moreover, user-assigned task roles table and user-assigned position roles scope table will be recreated.

3.2.3 Role Assignment Algorithm
Based on the created object model of XML-based ORBAC security policy, security manager creates UR object, calls role assignment algorithm to search the object model and assign task roles to users. When a user wants to carry out some tasks, he/she can directly get the required task roles by applying for those task roles. We call them applied task roles.

The advantage of the algorithm 1 presented in this section is that it satisfies the principle of least privilege. The principle of least privilege has been described as important for meeting security objectives [12]. In ORBAC, user applies for task roles directly and UR calls algorithm 1 to assign an authorize task role set to the user.

Algorithm 1: Role Assignment Algorithm
Input : UserID : identification of the user
        Applied Task Roles S: the task roles the user applies for.
Output : Authorize task role set T to the user.
{ temporary table 2 []=0;
  For each task role A in the applied task roles S {
    if task role A ∈ task roles of the user’s user-task roles table {
      For each position role that invokes task role A, K {
        if position role K ∈ the user’s assigned position roles scope
          insert position role K to temporary table 2
      }
      For each position role F in temporary table 2, checks its constraints {
        if (all constraints of position role F is satisfied) {
          goto 3:
        }
      }
    }
  }
}
For each direct parent role of the position F, G {
    if (position role G ∈ the user’s assigned position roles scope)
        Insert position role G to the temporary table 2
}
}

return the refuse message for the application of task position role A to the user
3: Insert task role A to the authorized task role set T
}

The authorized task role set T will be returned to the user
}

In Fig. 2, suppose user U1’s assigned role set is \{A\} and U1 applies for task role t5. The search path for task role t5 is shown in Fig. 2. For each position role A, B, C or F that satisfies (1) each of them directly or indirectly invokes task role t5, (2) each of them is included in the assigned role scope of the user. If any position role A, B, C or F satisfy all its position role constraints, then task role t5 will be assigned to the user, otherwise, an error message will be returned to the user.

Generally, a user may choose multiple task roles such as task roles t5, t3 and t1 in a session, in the worst condition, algorithm 1 will search all the position roles of the user’s authorized role scope, thus if we assume the number of position roles in the authorized role scope for the user is N, the computation complexity of algorithm 1 will be \(O(N)\).

4. Conclusion:

In this paper, we presented an XML-based security policy. The driving motivation of it is to simplify security policy administration on object-oriented RBAC systems. The main contribution of this paper is that we have presented a new approach to managing XML-based security policy by security manager. Unlike most existing implementations, with our approach the authorization is independently defined and is separated from policy representation and from implementation mechanisms. We believe that our concept can be applied to develop a generalized security policy language for expressing any security policy for an enterprise environment. Moreover, in the paper, role assignment algorithm is also presented.

Reference: