Cardinality Constraint Access Control Model and Implementation

Zhikun Zhang, Youping Geng, Jianguo Xiao, Zhonghui Huang

Advances on Information Sciences and Service Sciences. Volume 3, Number 1, February 2011

Abstract

Analysis constraints and its diversity in security access control model. Cardinality is a common constraint in access control model. An extended cardinality constraint access control model is established. According to actual demand, we describe the user-role, user-session, role-session cardinality constraints based on the RBAC model. The differences between static and dynamic cardinality constraints are also represented particularly. Describe the extended model with Schema based XML and extend corresponding model functional specification. In order to support the extended model, we give an implementation structure including access control management and query functions.

Keywords: Access Control, Cardinality Constraint, RBAC

1. Introduction

Along with the computer application system to improve the complexity and access control mode has been continuously developing. For has been widely accepted role-based access control mode (RBAC) [1] of extensive research and is mainly in the form of additional constraints. Constraint is model of mandatory rules must be followed [2]. In access control model with additional constraints can express various access control policy. The ANSI RBAC has contains the separation of duty constraints [3]. In order to express specific safety strategy, we can add other internal or external conditions into the access control model.

Cardinality Constraint is a common constraint. But it is not involved in the ANSI RBAC. The existing work mainly focused on the role cardinality constraints without explanation the static and dynamic meaning. The other elements, such as the permissions and objects are not described clearly. The relevant session cardinality constraint is also not involved.

According to the actual demand of the system we analyze the existing cardinality constraints in current access control model, make the potential of expansion, and construct some functions supporting the extended model.

2. Classification and diversity of constraints

Constraint is a group of conditions. When a user is trying to execute an operation he must satisfy these conditions. If the conditions are satisfied, the operation is allowed, otherwise operation are prohibited. In the operating process, we determine and evaluate system parameters, then get some true or false Boolean values. These parameters may be related to the subject and the object of the operation may also be not in them. It only depends on the operation of the external environment. Constraints in access control mode can be devided into different classification criterias.

According to the object constraint can be divided into role constraints, user constraint, permission constraint, and session constraint, etc. The separation of duty constraint is a typical role constraint.

According to the time and location constraint can be divided for static and dynamic constraint. Static constraints work in the security policy management, and dynamic constraints work in the access control decision-making. For example, the active role count constraint in a session is a dynamic constrain and has not static form. Generally speaking, the static constraint is strict and dynamic constraint is flexible.
3. Cardinality Constraint in access control model

3.1. Access control model

RBAC\(^{[1]}\) defines some basic components: USERS, ROLES, PRMS, OPS, OBS and SES, represent users, roles, permissions, operations, objects and sessions. Assignment is implemented in two ways: user-role assignment and permission-role assignment. User activate roles through sessions.

**Definition 1.** RBAC components:
- User set USERS\(=\{u_1,u_2,\ldots,u_n\}\)
- Role set ROLES\(=\{r_1,r_2,\ldots,r_n\}\)
- Operation set OPS\(=\{op_1,op_2,\ldots,op_n\}\)
- Object set OBS\(=\{ob_1,ob_2,\ldots,ob_n\}\)
- Permission set PRMS\(=\{p_1,p_2,\ldots,p_n\}\)
- Session set SES\(=\{s_1,s_2,\ldots,s_n\}\)

**Definition 2.** RBAC functions:
- assignedUsers\(: r \rightarrow 2^{USERS}\), map role \(r\) to a user set, returns all users assigned to role \(r\).
- assignedRoles\(: u \rightarrow 2^{ROLES}\), map user \(u\) to a role set, returns all roles assigned to user \(u\).
- userSessions\(: u \rightarrow 2^{'SES}\), map user \(u\) to a session set, returns all sessions user \(u\) creates.
- sessionUser\(: s \rightarrow USERS\), map session \(s\) to user \(u\), returns the user \(u\) creates the session \(s\).
- roleSessions\(: r \rightarrow 2^{'SES}\), map role \(r\) to a session set, returns all sessions activate role \(r\).
- sessionRoles\(: s \rightarrow 2^{ROLES}\), map session \(s\) to a role set, returns all roles activated in session \(s\).

3.2. Cardinality constraints on relationship of basic components

Cardinality is the elements number in a set. The restriction of element number in a set is cardinality constraint. In practical applications always need to restrict the user number of a role, such as the department leader role can only be assigned to one user. Cardinality constraints are widespread in the various elements and processes in access control model. Table 1 lists a part of cardinality constraints on access control model.

<table>
<thead>
<tr>
<th>Constraint Instance</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>USERS),ROLES,PRMS,OPS,OBS,SES</td>
</tr>
<tr>
<td>assignedUsers(r)</td>
<td>The number of users assigned to the role (r).</td>
</tr>
<tr>
<td>assignedRoles(u)</td>
<td>The number of roles assigned to the user (u).</td>
</tr>
<tr>
<td>userSessions(u)</td>
<td>The number of sessions owned by user (u).</td>
</tr>
<tr>
<td>roleSessions(r)</td>
<td>The number of sessions activate role (r).</td>
</tr>
<tr>
<td>sessionRoles(s)</td>
<td>The number of roles in session (s).</td>
</tr>
</tbody>
</table>

The cardinality constraints exist in corresponding components of access control model. In addition, there are cardinality constraints base on the session content. The existing work mainly focused on the role cardinality constraints. In fact the cardinality constraints are varied, and have static and dynamic features. The following is detailed description of the two kinds of cardinality constraints.

**Definition 3 :** The number of elements in a basic component set \(component \subseteq COMPONENT\), is called the component cardinality constraint, and written as \(component\_CARD\). For example, user set cardinality constraint is written as \(USER\_CARD\). Its mean is \(|USERS|\leq USER\_CARD\).
Definition 4: Given a basic component set $dom$ and $ran$, $dom \in COMPONENT$, $ran \in COMPONENT$, $ran \neq dom$, $F$ is the mapping relationship from $dom$ to $ran$, $e \in dom$. The $e$ cardinality constraint on $ran$ is the maximum of \{ $e$ \} on $F$, and written as $dorm_{\text{ran}}$ _CARD. For example, a user $u \subseteq USERS$, the $u$ cardinality constraint on ROLES, is the maximum of \{ $u$ \} on $UA$, and written as $USER_{\text{ROLE}}$ _CARD. It means $|\text{assignedRoles}(u)|<USER_{\text{ROLE}}$ _CARD. Such constraints referred to as “User-Role cardinality constraint” collectively.

The static cardinality constraint on USERS is the total number limit of users. Correspondingly, the dynamic cardinality constraint is the number limit of the users can access a running system. We summarize all kinds of cardinality constraints on model elements and their relations in table 2.

<table>
<thead>
<tr>
<th>Table 2. Cardinality constraints on elements and relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERS</td>
</tr>
<tr>
<td>USER CARD</td>
</tr>
<tr>
<td>ROLES</td>
</tr>
<tr>
<td>PERMS</td>
</tr>
<tr>
<td>OPS</td>
</tr>
<tr>
<td>OBS</td>
</tr>
</tbody>
</table>

In order to have a convenient description, we define some functions will be used in cardinality constraint at first:

3.3. Cardinality constraints on sessions

For a basic model element $e \subseteq element$, function $elementSessions(e)$ returns the sessions set of including element $e$.

Definition 5: The session cardinality constraint of $e$ is the maximum of the set returned by $elementSessions(e)$, and is written as $element_{\text{SESSION}}$ _CARD. For example, the user $u$ cardinality constraint is the maximum of sessions of $u$ can be created, and is written as $USER_{\text{SESSION}}$ _CARD. It means $|\text{userSessions}(u)|<USER_{\text{SESSION}}$ _CARD. Such constraints collectively referred to as “User-Session cardinality constraint”. Table 3 summarizes the session cardinality constraints. Obviously, these constraints are dynamic.

<table>
<thead>
<tr>
<th>Table 3. Session Cardinality constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint Name</td>
</tr>
<tr>
<td>USERS</td>
</tr>
<tr>
<td>ROLES</td>
</tr>
<tr>
<td>PERMS</td>
</tr>
<tr>
<td>OPS</td>
</tr>
<tr>
<td>OBS</td>
</tr>
</tbody>
</table>

We also need to be aware that some cardinality constraint is not common. An application system actual demand of cardinality constraint may only be a fraction of the numerous constraints. In a system contains all the above constraints completely is not necessary.

4. Expended cardinality constraint

The following we analyze and describe the expanded cardinality constraints based on ANSI RBAC model.

4.1. Role-User cardinality constraint (RUCC)

Static $RUCC$ $(SRUCC)$ means the maximum of users in a role at any time. Dynamic $RUCC$ $(DRUCC)$ means the maximum of users in a role within all sessions. We define the two concept as
following:

\[ SRUCC: \forall (r,n) \in SRUCC, \forall r \in ROLES \Rightarrow |assignedUsers(r)| < n. \]  
\[ SRUCC \subseteq (ROLES \times N) \] is the collection of static role-user cardinality constraints. Every element is a dual-relation \((r,n)\), \(r \in ROLES\), \(n = STATIC\_ROLE\_USER\_CARD\).

\[ DRUCC: \forall (r,n) \in DRUCC, \forall s \in SESSIONS, \forall r \in ROLES \Rightarrow |\bigcup s \in RoleSessions(r) sessionUser(s)| < n. \]  
\[ DRUCC \subseteq (ROLES \times N) \] is the collection of dynamic role-user cardinality constraints. Every element is a dual-relation \((r,n)\), \(r \in ROLES\), \(n = DYNAMIC\_ROLE\_USER\_CARD\), \(s \in SESSIONS\).

4.2. User session cardinality constraint (USCC)

\[ USCC \] is to restrict the maximum of sessions a user can create in a system. \(\forall (u,n) \in USCC \Rightarrow |userSessions(u)| < n. \] \[ USCC \subseteq (USERS \times N) \] is the collection of cardinality constraints of sessions a user can own. \((u,n)\) is an element, \(u \in USERS\), \(n = USER\_SESSION\_CARD_u\).

4.3. Role session cardinality constraint (RSCC)

\[ RSCC \] is to restrict the maximum of sessions a role can create in a system. \(\forall (r,n) \in RSCC \Rightarrow |roleSessions(r)| < n. \] \[ RSCC \subseteq (ROLES \times N) \] is the collection of cardinality constraints of sessions a user can own. \((r,n)\) is an element, \(r \in ROLES\), \(n = ROLE\_SESSION\_CARD_r\).

5. Extended model specification based on XML Schema

![XML instance of cardinality constraint on model elements](image)

Figure 1. XML instance of cardinality constraint on model elements
In traditional application, security policy description always adopts special format. It is not convenient to policy share and exchange. XML provides a unified platform-independent data representation standard, which greatly facilitates information sharing and cross-platform data distribution. So it becomes the first choice of security information exchanging in distributed system.

XML Schema [9] is a language to describe XML documents structure. It not only can express rich data types, but also itself is an XML document. XML Schema is appropriate adapting to more extensive access control policies and the needs of various extensions.

The XML instance of cardinality constraint in access control model is shown in figure 1. Constraint \textit{admin\_role\_user\_card} defines static and dynamic role-user cardinality as 3 and 2 respectively. The role with such cardinality constraint will be assigned two users at most and can only be activated by one user at the same time.

Similarly, figure 2 shows an XML instance of cardinality constraints associate with sessions. Constraint \textit{single\_user\_session\_card} limits a user can only create one effective session to access the system. Constraint \textit{admin\_role\_session\_card} limits a role can only own two sessions at most.

\begin{verbatim}
<SESSION_CARDINALITIES>
  <USER_SESSION_CARDINALITYs>
    <CARDINALITY id="single_user_session_card" name="session cardinality for student user">1</CARDINALITY>
    <CARDINALITY id="admin_role_session_card" name="session cardinality for administrator role">3</CARDINALITY>
  </USER_SESSION_CARDINALITYs>
  <ROLE_SESSION_CARDINALITYs>
    <CARDINALITY id="single_user_session_card" name="session cardinality for student user">1</CARDINALITY>
    <CARDINALITY id="admin_role_session_card" name="session cardinality for administrator role">3</CARDINALITY>
  </ROLE_SESSION_CARDINALITYs>
</SESSION_CARDINALITIES>
</xs:schema>
\end{verbatim}

Figure 2. XML instance of user-session and role-session cardinality constraints
6. Implementation of access control system

An access control system includes independent two parts: policy management and authorization decision. Policy management is responsible for basic model elements, relations and cardinality constraints. According to the requests the authorization decision module extracts necessary safety policies, including user, role, permission, $UA$, $PA$, and cardinality constraint, etc. At the same time, it picks up necessary resources and environment information, gives a decision and returns authorization result.

6.1. Policy management

Policy management system defines security policy and ensures the integrity and consistency of the policy. The security policy is always stored as relational data. It also can be transferred between relational data and XML document. But its integrity and consistency could be destructed after the transformation. Integrity is the contents express by the policy are complete and all the referenced contents are existent. When the integrity is destroyed, there will be null references. Such as the role assigned to a user does not exist will cause the user cannot use the system. Consistency means there are not conflict policy is defined in the policy library. Destruction of consistency is also composed of violation of constraints. For example, SRUCC has limited a user can only own one role. But the user may have more than one role when the SRUCC is defining. The integrity and consistency module work since the system initialization to ensure the initial policy is correct.

In general, if the policy exchange which may cause damage of integrity and consistency does not occur frequently, in order to improve system efficiency, the integrity and consistency validation process is optional in the system.

6.2. Session management

Session management is the main part of authorization decision. The authorization decision process is composed of three steps: receive the request, make decision and return the true or false result. Each step is constructed base on the session management. The session management contains session creating, renewing and revoking. The data structure of a session is shown in table 4 and the algorithm to create a session is shown in figure 3:
Cardinality Constraint Access Control Model and Implementation  
Zhikun Zhang, Youping Geng, Jianguo Xiao, Zhonghui Huang  
Advances on Information Sciences and Service Sciences. Volume 3, Number 1, February 2011

CreateSession
INPUT user, roles active_roles, string session  
OUTPUT boolean
Check if u is a valid user
Check if session does not exist
Check if all roles in active_roles are assigned to u
Check DSD consistency
/system session cardinality constraint
IF SESSIONS[u] = SystemSessionCardinality() THEN
RETURN false
END IF
//RUC
us_c = UserSessionCardinality(u)
IF UserSessions[u] + 1 = us_c THEN
RETURN false
END IF
//RSCC
FOR EACH r in active_roles
IF RoleSessionCardinality(r) THEN
RETURN false
END FOR
//DRUCC
FOR EACH r with role-user dynamic cardinality in active_roles
IF RoleUserCardinality(r, false) = false-dynamic active_u = [r]
FOR EACH session in SESSIONS
IF r in SessionRoles(session) THEN active_u = active_u U SessionUser(session)
END FOR
IF active_u = dru_c THEN RETURN false
END FOR
//DRUCC
dur_c = UserRoleCardinality(u, false) = false-dynamic
IF dur_c > 0 THEN
active_r = active_roles
FOR EACH session in SESSIONS
IF u = SessionUser(session) THEN active_r = active_r U SessionRoles(session)
IF active_r = dur_c THEN RETURN false
END FOR
END IF
Establish the session structure for new session
RETURN true

Figure 3. Algorithm of function CreateSession()

<table>
<thead>
<tr>
<th>Data name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session_id</td>
<td>String</td>
<td>Session’ id</td>
</tr>
<tr>
<td>User_id</td>
<td>String</td>
<td>User’s id who create the session</td>
</tr>
<tr>
<td>User_name</td>
<td>String</td>
<td>User’ name who create the session</td>
</tr>
<tr>
<td>Active_roles</td>
<td>ArrayList</td>
<td>Role set activated by the session</td>
</tr>
<tr>
<td>Login_id</td>
<td>String</td>
<td>Login’s id</td>
</tr>
<tr>
<td>Login_time</td>
<td>Datetime</td>
<td>Session’s created time</td>
</tr>
<tr>
<td>Expire_time</td>
<td>Datetime</td>
<td>Session’s expire time</td>
</tr>
</tbody>
</table>

6.3. Expanded model functions supporting cardinality constraint

Model function is one of the RBAC model standard components. The system implementation needs the supporting of model functions. The RBAC model has defined three types of model functions: administration functions, system support functions and query functions. Administration functions
create and delete basic model elements and their mapping relations, including the users, roles, user-role assignments and permission-role assignment. System support functions support system sessions, check permissions and activate roles. The system support function is the key part of access control implementation. Its main content is session management. Review functions can be divided into basic and advance review functions. They are summarized in Table 5.

<table>
<thead>
<tr>
<th>Administration functions</th>
<th>System support functions</th>
<th>Basic review function</th>
<th>Advance review functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddUser</td>
<td>CreateSession</td>
<td>AssignedUsers</td>
<td>RolePermissions</td>
</tr>
<tr>
<td>DeleteUser</td>
<td>DeleteSession</td>
<td>AssignedRoles</td>
<td>UserPermissions</td>
</tr>
<tr>
<td>AddRole</td>
<td>AddActiveRole</td>
<td></td>
<td>SessionRoles</td>
</tr>
<tr>
<td>DeleteRole</td>
<td>DropActiveRole</td>
<td></td>
<td>SessionPermissions</td>
</tr>
<tr>
<td>AssignUser</td>
<td>CheckAccess</td>
<td></td>
<td>RoleOperationsOnObject</td>
</tr>
<tr>
<td>DeassignUser</td>
<td></td>
<td></td>
<td>UserOperationsOnObject</td>
</tr>
<tr>
<td>GrantPermission</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RevokePermission</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5. RBAC functions

The expanded model is added cardinality constraint. So the model functions need to be expanded too. It includes two part contents: Add new model function to support new cardinality constraint, and redefine existing model function part, including setting and cancel the cardinality constraint functions, query and verify functions (Table 6).

<table>
<thead>
<tr>
<th>New administration functions</th>
<th>New query functions</th>
<th>Redefined functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetRoleUserCardinality</td>
<td>RoleUserCardinality</td>
<td>AssignUser</td>
</tr>
<tr>
<td>SetUserRoleCardinality</td>
<td>UserRoleCardinality</td>
<td>CreateSession</td>
</tr>
<tr>
<td>SetUserSessionCardinality</td>
<td>UserSessionCardinality</td>
<td>AddActiveRole</td>
</tr>
<tr>
<td>SetRoleSessionCardinality</td>
<td>SystemSessionCardinality</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SessionUser</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UserSessions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RoleSessions</td>
<td></td>
</tr>
</tbody>
</table>

Table 6. New Functions supporting cardinality constraint

SetRoleUserCardinality() and SetUserRoleCardinality() can set and cancel the dynamic and static cardinality constraint. Using object-oriented method, the realization of the two new functions only needs us to add a new method SetCardinality() to the corresponding model elements.

7. Summary and future work

Based on the ANSI model RBAC, we analyze expounded classification and diversity of cardinality constraints. According to the actual demand describe some cardinality constraints, expand and establish a extended access control model. Create and modify the constraints required administration functions and query the functions.

With deepening of the research and practice of the security access control model, we need to expand more constraints. As for time in actual application precondition, characters, environmental attribute (such as location), context and role levels may come to be our further research directions.

8. References


