Description Logic Based Conflict Detection Methods for RB-RBAC Model

Haibo Yu, Qi Xie and Haiyan Che

College of Computer Science and Technology, Jilin University, Changchun 130012, China

Summary
RB-RBAC (Rule-Based RBAC) provides the mechanism to dynamically assign users to roles based on a finite set of authorization rules defined by the enterprise's security policy. The RB-RBAC family introduces negative authorization, represented by negative roles, which may bring conflict, and conflict detection and resolution become an import work in RB-RBAC policy management. We proposed a formalization of RB-RBAC model by description logic and developed conflict detection methods based on description logic reasoning service. Conflicts can be detected when all authorization rules have been defined, and a revised detection method is also given to improve the system efficiency when dynamically adding new authorization rule to system. Conflicts among related rules and among unrelated rules can be distinguished by these methods. We also demonstrate a simple method to resolve conflict.

Key words:
RB-RBAC, Description Logic, Policy conflict, Conflict detection

1. Introduction

Role-Based Access Control (RBAC) has emerged as a widely deployed alternative to traditional discretionary and mandatory access controls [1],[2]. Usually, enterprise security officer manually assign users to roles based on criteria specified by the enterprise. But in many environments, the number of users can be in the hundreds of thousands or millions. This renders manual user-to-role assignment a formidable task. Rule-Based RBAC (RB-RBAC) [3],[4],[5] is introduced to automatically assign users to roles based on a finite set of authorization rules defined by enterprise. RB-RBAC is an excellent authorization model especially for distribution environments with a large number of users.

The RB-RBAC family introduces negative authorization, represented by negative roles, to the RBAC world [5]. Introducing negative authorization may lead to conflict, and conflict detection and resolution become an import work in RB-RBAC policy management. In [5] only analysis about conflict and some resolution are discussed. Some logic methods [6],[7],[8] were proposed, most of them did not have efficient implementations. In [9],[10],[11] policy based system was build. Most of these works do not support complex attribute expression definition, quasi-order relation definition among attribute values and RB-RBAC seniority level reasoning.

We propose a description logic based approach to deal with components in RB-RBAC. Description logic (DLs) [12] is a family of languages used to describe and classify concepts and their instances. Compared with first-order logic, DLs achieve a better tradeoff between computational complexity of reasoning and the expressiveness of the language. In this approach, attribute expression should be represent in a manner that makes seniority level reasoning become a simple work. Comparison between attribute expressions is less restricted to allow insight on the relations of authorization rules even they are not identical syntax structures. Most important, the detecting methods are efficient enough for implementation. We also demonstrate a simple method to rewrite conflicted rules for eliminating conflict.

The paper is organized as follow. In section 2, we give an overview of RB-RBAC model. In section 3, we introduce description language $\mathcal{ALC}$ [12]. In section 4, we represent the RB-RBAC model in $\mathcal{ALC}$. In section 5, we discuss our conflict detection methods. Section 6 concludes the paper.

2. RB-RBAC Model

The main components of the RB-RBAC model are users, attribute expressions, roles and permissions. The component users, roles and permissions are imported from RBAC96 [1].

In RB-RBAC, the security policies of the enterprise are expressed in form of a set of authorization rules. Each rule takes as an input the attributes expression that is satisfied by a user and produces one or more roles. Every attribute expression actually defines a specific user set. The following is an example of a rule: $ae \Rightarrow r$, where $ae$ is attribute expression and $r$ is the produced role. If user $u$ satisfies $ae$, then $u$ is authorized to the role(s) in the right hand side of rule. RB-RBAC family allow negative authorization such as following $ae \Rightarrow \neg r$. The rule above
states that once a user satisfies $ae_i$ system that implements RB-RBAC will prohibit that user from assuming $r_i$.

To capture the seniority relations that might exist among authorization rules, the dominance binary relation on attribute expressions is introduced: $ae_i$ is said to dominate $ae_j$ only if $ae_i$ implicates $ae_j$ logically. That indicates each counterpart attribute value of attribute expressions also exists seniority levels. Another way of stating the above relation between $ae_i$ and $ae_j$ is to say that $rule_i$ is senior to $rule_j$ (denoted by $\geq$): $\text{rule}_i \geq \text{rule}_j \iff (ae_i \rightarrow ae_j)$.

This implies that users who satisfy $\text{rule}_j$ also satisfy $\text{rule}_i$ and, hence, are authorized to the roles produced by $\text{rule}_i$. Introducing "\$\neg\$" to the right hand side may lead to conflict in the state of a single user wrt a single role. The conflict is due to simultaneous positive and negative authorizations. Figure 1 describes a set of authorization rules. Conflict among unrelated rules likes the one among $\text{rule}_2$ and $\text{rule}_3$. If a user satisfies $\text{rule}_1$ and $\text{rule}_2$ simultaneously then $\text{rule}_3$ should be authorized to activate $r_3$, but at the same time, authorized to assume $r_1$, because $\text{rule}_2 \geq \text{rule}_1$.

3. The Description Logic for Modeling RB-RBAC

We choose a DL language AAX[12] to represent and reason on RB-RBAC according to features of RB-RBAC. In DLs, the vocabulary consists of concepts, which denote sets of individuals, and roles, which denote binary relationships between individuals.

Elementary descriptions are atomic concepts (denoted by $A$) and atomic roles (denoted by $R$) and complex ones can be built from them inductively with concept constructors. AAX concepts (denoted by $C,D$) are formed inductively according to the following syntax rules:

$$C,D \rightarrow A \mid \neg C \mid C \circ D \mid \exists R.C$$.

- is defined as universal concept, and $\bot$ is defined as bottom concept, such that $\bot = \neg \top$. We can also define the constructors: $C \Delta D = \neg (C \circ D)$, $\forall R.C = \neg (\exists R.\neg C)$, $C \circ D = (C \circ D) \neg (C \circ D)$, and so on.

In order to define a formal semantics of AAX-concepts, we consider interpretations $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, that consist of a domain of the interpretation $\Delta^\mathcal{I}$ and an interpretation function $\cdot^\mathcal{I}$, which assigns to every atomic concept $A$ a set $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$ and to every atomic role $R$ a binary relation $R^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$. The syntax and semantics of AAX is summarized in Table 1.

<table>
<thead>
<tr>
<th>Constructors</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>universal concept</td>
<td>$\Delta$</td>
<td>$\mathcal{A}$</td>
</tr>
<tr>
<td>atomic concept</td>
<td>$A$</td>
<td>$A^\mathcal{I}$</td>
</tr>
<tr>
<td>concept negation</td>
<td>$\neg C$</td>
<td>$\Delta^\mathcal{I} \setminus C^\mathcal{I}$</td>
</tr>
<tr>
<td>intersection</td>
<td>$C \circ D$</td>
<td>$C^\mathcal{I} \cap D^\mathcal{I}$</td>
</tr>
<tr>
<td>existential restriction</td>
<td>$\exists R.C$</td>
<td>${a \in \Delta^\mathcal{I} \mid \exists b.(a,b) \in R^\mathcal{I} \land b \in \Delta^\mathcal{I}}$</td>
</tr>
</tbody>
</table>

A knowledge base (KB) comprises two components, the TBox and the ABox.

TBox (denoted as $\mathcal{T}$ ) is a finite set of terminological axioms. Generally, they have two forms:

$$C \circ D (R \in S) \text{ or } C = D (R = S),$$

where $C, D$ are concepts (and $R, S$ are roles). Axioms of the first kind are called inclusions, while axioms of the second kind are called equalities. To simplify the exposition, we deal in the following only with axioms involving concepts. An interpretation $\mathcal{I}$ satisfies an inclusion $C \circ D$ if $C^\mathcal{I} \subseteq D^\mathcal{I}$, and it satisfies an equality $C = D$ if $C^\mathcal{I} = D^\mathcal{I}$. If $\mathcal{I}$ satisfies an axiom (resp. a set of axioms), then we say that it is a model of this axiom (resp. set of axioms). Two axioms or two sets of axioms are equivalent if they have the same models.

ABox (denoted as $\mathcal{A}$) is a finite set of individual assertions: $\mathcal{A}(a)$ or $R(b,c)$, where $C$ is a concept, $R$ is a role, $a$, $b$ and $c$ are individuals. By the first kind, called concept assertions, one states that $a$ belongs to (the interpretation of) $C$, by the second kind, called role assertions, one states that $c$ is a filler of the role $R$ for $b$.

Typical reasoning tasks for a terminology are satisfiability and subsumption. A concept $C$ is satisfiable with respect to $\mathcal{T}$ if there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^\mathcal{I}$ is nonempty. A concept $C$ is subsumed by a concept $D$ with respect to $\mathcal{T}$ if $C^\mathcal{I} \subseteq D^\mathcal{I}$ for every model $\mathcal{I}$ of $\mathcal{T}$, denoted as $\mathcal{T} \models C \circ D$. In AAX, subsumption can be reduced to satisfiability as follow: $C$ is subsumed by $D$ if and only if $D \models \neg C \iff \neg \exists R.C$ is unsatisfiable.

4. Representing and Reasoning on RB-RBAC

Fig.1. Example of conflict
Given a RB-RBAC system, we define a DL knowledge base K and assume that users, roles, attributes and permissions are finite.

The vocabulary of K includes the following atomic concepts and atomic roles:

(i) The atomic concepts User, Role and Permission, represent the users, roles and permissions,

(ii) For each role ri in system, one atomic concept Rolei,

(iii) For each permission pi in system, one atomic concept Permissioni,

(iv) For each attribute expression, one atomic concept AEi,

(v) For each attribute Ai, one atomic concept CAi, and for each attribute value of attribute Ai, one atomic concept CAvali,

(vi) For each attribute Ai, one atomic role hasAi, represents the user hold attribute value of attribute Ai,

(vii) The atomic role assignRole, indicate the user can be assigned the role automatically,

(viii) The atomic role holdPermission, represent the user can hold the permission.

The TBox of K includes five catalogs of axioms:

*Attribute inclusion axioms* state the seniority levels among attribute values. For each seniority relation: vj is senior to vi, we should setup axioms with the form CAvalj \( \subseteq \) CAvali. Moreover, each concept CAvali is a subconcept of CAi, so axioms CAvalj \( \subseteq \) CAi should be included for each attribute value.

For example, in a department of a company, there are two positions: department manager (dm) and project manager (pm) and a dm also acts as a pm. First, we define atomic concepts CPosition, DM and PM, and an atomic role hasPosition. Then, we set up axioms DM \( \subseteq \) CPosition, PM \( \subseteq \) CPosition and DM \( \subseteq \) PM in TBox. Concept \( \exists \)hasPosition,DM is interpreted as users whose position is department manager.

*Role inclusion axioms* declare the role hierarchies. Axiom Rolei \( \subseteq \) Rolej should be included for each role hierarchy: role \( r_j \) inherits permissions of \( r_i \). Each concept Rolei is also a subconcept of CRole, we should setup axioms Rolei \( \subseteq \) CRole for each role.

*Attribute expression definition axioms* define the attribute expressions and specify the concrete attribute values which users should hold. For each authorization rule rulei, definition axioms have the general form:

\[ AEi = \exists A, C, \text{CAval} \quad \text{if} \quad \exists A, \text{CAval} \quad \text{and} \quad \text{CAval} \quad \text{are} \quad \text{correct} \].

If some kinds of attributes do not exist in an attribute expression, they should disappear in the definition axioms and need not be donated as \( \exists A, \text{CAval} \). If an attribute expression requires more than one values about some kinds of attributes, they should be defined as such form: \( \exists A, \text{CAval} \). More complex conditions can be defined using other constructors.

*Role assignment axioms* express roles are assigned automatically to users who satisfy attribute expressions of authorization rules. For each authorization rule rulei, role assignment axioms have the general form:

\[ AEi \equiv \exists \text{assignRole}(Rolei, \text{Role}i) \].

Where \( \text{Role}i \), \( \text{Role}i \), \( \text{Role}im \) are roles produced by rulei. These axioms indicate if a user satisfies the attribute expression of an authorization rule then it will be assigned roles produced by that rule. For example, in figure 1, authorization rule rule2 can be represented as

\[ AEi \equiv \exists \text{assignRole}(\text{Role}i, \text{Role}i) \].

Of course, we can set up such axiom as \( AEi \equiv \exists \text{assignRole}(\text{Role}i) \), which represents users are prohibited to assume the corresponding role \( r_i \). These axioms indicate if a user satisfies the attribute expression of an authorization rule then it will be assigned roles produced by that rule. For example, in figure 1, authorization rule rule2 can be represented as

\[ AEi \equiv \exists \text{assignRole}(\text{Role}i, \text{Role}i) \].

Concept \( \exists \text{assignRole}(\text{Role}i) \) is interpreted as the set of users that can be authorized the permissions \( p_i \). Concept \( \exists \text{assignRole}(\text{Role}i, \text{Role}i) \) is interpreted as the set of users that are automatically assigned to role. This axiom indicates that if a user has been automatically assigned to the role \( r_i \) then this user can be authorized the permission \( p_i \).

The ABox of K includes five catalogs of assertions:

(i) User concept assertions have the form CUser(u) and introduce the users.

(ii) Role concept assertions have the form Role(r) and declare that each role belongs to corresponding role concept.

(iii) Attribute value concept assertions have the form CAval(v) and declare that each attribute value belongs to corresponding attribute value concept.

(iv) Permission concept assertions have the form Permission(p) and declare that each permission belongs to corresponding permission concept.

(v) User attribute assertions have the form hasA(u, v) and indicate that user u holds attribute value v of attribute A.

Now, we can use the reasoning services provided by DL to achieve some reasoning tasks and make access control decision. A so-called "Tell&Ask" interface specifies operations that enable knowledge base construction (Tell operations) and operations that allow one to get information out of the knowledge base (Ask operations).

We can query if a user u is automatically assigned to role \( r_i \) using such statement: \( \text{Ask}(\exists \text{assignRole}(\text{Role}i, u), \text{user} (u)) \).
which checks if \( u \) is an instance of \( \exists \text{assignRole}.\text{Role}_r \), or whether \( u \) is automatically assigned role \( r \). We can ask knowledge base to query whether a user \( u \) is authorized permission \( p \), by \( \text{Ask}(\exists \text{holdpermission}._r(p))(u) \). The statement \( \text{Ask}(\exists \text{holdpermission}._r(p))(u) \) can test whether a user \( u \) satisfies the corresponding attribute expression \( ae \).

According to role assignment axioms and authorization axioms, we can conclude that if a user \( u \) satisfies an attribute expression which is automatically assigned roles holding permission \( p \), then user \( u \) is authorized permission \( p \).

In [3], authorization rules as well as attributes expressions that have identical syntax structures can be compared to determine seniority levels among them. That is too restricted to detect the insight about relationships among rules. We remove this restriction for comparisons and determine relations among rules only based on comparison of user sets specified by attribute expressions on the left hand sides of authorization rules.

Besides reasoning on access control decision, TBox inference can help us to determine dominance relation between attribute expressions. For arbitrary attribute expression concepts \( AE_i \) and \( AE_j \), if there is \( T \models AE_i \circ AE_j \), which indicates each user satisfies \( ae_i \) also satisfies \( ae_j \), then we can say \( ae_j \) dominates \( ae_i \). If \( T \models AE_i \circ AE_j \) and \( T \not\models AE_j \circ AE_i \), then we can conclude that there is no seniority relation between \( ae_i \) and \( ae_j \).

### 5. Conflict Detection

When we add each of attribute expression definition axioms to TBox, we must check whether that atomic concept is satisfiable by calling TBox coherence check. That will preclude TBox from accepting incorrect attribute expression definition. For example, in a department of a company, there are two positions: department manager (DM) and project manager (PM). A department manager also acts as a project manager. Then we define an attribute expression concept \( \text{AEmis} \) as form

\[
\text{AEmis} = \exists \text{hasPosition}.(\text{DM} \circ \neg \text{PM}),
\]

which specifies a set of user who is a department manager but not a project manager. Because each department manager is also a project manager, the attribute inclusion axiom \( \text{DM} \circ \text{PM} \) is included in TBox. Consequently, the concept \( \text{AEmis} \) is unsatisfiable with respect to TBox.

In RB-RBAC, conflict among related rules and conflict among unrelated rules are main conflict types. Conflict among related rules arises from the following situations: if there are seniority relations between two authorization rules \( \text{rule}_i \) and \( \text{rule}_j \), i.e. \( \text{rule}_i \geq \text{rule}_j \) or \( \text{æe}_i \) dominating \( \text{æe}_j \), and there is a role \( r \) which appears in the two sets of roles produced by these rules respectively with form \( r \) and \( \neg r \).

Conflict among unrelated rules arises from the following situations: if there are same users who satisfy both authorization rules \( \text{rule}_i \) and \( \text{rule}_j \), and there a role \( r \) which appears in the two sets of roles produced by these rules respectively with form \( r \) and \( \neg r \).

In some sense, conflict among related rules are specifics of conflict among unrelated rules, and users satisfying senior rules is just the users satisfying both rules. So, we can have a conflict detection method to detect all these conflict at the same time. This conflict detection method can deal with a set of authorization rules simultaneously and we recommend that role assignment axioms should be kept in a separate location with other axioms when implementing this method. The method is as follows.

Firstly, all axioms except role assignment axioms are loaded to TBox and TBox should be still coherent. For arbitrary attribute expression definition axioms \( AE_i \) and \( AE_j \) in TBox, we check whether concept \( AE_i \circ AE_j \) is satisfiable with respect to TBox. For each satisfiable concept \( AE_i \circ AE_j \), concept pair \( (AE_i, AE_j) \) is added to the set \( \text{OverlappedAEs} \).

Secondly, all role assignment axioms are added to TBox after attribute expression definition axioms. For each concept pair \( (AE_i, AE_j) \) in \( \text{OverlappedAEs} \), we check again whether concept \( AE_i \circ AE_j \) is satisfiable with respect to current TBox. If concept \( AE_i \circ AE_j \) is unsatisfiable, i.e. \( T \not\models AE_i \circ AE_j \), then there must exist some kind conflict between \( \text{rule}_i \) and \( \text{rule}_j \).

Thirdly, if we want to distinguish the different conflict types, we need remove all role assignment axioms of conflicted authorization rules from TBox. For each conflict between \( \text{rule}_i \) and \( \text{rule}_j \), detected in previous step, if concept subsumption between \( AE_i \) and \( AE_j \) is satisfiable, i.e. \( T \models AE_i \circ AE_j \) or \( T \models AE_j \circ AE_i \), then the conflict arises among related rules, else it arises among unrelated rules.

Considering system efficiency, the above conflict detection method will be not a good choice when adding a new authorization rule to a system without any conflict. Because it will calculate all attribute expression pairs of that system even though they have been detected and revised to eliminate any conflict. Hence, we tailor it to the situations when adding a new authorization rule to a set of authorization rules without any conflict.

Firstly, we add an attribute expression definition axiom \( AE_i \) to TBox and TBox should be still coherent. For each attribute expression definition axiom \( AE_j \) already included in TBox, we check whether concept \( AE_i \circ AE_j \) is satisfiable with respect to TBox. For each
satisfiable concept \(AE_i \cap AE_j\), concept pair \((AE_i, AE_j)\) is added to the set OverlappedAEs.

Secondly, we add the role assignment axiom about \(AE_i\) to TBox. For each concept pair \((AE_i, AE_j) \in OverlappedAEs\), we check again whether concept \(AE_i \cap AE_j\) is satisfiable with respect to current TBox. If concept \(AE_i \cap AE_j\) is unsatisfiable, i.e. \(T \not\models AE_i \cap AE_j\), then there must exist some kind conflict between \(rule_i\) and \(rule_j\).

Thirdly, if we want to distinguish the different conflict types, we need remove the role assignment axiom about \(AE_i\) from TBox. For each conflict between \(rule_i\) and \(rule_j\), detected in previous step, if concept subsumption between \(AE_i\) and \(AE_j\) such as \(T \models AE_i \supset AE_j\) or \(T \models AE_j \supset AE_i\) is satisfiable, then the conflict arises among related rules, else it arises among unrelated rules.

The security officers can choose appropriate method according different detecting tasks.

Conflicts detected can be resolved automatically by rewriting algorithms or manually by security officers. We just give a simple resolution method to demonstrate how to rewrite axioms to resolve conflict among related rules and unrelated rules respectively, although we can resolve conflicts needlessly to knowing their types.

For conflict among related rules, we give a simple conflict resolution method based on Denial Takes Precedence (DTP) [5], and after rewriting new rules should have same semantics with old ones. First, we remove conflict role assignment axioms. Second, we define new attribute concepts: \(AE_i^\prime \equiv \neg AE_i \cap AE_j\), \(AE_j^\prime \equiv \neg AE_i \cap AE_j\), and \(AE_{ij} \equiv \neg AE_i \cap AE_j\). For the example in figure 1, we give rewritten axioms in figure 2(a).

This resolution method is good choice for simple situation, but it may not ensure to add the least number of rules to TBox when more then one conflict exists at the same time. For example, for such rules \(rule_i \supset rule_j\) and \(rule_k \supset rule_j\), there are conflict between \(rule_i\) and \(rule_j\), and conflict also between \(rule_i\) and \(rule_k\). We add \(rule_i\) to TBox after \(rule_j\) and \(rule_k\) have been added. Above resolution method can eliminate conflict between \(rule_i\) and \(rule_j\), and conflict between \(rule_i\) and \(rule_k\). That adds four new rules which include two role assignment axioms both about \(AE_i\). From security officer's opinion, these two role assignment axioms should be merged to one and that will not tamper the original semantics.

Conflicts among unrelated rules also could be resolved by rewriting authorization rules, and after rewriting new rules should have same semantics with old ones. We give a simple conflict resolution method also based on DTP. First, we remove conflict role assignment axioms. Second, we define new attribute concepts: \(AE_i^\prime \equiv AE_i \cap \neg AE_j\), \(AE_j^\prime \equiv \neg AE_i \cap AE_j\), respectively represent users satisfying \(rule_i\) but not satisfying \(rule_j\), users satisfying \(rule_j\) but not satisfying \(rule_i\), and ones satisfying \(rule_i\) and \(rule_j\) simultaneously. Finally, we specify new role assignment axioms to \(AE_i^\prime\), \(AE_j^\prime\), and \(AE_{ij}\): \(AE_i^\prime\) will be assigned roles of \(rule_i\), \(AE_j^\prime\) will be assigned roles of \(rule_j\) and \(AE_{ij}\) will be assigned the union of roles of these two rules except all opposite role pairs. For the example in figure 1, we give rewritten axioms in figure 2(b). But this resolution method is suitable for only one conflict detected in TBox. If more then one conflict is detected, then this method can not get optimized resolution. More complex algorithms can be given to resolve this problem.

Fig. 2. Examples of rewriting

6. Conclusion

We have shown how to detect conflicts among authorization rules in RB-RBAC. A description logic based formalization also has been demonstrated to...
represent and reason on RB-RBAC model. Besides performing make authorization decision and basic DL reasoning task, we mainly proposed the approaches to detect conflicts among authorization rules. Conflicts can be detected when all authorization rules have been defined. A revised detection method is also given to improve the system efficiency in the process of dynamically adding new authorization rule to system. We also demonstrate simple methods to rewrite conflict rules for eliminating conflict.

A complex rewriting algorithm should be developed to optimize the resolution of more than one conflict. In order to express such properties as \( \text{age} \geq 18 \), concrete domain [12] should be considered to improve expressive power of the language we use.

References


Haibo Yu received the B.E. and M.E. degrees, from Jilin Univ. in 1997 and 2000, respectively. After working as a research assistant (from 2000), an instructor (from 2002) in the College of Computer Science and Tech., Jilin Univ.. His research interest includes network security, software engineering.