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 $\mathcal{ALX}$ [12]. In section 4, we represent the RB-RBAC model in $\mathcal{ALX}$. 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 $rule_i$: $ae_i \Rightarrow r_g$, where ae_i is attribute expression and r_g is the produced role. If user u satisfies ae_i, then u is authorized to the role(s) in the right hand side of $rule_i$. RB-RBAC family allow negative authorization such as following $ae_k \Rightarrow \neg r_j$. The rule above

Manuscript revised January 28, 2006.

states that once a user satisfies ae_k system that implements RB-RBAC will prohibit that user from assuming r_j.

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 $\geq$):

$$rule_i \geq rule_j \leftrightarrow (ae_i \rightarrow ae_j).$$

This implies that users who satisfy $rule_i$ also satisfy $rule_j$ and, hence, are authorized to the roles produced by $rule_j$.

Introducing "¬" 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 between $rule_1$ and $rule_2$. If u satisfies $rule_1$ and $rule_2$ simultaneously then u should be authorized to activate r_1 and denied r_1 at the same time. Conflict among related rules is as following. $rule_2$ and $rule_4$ are conflicting because if u satisfies $rule_2$ then he is denied r_1, but at the same time, authorized to assume r_1 because $rule_2 \geq rule_4$.

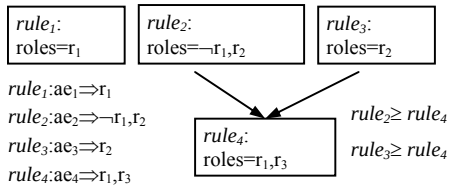

Fig.1. Example of conflict

3. The Description Logic for Modeling RB-RBAC

We choose a DL language AΛX[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. AΛX concepts (denoted by C,D) are formed inductively according to the following syntax rules:

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

$\bullet$ is defined as universal concept, and $\perp$ is defined as bottom concept, such that $\bullet = \neg\perp$. We also can define the constructors: $C \sqcup D = \neg(\neg C \sqcap \neg D)$, $\forall R.C = \neg(\exists R.\neg C)$, $C \; xor \; D = (C \sqcup D) \sqcap \neg(C \sqcap D)$, and so on.

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

Table 1: The syntax and semantics of AΛX

Constructors	Syntax	Semantics
universal concept	$\bullet$	Δ^I
atomic concept	A	A^I
concept negation	$\neg C$	$\Delta^I \setminus C^I$
intersection	$C \sqcap D$	$C^I \cap D^I$
existential restriction	$\exists R.C$	$\{a \in \Delta^I \mid \exists b.(a,b) \in R^I \wedge b \in C^I\}$

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:

$$C \sqsubseteq D \; (R \sqsubseteq S) \; or \; C \equiv D \; (R \equiv S),$$

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 I satisfies an inclusion $C \sqsubseteq D$ if $C^I \subseteq D^I$, and it satisfies an equality $C \equiv D$ if $C^I = D^I$. If 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 A) is a finite set of individual assertions: $C(a)$ 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 satisfiablity and subsumption. A concept C is *satisfiable* with respect to $\mathcal{T}$ if there exists a model I of $\mathcal{T}$ such that C^I is nonempty. A concept C is *subsumed* by a concept D with respect to $\mathcal{T}$ if $C^I \subseteq D^I$ for every model I of $\mathcal{T}$, denoted as $\mathcal{T} \; C \sqsubseteq D$. In AΛX, subsumption can be reduced to satisfiability as follow: C is *subsumed* by $D \Leftrightarrow C \sqcap \neg D$ is *unsatisfiable*.

4. Representing and Reasoning on RB-RBAC

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 ***CUser***, ***CRole*** and ***CPermission***, represent the users, roles and permissions,

(ii) For each role r_i in system, one atomic concept $\boldsymbol{Role_i}$,

(iii) For each permission p_i in system, one atomic concept $\boldsymbol{Permission_i}$,

(iv) For each attribute expression, one atomic concept $\boldsymbol{AE_i}$,

(v) For each attribute A_i, one atomic concept $\boldsymbol{CA_i}$, and for each attribute value of attribute A_i, one atomic concept $\boldsymbol{CAval_i^j}$,

(vi) For each attribute A_i, one atomic role $\boldsymbol{hasA_i}$, represents the user hold attribute value of attribute A_i,

(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: v_i^j is senior to v_i^k, we should setup axioms with the form $CAval_i^j \sqsubseteq CAval_i^k$. Moreover, each concept $\boldsymbol{CAval_i^j}$ is a subconcept of $\boldsymbol{CA_i}$, so axioms $CAval_i^j \sqsubseteq CA_i$ should be included for each attribute value.

For example, in a department of a company, there are two positions: department manger (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 $DM \sqsubseteq CPosition$, $PM \sqsubseteq CPosition$ and $DM \sqsubseteq PM$ in TBox. Concept $\exists hasPosition.DM$ is interpreted as users whose position is department manager.

Role inclusion axioms declare the role hierarchies. Axiom $Role_i \sqsubseteq Role_j$ should be included for each role hierarchy: role r_i inherits permissions of r_j. Each concept $\boldsymbol{Role_i}$ is also a subconcept of ***CRole***, we should set up axioms $Role_i \sqsubseteq CRole$ 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 $rule_i$, definition axioms have the general form:

$$AE_i \equiv \exists hasA_1.CAval_1^{j_1} \sqcap \ldots \sqcap \exists hasA_n.CAval_n^{j_n}.$$

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 $\exists hasA_i.\bullet$. If an attribute expression requires more then one values about some kinds of attributes, they should be defined as such form: $\exists hasA_i.(CAval_i^{k1} \sqcap \ldots \sqcap CAval_i^{km})$. 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 $rule_i$, role assignment axioms have the general form:

$$AE_i \sqsubseteq \exists assignRole.(Role_{k1} \sqcup \ldots \sqcup Role_{km}).$$

Where $Role_{k1}, \ldots, Role_{km}$ are roles produced by $rule_i$. 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 $rule_2$ can be represented as

$$AE_2 \sqsubseteq \forall assignRole.\neg Role_1 \sqcap \exists assignRole.Role_2.$$

Of course, we can set up such axiom as $AE_i \equiv \exists assignRole.(Role_1 \ xor \ Role_2)$, which represents users are prohibited to assume the corresponding role r_1 and r_2 at the same time.

Authorization axiom declares users can get permissions by automatically assigned roles. For each role-permission assignment ($role_i$, p_k), authorization axioms have the general form:

$$\exists assignRole.Role_i \sqsubseteq \exists holdPermission.P_k.$$

Concept $\exists holdPermission.P_k$ is interpreted as the set of users that can be authorized the permission p_k, and concept $\exists assignRole.Role_i$ is interpreted as the set of users that are automatically assigned to $role_i$. This axiom indicates that if a user has been automatically assigned to the $role_i$ then this user can be authorized the permission p_k.

The ABox of K includes five catalogs of assertions:

(i) *User concept assertions* have the form $\boldsymbol{CUser}(u)$ and introduce the users.

(ii) *Role concept assertions* have the form $\boldsymbol{Role_i}(r_i)$ and declare that each role belongs to corresponding role concept.

(iii) *Attribute value concept assertions* have the form $\boldsymbol{CAval_i^j}(v_i^j)$ and declare that each attribute value belongs to corresponding attribute value concept.

(iv) *Permission concept assertions* have the form $\boldsymbol{Permission_i}(p_i)$ and declare that each permission belongs to corresponding permission concept.

(v) *User attribute assertions* have the form $\boldsymbol{hasA_i}(u, v)$ and indicate that user u holds attribute value v of attribute A_i.

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: $\textbf{Ask}((\exists assignRole.Role_i)(u))$,

which checks if u is an instance of $\exists assignRole.Role_i$, or whether u is automatically assigned role r_i. We can ask knowledge base to query whether a user u is authorized permission p_i by $\mathbf{Ask}((\exists holdpermission.P_i)(u))$. The statement $\mathbf{Ask}(AE_i(u))$ can test whether a user u satisfies the corresponding attribute expression ae_i.

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 prevent 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 $\top \sqsubseteq AE_i \sqcap AE_j$, which indicates each user satisfies ae_i also satisfies ae_j, then we can say ae_i dominates ae_j or $rule_i \geq rule_j$. If $\top \sqsubseteq AE_i \sqcap AE_j$ and $\top \sqsubseteq AE_j \sqcap AE_i$, 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 manger (DM) and project manager (PM). A department manger also acts as a project manager. Then we define an attribute expression concept *AEmis* as form

$$AEmis \equiv \exists hasPosition.(DM \sqcap \neg PM),$$

which specifies a set of user who is a department manger but not a project manager. Because each department manager is also a project manager, the attribute inclusion axiom $DM \sqsubseteq PM$ is included in TBox. Consequently, the concept *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 $rule_i$ and $rule_j$, i.e. $rule_i \geq rule_j$ or ae_i dominating ae_j, and there is a role r which appears in the two sets of roles produced by these rules respectively with form r and form $\neg r$.

Conflict among unrelated rules arises from the following situations: if there are same users who satisfy both authorization rules $rule_i$ and $rule_j$, and there a role r which appears in the two sets of roles produced by these rules respectively with form r and form $\neg r$.

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 give 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 $AE_i \sqcap AE_j$ is satisfiable with respect to TBox. For each satisfiable concept $AE_i \sqcap AE_j$, concept pair (AE_i, AE_j) is added to the set *OverlappedAEs*.

Secondly, all role assignment axioms are added to TBox after attribute expression definition axioms. For each concept pair $(AE_i, AE_j) \in OverlappedAEs$, we check again whether concept $AE_i \sqcap AE_j$ is satisfiable with respect to current TBox. If concept $AE_i \sqcap AE_j$ is unsatisfiable, i.e. $\top \sqsubseteq AE_i \sqcap AE_j$, 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 all role assignment axioms of conflicted authorization rules from TBox. For each conflict between $rule_i$ and $rule_j$ detected in previous step, if concept subsumption between AE_i and AE_j is satisfiable, i.e. $\top \sqsubseteq AE_i \sqcap AE_j$ or $\top \sqsubseteq AE_j \sqcap AE_i$, 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 $AE_i \sqcap AE_j$ is satisfiable with respect to TBox. For each

satisfiable concept AE_i ó 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 ó AE_j is satisfiable with respect to current TBox. If concept AE_i ó AE_j is unsatisfiable, i.e. $\top \hat{o} AE_i$ ó 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 $\top \hat{o} AE_i$ ô AE_j or $\top \hat{o} AE_j$ ô 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.

Conflict 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 all conflict role assignment axioms. Second, for each conflict between $rule_i$ and $rule_j$, we define a new attribute concept $AE_j' \equiv AE_j$ ó $\neg AE_i$ for $rule_i \geq rule_j$ (or $AE_i' \equiv AE_i$ ó $\neg AE_j$ for $rule_j \geq rule_i$), which specifies users satisfying $rule_j$ but not satisfying $rule_i$. Last, for each concept pair AE_i and AE_j' defined above, we specify new role assignment axioms respectively to AE_i and AE_j': AE_i will be assigned the union of roles of these two rules except all opposite role pairs and AE_j' will be assigned roles of $rule_j$. 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 $rule_i \geq rule_k$ and $rule_k \geq rule_j$, 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 $rule_k$ and $rule_j$. 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.

Conflict 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: $AE_i' \equiv AE_i$ ó $\neg AE_j$, $AE_j' \equiv \neg AE_i$ ó AE_j and $AE_{ij} \equiv AE_i$ ó 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', AE_j' and AE_{ij}: AE_i' will be assigned roles of $rule_i$, AE_j' 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 than one conflict is detected, then this method can not get optimized resolution. More complex algorithms can be given to resolve this problem.

Axioms before rewriting:

$AE_1 \hat{o} \exists assignRole.Role_1$,

AE_2 縞$\exists assignRole.(\neg Role_1 \quad Role_2)$

Axioms after rewriting:

$AE_1' \equiv AE_1$ ó $\neg AE_2$

$AE_1' \hat{o} \exists assignRole.Role_1$

$AE_2' \equiv \neg AE_1$ ó AE_2

AE_2' 縞$\exists assignRole.(\neg Role_1 \quad Role_2)$

$AE_{12} \equiv AE_1$ ó AE_2

$AE_{12} \hat{o} \exists assignRole.Role_2$

(a)

Axioms before rewriting:

$AE_1 \hat{o} \exists assignRole.Role_1$

AE_2 縞$\exists assignRole.(\neg Role_1 \quad Role_2)$

Axioms after rewriting:

$AE_1' \equiv AE_1$ ó $\neg AE_2$

$AE_1' \hat{o} \exists assignRole.Role_1$

$AE_2' \equiv \neg AE_1$ ó AE_2

AE_2' 縞$\exists assignRole.(\neg Role_1 \quad Role_2)$

$AE_{12} \equiv AE_1$ ó AE_2

$AE_{12} \hat{o} \exists assignRole.Role_2$

(b)

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 age≥18, concrete domain [12] should be considered to improve expressive power of the language we use.

References

[1] R. Sandhu, E. Coyne, H. Feinstein and C. Youman, "Role-Based Access Control Model", IEEE Computer, vol.29, no.2, pp.38~47, Feb. 1996.

[2] D. Ferraiolo, R. Sandhu, S. Gavrila and R. Kuhn, "Proposed NIST Standard for role-based access control: towards a unified standard", ACM TISSEC, vol.4, no.3, pp. 224~274, August 2001.

[3] M. Al-Kahtani and R. Sandhu, "A Model for Attribute-Based User-Role Assignment", Proc. 18th Annu. Computer Security Applications Conf., Las Vegas, Nevada, USA, pp.353-362, December 2002.

[4] M. Al-Kahtani and R. Sandhu, "Induced Role Hierarchies with Attribute-Based RBAC", Proc. ACM SACMAT'03, Villa Gallia, Como, Italy, pp.142-148, June 2003.

[5] M A. Al-Kahtani, Ravi Sandhu, "Rule-Based RBAC with Negative Authorization", Proc. 20th Annu. Computer Security Applications Conf., Tucson, Arizona, USA, pp.405~415, December 2004.

[6] S. Benferhat, R. E. Baida, and F. Cuppens, "A Stratification-based Approach for Handling Conflicts in Access Control", Proc. 8th ACM symposium on Access control models and technologies, Como, Italy, pp. 189–195, June 2003.

[7] J. D. Moffett and M. S. Sloman, "Policy Conflict Analysis in Distributed System Management", Journal of Organisational Computing, vol. 4, no. 1, pp.1-22, 1994.

[8] E. Lupu and M. Sloman, "Conflict Analysis for Management Policies", 5th IFIP/IEEE International Symposium on Integrated Network Management, San-Diego,CA, pp.430-443, May 1997

[9] A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. Breedy, L. Bunch, M. Johnson, S. Kulkarni, J. Lott, "KAoS policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement", Proc. IEEE 4th International Workshop on Policy, Lake Como, Italy, pp.93-98, June 2003.

[10] N. Damianou, N. Dulay, E. Lupu, M. Sloman, "The Ponder Policy Specification Language". Proc. Policy 2001: Workshop on Policies for Distributed Systems and Networks, Bristol, U.K., Springer-Verlag, LNCS 1995, pp.18-39, January 2001.

[11] L. Kagal, T. Finin, A. Johshi, "A Policy Language for Pervasive Computing Environment", Proc. IEEE 4th International Workshop on Policy, Lake Como, Italy, pp.63-76, June 2003.

[12] Franz Baader, Diego Calvanese et al., The Description Logic Handbook: Theory, Implementation and Applications, Cambridge University Press, 2003

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.