Information Sciences 179 (2009) 2629–2642

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Specification and enforcement of flexible security policy for active cooperation ☆

Yuqing Sun [a,*], Bin Gong [a], Xiangxu Meng [a], Zongkai Lin [c], Elisa Bertino [b]

[a] School of Computer Science and Technology, Shandong University, No. 27 Shanda South Road, Jinan Shandong 250100, China
[b] CERIAS and Department of Computer Science, Purdue University, USA
[c] Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China

A R T I C L E I N F O

Article history:
Received 1 August 2007
Received in revised form 29 May 2008

Keywords:
Security policy
Access control
Flexibility
Cooperation
RBAC

A B S T R A C T

Interoperation and services sharing among different systems are becoming new paradigms for enterprise collaboration. To keep ahead in strong competition environments, an enterprise should provide flexible and comprehensive services to partners and support active collaborations with partners and customers. Achieving such goals requires enterprises to specify and enforce flexible security policies for their information systems. Although the area of access control has been widely investigated, current approaches still do not support flexible security policies able to account for different weighs that typically characterize the various attributes of the requesting parties and transactions and reflect the access control criteria that are relevant for the enterprise. In this paper we propose a novel approach that addresses such flexibility requirements while at the same time reducing the complexity of security management. To support flexible policy specification, we define the notion of restraint rules for authorization management processes and introduce the concept of impact weight for the conditions in these restraint rules. We also introduce a new data structure for the encoding of the condition tree as well as the corresponding algorithm for efficiently evaluating conditions. Furthermore, we present a system architecture that implements above approach and supports interoperation among heterogeneous platforms.

1. Introduction and motivation

Today enterprises heavily rely on information systems and applications. As a result many tasks that in the past were carried by humans are today automatically executed by computer systems. As a consequence sharing, interoperating and combining services across multiple enterprises are today easier. To keep ahead in strong competition environments, enterprises should provide flexible and comprehensive services to partners and support active collaborations with partners and customers. Achieving such goals requires the development of novel access control models and mechanisms able to address the following requirements.

Flexible specification of access control policies: Consider a scenario of supply chain management, and suppose that an enterprise would grant different access rights to sensitive information to partners according to their qualifications, relationships

☆ Research supported in part by the National High Technology Research and Development Program (863 Program) of China (2006AA01A113), by the US NSF Grant 0712846 "IPS: Security Services for Healthcare Applications", by the NSF grant of Shandong Province of China (Y2008G28), and by the sponsor of CERIAS.

* Corresponding author. Address: School of Computer Science and Technology, Shandong University, No. 27 Shanda South Road, Jinan Shandong 250100, China.

E-mail addresses: sun_yuqing@sdu.edu.cn (Y. Sun), gb@sdu.edu.cn (B. Gong), mxx@sdu.edu.cn (X. Meng), lzk@ict.ac.cn (Z. Lin), bertino@cs.purdue.edu (E. Bertino).

0020-0255/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2009.01.040

or transaction contents. For example, a medical instrument manufacturer would like to grant the permission for accessing detailed information from its latest database to a VIP partner, while would like to grant a browsing permission to summary information to a generic partner. Such flexible access control policies require to perform access control by taking into account a comprehensive set of information about the requesting partner. In such process, each information item may have a different importance. For example, the enterprise may consider more relevant the certification granted by a trusted national organization compared with history transaction. Thus a greater weight should be assigned to certification while a smaller weight should be assigned to history transaction. Policies are also characterized by a temporal dimension and therefore historical data should also be taken into account in access control. We would like to associate different weights with data from different temporal periods in order to reflect temporal criteria, such as that more recent data have greater weights.

Flexible enforcement of access control policies: It is increasingly important to take into account environment factors when enforcing access control policies. To address such requirement an access control mechanism should dynamically monitor state changes of the underlying system and take into account such changes in the policy enforcement process [1,14]. For example, qualifications of a user might be changed by the transactions carried out by the underlying systems; for example the relationship *partner* could be upgraded to *VIP partner* on the assumption that the trade amount exceeds 100 million dollar within one year.

Flexible adaptation of access control policies: To effectively support collaborations, access control policies need to evolve along with the business developments and changes in the enterprise mission. For example, an enterprise may decide to open more sensitive information to partners than in the past, and thus adjust the qualification threshold of *VIP partner* to a lower level 75 million dollar of trade amount.

The above flexibility requirements about access control systems arise not only from industry but also in the military and government domains. Therefore, they are important requirements for the development of access control models and mechanisms to be used in collaborative applications.

Although widely used access control models, like DAC (Discretionary Access Control), MAC (Mandatory Access Control) and RBAC (Role Based Access Control) [7], are appropriate for conventional database and application environments, they do not meet the above requirements. In most cases, they are based on predefined regulations; if changes are required, the access control policies must be manually adjusted which is time consuming and error-prone. When a system supports a large amount of users, characterized by a large diversity of factors, upgrading the access control policies quickly becomes an impossible task.

Recently, extensions to conventional access control models have been proposed to support advanced applications. Examples of such extensions include content-aware access control, user qualification based authorization management, and attribute based access control (ABAC) [2,6,22]. Despite their advantages, they are still not able to support more active security policies, in particular because they do not take into account data transactions. Rule-based access control, like rule-based user-role assignment, is also considered an effective method to support flexible enforcement of access control policies. However, it does not take into account the important requirement that different weights may have to be assigned to the various attributes, characterizing the parties requesting access, nor the fact that authorizations may have to change as consequence of transactions executed by these parties.

In this paper we propose a novel approach that addresses the above flexibility requirements, while at the same time reducing the complexity of security management. Our approach focuses on the specification and enforcement of flexible access control policies, by taking into account the underlying transactions, user attributes, and application context when making access control decisions. Our approach is based on the notion of *restraint rules* which are associated with authorization management processes. We also introduce the concept of *impact weight*; this weight is associated with the condition of each restraint rule. To efficiently enforce such rules, we introduce a new data structure, referred to as *condition tree*, as well as corresponding algorithms to evaluate rule conditions. Furthermore, we also present a system architecture that implements the proposed approach and is able to support interoperation among heterogeneous platforms.

The remainder of this paper is organized as follows. Next section discusses related work. The proposed model is introduced in Section 3. Section 4 discusses the calculation of restraint rules. The subsequent sections present a comprehensive example to illustrate our approach and an overview of the system architecture, respectively. The final section outlines conclusions and future work.

2. Related work

RBAC is a widely adopted access control model to secure resources in an information system [17]. In RBAC, permissions are associated with roles, and users acquire permissions by being assigned roles. Roles within an organization typically have overlapping sets of permissions and thus they can be organized according to role hierarchies. Constraints are used to reflect security policies of an organization, like Separation of Duty (SoD) that formulates multi-person control policies to discourage frauds. Although RBAC provides a powerful mechanism to secure large systems, manual adjustment of authorizations has to be carried out to reflect policy changes by the enterprise.

Recently, approaches have been proposed to support automatic authorization management [9]. AI-Kahtani et al. propose a family of models, called RB-RBAC, to support automatic user-role assignments based on user attributes and a set of authorization rules [3,4]. The central features of RBAC such as roles, role hierarchies, and constraints can be specified based on user attributes. Although their work provides a theoretical foundation for the implementation of automatic authorization assignment, complex constraints, such as user attributes with different weights, cannot be supported. Also their approach does not take into account the underlying transactions.

Another influential work is the Open Architecture for Secure Inter-working Services (OASIS), proposed by Bacon et al., which allows self-managed domains to specify their own access control policies and interoperate using service level agreement [5]. It is a rule-based approach in which a role is activated only when the rules and constraints associated with the role are satisfied. Examples of such constraints include the activation of prerequisite roles or conditions on time and location. Each environment constraint is considered as an atomic proposition. Constraints are tested at the time of activation and triggered along with the activation. Once a constraint becomes false, the role is deactivated at once. Although OASIS is expressive, it does not consider transactions that are the basis of active authorizations, and many other practical requirements.

Our work is also related to content-dependent access control models. One such model has been developed by Bertino et al. [6] in the context of object-oriented databases. Another model has been proposed by Adam et al. [2] in the context of digital libraries; this model supports the specification of flexible authorizations based on the qualifications of users and protected resources. The main difference is that our approach supports dynamic changes to authorizations by taking into account the underlying transactions and the different weights that the various user attributes may have with respect to access control decisions.

Our work is also related to attribute based access control (ABAC), which has recently been proven to be successful in access control for distributed systems [16,19,22]. ABAC manages attribute-to-permission mappings and uses attributes to avoid the need of setting up and managing roles for RBAC. Despite the fact that ABAC simplifies the assignment and revocation of permissions, it does not support the association of different weights with attributes and does not take into account transactions.

There are other related approaches. Herzberg et al. propose the Trust Establishment system which supports the automatic mappings of unknown subjects to predefined business roles by using logical rules and certificates issued by third parties. Each role has one or more rules defining how a subject can be assigned to it [11]. However such approach does not consider the relationships among different rules. In the approach by Zhong et al. [23], users on the Web can be automatically assigned roles by assignment policies according to a trustworthiness threshold specified by a system administrator. However user trustworthiness is computed based on the user performance and therefore the overall security of the system is low. Kern et al. propose the Enterprise RBAC (ERBAC) model, which is implemented as a Security Administration Management (SAM) in Jupiter [12,13]. SAM Jupiter relies on an automatic process for assigning users to roles. However no formal model is given for this process. The problem of access control for collaborative applications has also been discussed in other papers [8,10,15,18,20]; however all the approaches proposed in these papers are very limited in that do not address the problem of assigning different weights to attributes and other factors used in access control decisions and do not take transactions into account.

3. Active authorization management model

This section first presents an overview of the proposed model and then describes in details the various components of the model.

3.1. Overview of the proposed model

Since RBAC is supported by a large number of products and widely adopted in various applications, our model has been defined by extending RBAC with constructs supporting flexible authorization management. Permissions in our model as in RBAC are generally stable and assigned to roles with hierarchies. Users are assigned to different roles so as to acquire the corresponding permissions. Flexible access control policies in our model are supported through the use of restraint rules; these rules are categorized into three classes according to the typical steps followed in authorization processes: authorization rules, assignment rules and activation rules, which are respectively enforced on user-role assignments, permission role assignments and activation of permissions. Fig. 1 presents a graphical representation of our model and shows for each RBAC relationship the relevant class of restraint rules.

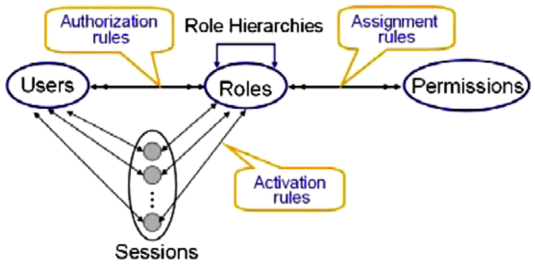

Fig. 1. The active authorization management model.

In the following, we first describe the basic components of our model and the relations among them. We then formally define the conditions and rules. Throughout the discussion, *U*, *R* and *P* denote the sets of users, roles and permissions, respectively.

Users: Users generally are human beings who are assigned responsibilities to perform certain job functions in cooperative business process.

Permissions: Permissions are the approvals for users to access sensitive resources.

Roles: A role is a job function with some associated semantics regarding authority and responsibility. Roles are organized according to role hierarchies based on the inheritance relation, denoted as $\geqslant$. The inheritance relation between roles r_1 and r_2, namely $r_1 \geqslant r_2$, specifies that r_1 inherits all the permissions assigned to r_2 and that all users of r_1 are also users of r_2.

Session: A session is a limited lasting connection between a subject and the protected system, which is generally activated by the subject. A subject would be mapped to some possible roles in the session and is permitted to open multiple sessions at the same time.

Authorization rule: An authorization rule models a regulation concerning user assignments to roles. It takes into account the underlying transaction and application context. It specifies the qualifications that a user must possess for acquiring a role.

Activation rule: An activation rule specifies some prerequisite conditions for the use of a role in a session; such conditions can be based on time and location.

Assignment rule: An assignment rule regulates the assignment of permissions to roles, which is crucial for supporting temporary assignment.

A restraint rule consists of prerequisite conditions and a consequence. Each condition is in form of one or more weighted atomic conditions combined through logic operation connectors. We now present the detailed definitions of such rules by first introducing the notion of atomic condition and then the notion of compound condition. Based on such notions, we then introduce the formal definition of restraint rule.

3.2. Atomic conditions

An atomic condition is a relational expression that specifies a single security or business requirement. Each atomic condition typically contains an attribute and associated value connected by a relational operator. An attribute is a variable of a specific data type, which includes a set of possible values (domain) and operators on them. Attributes in our model are classified into three categories: object attributes, system attributes, and security logic attributes. Object attributes describe user information and business transaction data, like items in database. System attributes define system context, like date, time or IP address. Security logic attributes represent security-relevant information, like SOD.

Definition 1 (*Attribute Set, abbr. ATTR).* An element of attribute set *ATTR* is defined as follows:

1. any item defined in business transaction database,
2. any system attribute like date, time or IP address, which is respectively denoted as *sys.date*, *sys.time* and *sys.IP* in our model,
3. any name of user, role or permission that is defined by security administrator.

Definition 2 (*Relation operator).* A relation operator is an element of the set *RL_OP* = {$\leqslant$, <, =, $\geqslant$, >}, which denote *equal or smaller, smaller, equal, equal or larger, larger* relations, respectively.

Definition 3 (*Atomic condition).* An atomic condition *atc* is in form of (*attr* $\mathscr{op}$ *attr_value*), where $\mathscr{op} \in$ *RL_OP, attr* $\in$ *ATTR* and *attr_value* is an element of the domain of *attr*.

Let *AT_SET* denote the set of all atomic conditions that can be defined in a system. The following example shows several atomic conditions that can be used in a restraint rule for preventing a user from arbitrarily modifying his submitted proposal on the assumption that he is already assigned role *partner* in a bid system.

Example 1. Atomic conditions:
(*sys.time* < *deadline*)
(*u.app_bid* = *bid.serialno*)
(*role*! = *third_party*)

In this example, the first atomic condition means that any modification of proposal is limited before a given time *deadline*. The second requires that the provided serial number of proposal should be same of the submitted proposal, while the third specifies that the role the user is taking on should not be the *third-party* role.

3.3. Compound conditions

More generally, several conditions need to be verified for access control. Therefore we introduce the concept of *compound condition* in our model. In what follows, we first give its overall definition of compound condition and then describe each element in detail.

Definition 4 (*Compound condition, abbr. cpc*). The set *CP_SET* of compound conditions is defined as follows:

- An atomic condition is an element of *CP_SET*.
- If cpc_1 and cpc_2 are elements of *CP_SET*, then $cpc_1 \wedge cpc_2$, $cpc_1 \vee cpc_2$ or $\neg cpc_1$ are elements of *CP_SET*, where $\wedge$, $\vee$, $\neg$ are the logic operators *and*, *or*, and *not*, which satisfy the commutative, associative and distributive laws.
- A weighted condition is an element of *CP_SET*.
- A historical condition is an element of *CP_SET*.

A weighted condition in the above definition is used for the case that we would take into account a comprehensive set of information with different importance. For example, certification granted by a trusted third-party is more relevant than bid price. So, in our model we associate conditions in a *weighted condition* with different weights.

Definition 5 (*Weighted condition*). A weighted condition is defined by the tuple (*ex-list, A, threshold*), where:

- *ex-list* = [E_1, E_2,..., and E_m] is a list of compound conditions,
- *A* is a vector ($\alpha_1, \alpha_2, \ldots, \alpha_m$), where each α_i is an impact weight associated with E_i, $0 < \alpha_i \leqslant 1$, $\sum_{i=1}^{m} a_i = 1$,
- *threshold* is the threshold for this condition to hold with $0 < threshold \leqslant 1$.

The semantic of a weighted condition is that *the condition* holds only if the following inequality is valid:

$$\sum_{i=1}^{m} \alpha_i * E_i \geqslant threshold \tag{1}$$

Example 2. The following are compound conditions:

CP_SET = {at_1, at_2, at_3, cp_1}
at_1 = (*T.amount* > 10,000)
at_2 = (*T.sale* > 5,500,000)
at_3 = (*user.certification* = ISO9000)
cp_1 = ((at_1, at_2, at_3), (0.3, 0.3, 0.4), 0.6)

Example 2 gives three atomic conditions at_1, at_2, and at_3 and a compound condition cp_1. The semantics of at_1 is that the trade amount should be larger than 10,000 pieces, while at_2 means that the trade money should be larger than 5.5 million dollar and at_3 means that a user should hold the certification ISO9000. In cp_1, different weights (0.3, 0.3, 0.4) are associated with the corresponding conditions (at_1, at_2, at_3) respectively. The threshold is set to 0.6. Therefore, we can argue that cp_1 holds only when the following inequality is valid.

$$(at_1 * 0.3 + at_2 * 0.3 + at_3 * 0.4) \geqslant 0.6$$

We observe that if any two of the three atomic conditions in cp_1 are TRUE, cp_1 is TRUE.

Furthermore, in many cases we should take into account history transactions. For example, although a partner had a good reputation during the past ten years and satisfied the qualification for taking on the role of *senior partner*, it still cannot be assigned the role *VIP* if it has made no progress in recent years. To model such type of condition, we adopt the *TRACE* mechanism that keeps track of the dynamic relationship of a subject over time. The total considered history can be divided into several periods and each period is called an interval. We can assign different influence weights to intervals according to their impacts to the total result. The *TRACE* mechanism periodically updates data so as to reflect *fresh* progress, such as *past five years from now* means from year 2004 until 2008 while for next year it would be from 2005 until 2009. We now introduce the concept *historical condition* to express the weighted history based authorization.

Definition 6 (*Historical condition*). A historical condition is defined by the tuple (*Ex_Mtrx, A, B, threshold*), where

- *Ex_Mtrx* is a vector ($E_1, E_2, \ldots, E_k$) and each E_i is a condition vector ($E_{i1}, E_{i2}, \ldots, E_{im}$), $E_i \in CP\_SET$,
- *A* is a vector ($\alpha_1, \alpha_2, \ldots, \alpha_m$), with $0 < \alpha_j \leqslant 1$, $\sum_{i=1}^{m} a_j = 1$, α_j is associated with each element E_{ij} in each row E_i and denotes the impact weight of each condition E_{ij} in E_i,
- *B* is a vector of interval weights ($\beta_1, \beta_2, \ldots, \beta_k$) with $0 < \beta_i \leqslant 1$, $\sum_{i=1}^{k} \beta_i = 1$, β_i is associated with each row E_i and denotes the impact factor of condition E_i in *Ex_Mtrx*, and
- *threshold* $\in [0, 1]$ is the threshold for this condition to hold.

The semantics of a historical condition is that the condition holds only when the inequality (2) is verified. Each condition E_{ij} in the determinant of (2) is easy to evaluate since it is either *TRUE* or *FALSE*, respectively denoted as 1 or 0. The threshold is set according to the security requirements and business mission of the enterprise, and could be dynamically adjusted.

$$\begin{pmatrix} E_{11}, E_{12}, \ldots, E_{1m} \\ E_{21}, E_{22}, \ldots, E_{2m} \\ \cdots \\ E_{k1}, E_{k2}, \ldots, E_{km} \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \cdots \\ \alpha_m \end{pmatrix} (\beta_1, \beta_2, \ldots, \beta_k) \quad \geqslant \textit{threshold} \tag{2}$$

3.4. Restraint rules

As we have already discussed, access control policies are expressed as restriction rules and enforced on three major steps of authorization process: user-role assignment, permission role assignment, and permission activation. We thus define five categories of rules on these aspects that are: permission assignment rules, user authorization rules, role update rules, activation rules, and repeal rules.

Definition 7 (*Rule type, abbr. Rtype).* A rule type is an element of set *RTYPE* = {*Pasgn*, *Uauth*, *Rupdt*, *Pactv*, *Repeal*}, where *Pasgn*, *Uauth*, *Rupdt*, *Pactv* and *Repeal* denote the restraint type of permission assignment rule, user authorization rule, role update rule, permission activation rule, and repeal rule, respectively.

Definition 8 (*Restraint rule).* A restraint rule has the form *Rtype*:: *rule_name* = $A \to B$, where $Rtype \in RTYPE$ is a rule type, and *rule_name* is the name given to the rule by the security administrator. A is called the prerequisite and B is called the consequence.

Definition 9 (*Permission assignment rule, abbr. Pasgn).* A permission assignment rule *Pasgn* specifies the prerequisite condition of a permission role assignment and has the form

$$Pasgn :: rule\_name = (p_i, cpc) \to (r_j)$$

where $p_i \in P$, $r_j \in R$, and $cpc \in CP\_SET$.

Its semantics is that permission p_i can be assigned to role r_j only when condition *cpc* is satisfied. Or we say that condition *cpc* if true results in the assignment of p_i to r_j.

Definition 10 (*User authorization rule, abbr. Uauth).* A user authorization rule *Uauth* controls the assignment of a role to a user. It has the form

$$Uauth :: rule\_name = (r_i, cpc) \to (u_j)$$

where $r_i \in R$, $u_j \in U$, and $cpc \in CP\_SET$.

This rule means that role r_i can be assigned to user u_j only when condition *cpc* is satisfied. Or we say that condition *cpc* yields role r_i to u_j.

Definition 11 (*Role update rule, abbr. Rupdt).* A role update rule *Rupdt* specifies the prerequisite for a user to take on a senior role r_j on the assumption that he has been assigned role r_i. It has the form:

$$Rupdt :: rule\_name = (r_i, cpc) \to (r_j)$$

where r_i, $r_j \in R$, and $cpc \in CP\_SET$.

We say that any user who has already been assigned role r_i is permitted to take on a senior role r_j on the condition of *cpc* being true.

Definition 12 (*Permission activation rule, abbr. Pactv).* A permission activation rule *Pactv* specifies the prerequisite for a permission to be active and has the form:

$$Pactv :: rule\_name = (cpc) \to (p_i)$$

where $p_i \in P$, and $cpc \in CP\_SET$. We say that permission p_i can be activated or take effect only when condition *cpc* is satisfied. Or we say that condition *cpc* yields p_i effective.

Definition 13 (*Repeal rule, abbr. Repeal).* A repeal rule *Repeal* has the form:

$$Repeal :: rule\_name(cpc)$$

where $cpc \in CP\_SET$.

This means when condition *cpc* is satisfied, the rule identified by *rule_name* is repealed. Or we say that condition *cpc* results in rule *rule_name* to become ineffective. This rule is useful for active authorization to support the repeal of a restraint rule.

Fig. 2 reports the syntax of restraint rule language.

Syntax of Restraint Rule Specification Language

Terminal symbol set = {$\leq$, $<$, = , $\geq$, $>$, $\wedge$, $\vee$, $\neg$, 0,1,2,3,4,5,6,7,8,9, .}
Non-Terminal symbol set = {attributes, attribute value, users, roles, permissions, Date, Time, IP }*

The production rule = rule type:: rule name= ((Roles | Permissions,) conditions) $\rightarrow$(Role | User | Permission | EMPTY) +
Rule type =*Pasgn* | *Uauth* | *Rupdt* | *Pactv* | *Repeal*
Condition = atom condition | weighted condition | historical condition |(Condition) logic-operator Condition
Atom condition = (attribute relation-operator attribute value)
Relation-operator = $\leq$ | $<$ | = | $\geq$ | $>$
Logic-operator = $\wedge$ | $\vee$ | $\neg$
Weighted condition = ((condition, ..., condition), weighted vector, threshold)
Historical condition = ((weighted condition, ..., weighted condition), weighted vector, threshold)
Weighted vector = (α_1, α_2, ..., α_m)
α_i = 0< $\alpha_i \leq 1$
Threshold = 0< threshold $\leq$ 1
Roles = Role | { Roles set }
Attribute = { *specified by enterprise system administrator* }
Role = { *specified by enterprise system administrator*}
User = { *specified by enterprise system administrator*}
Permission = { *specified by enterprise system administrator*}
Rule name = { *specified by enterprise system administrator*}

Note: *The Non-Terminal symbols are customized for concrete enterprise
+The symbol "$\rightarrow$" means "generate" or "yield"

Fig. 2. The syntax of restraint rule specification language.

4. Calculation of restraint rules

Since security policies are expressed in terms of restraint rules, access control relies on the evaluation of the conditions in restraint rules. In this section, we focus on how to efficiently evaluate conditions. We first explore a new data structure, referred to as *condition tree*, to encode conditions. Then we discuss how to efficiently evaluate the conditions with the help of *key nodes* and *strong nodes* on *condition tree*. Finally, we give the details of determination process and corresponding algorithm.

4.1. Condition tree

Generally, there are three issues when dealing with an access control model like ours: how to represent complex policy in a unified form so that they can be easily analyzed; how to identify all required information for access control; how to ensure that policies are consistent. In order to address such issues, we propose the *condition tree* data structure for representing each condition in each restraint rule.

A *condition tree* is a connected, acyclic graph. It has a unique *root* node. Each *node* is either a *leaf* or an *internal node*. An internal node has one or more *child* nodes and is called the *parent* of its child nodes. All children of the same parent node are siblings. Compared with general tree structure, a *condition tree* has two distinguishing characteristics: (1) each branch is associated with a weight value that denotes the impact factor of a child node to its parent; (2) each node is associated with different semantics according to its functionality. Leaf nodes are called *atomic nodes* and denote atomic conditions. Internal nodes are called *threshold nodes*, and each of them is associated with a value denoting the threshold for the condition in the subtree to hold. Here we give the formal definition of the notion of condition tree and depict it in Fig. 3.

Definition 14 (*Condition tree*). A condition tree is defined as the the tuple (V, E, *root*), where V is the set of nodes, E is the set of edges, root $\in V$ is a special node that has no incoming edges. A condition tree satisfies the following conditions:

- It has a unique root.
- All nodes, except the root node can be divided into several sets; each set is iteratively organized as a sub tree and does not intersect with each other.
- All nodes are classified into two categories: *atomic node* and *threshold node*. An *atomic node* denotes an atomic condition that can be evaluated immediately according to the underlying transaction data or system context. A *threshold node* gives the threshold of a sub tree to be satisfied.
- Each edge $e_j = (v_i, v_j, weight_j)$ is associated with a value $weight_j \in (0, 1]$, which denotes the weight of child node v_j.

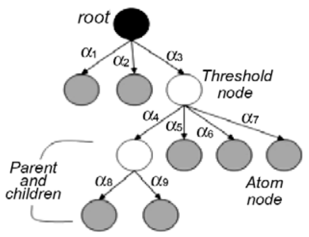

Fig. 3. Condition tree. Grey node: atom condition; white node: threshold; deep grey node: root.

The semantics of a condition tree is as follows: for any edge (v_i, v_j, *weight*$_j$), if the child node v_j is TRUE then it contributes *weight*$_j$ to its parent v_i. For any parent node v_i, if the total contribution from its children exceeds the *threshold*$_i$ in v_i, v_i is evaluated to TRUE.

Now we describe how to express various forms of conditions defined in Section 3 by the *condition tree*. An atomic condition is directly expressed as a leaf node while a compound condition must be translated into a sub tree with a threshold node, a set of atomic nodes and corresponding weighted branches. The translation rules are given below. Rule 1 and rule 2 are for logic conditions while rule 3 is for weighted conditions and historical conditions.

RULE 1 (*TR_Intersection*). Let $at_1, at_2, \ldots,$ and at_m be conditions. Suppose that they are expressed by condition trees $T_1, T_2, \ldots,$ and T_m, where $T_i = (V_i, E_i, Root_i)$. The condition tree resulting from their logic intersection, namely $at_1 \wedge at_2 \wedge \cdots \wedge at_m$, is the tree *IntsT* = (V, E, *Root*), where $V = V_1 \cup V_2 \cup \cdots \cup V_m \cup \{Root\}$, $E = E_1 \cup E_2 \cup \cdots \cup E_m \cup \{(Root, Root_i)| \ i = 1, 2, \ldots, m\}$, the threshold for *Root* is 1, and the weight of each branch (*Root*, *Root*$_i$) is $1/m$. We show the intersection tree in Fig. 4(1).

RULE 2 (*TR_Union*). Let $at_1, at_2, \ldots,$ and at_m be conditions. Suppose that they are expressed as condition trees $T_1, T_2, \ldots,$ and T_m, where $T_i = (V_i, E_i, Root_i)$. The condition tree resulting from their logic union combination, namely $at_1 \vee at_2 \vee \cdots \vee at_m$, is the tree *UnionT* = (V, E, *Root*), where $V = V_1 \cup V_2 \cup \cdots \cup V_m \cup \{Root\}$, $E = E_1 \cup E_2 \cup \cdots \cup E_m \cup \{(Root, Root_i)| \ i = 1, 2, \ldots, m\}$, the threshold of *Root* is $1/m$, and the weight of each branch (*Root*, *Root*$_i$) is $1/m$. We show the union tree in Fig. 4(2).

Notice that the selection of the branch weight and root threshold could be different. The only requirement for logic intersection operation is that the threshold of the root is equal to the sum of all children branches weights since logic intersection operation means the final TRUE only on condition of all sub conditions TRUE. Similarly, for union operation, the requirement is that the root threshold should be less than the weight of any child branch. In our model, for the purpose of normalization, we still choose a value within [0, 1] and set the average value $1/m$ for each branch.

Since the logic negative operator only has one parameter, we can consider it as an atomic condition. Now we discuss the notion of weighted condition. The formal translation rule is given below and illustrated by Fig. 4(3).

RULE 3 (*TR_Weight*). Let $C_1, C_2, \ldots,$ and C_m be conditions. Suppose that they are expressed as separated condition trees $T_1, T_2, \ldots,$ and T_m, where $T_i = (V_i, E_i, Root_i)$. The condition tree *WeightT* = (V, E, *Root*) of condition $((C_1, C_2, \ldots, C_m), (\alpha_1, \alpha_2, \ldots, \alpha_m), threshold_w)$ is such that: $V = V_1 \cup V_2 \cup \cdots \cup V_m \cup \{Root\}$, $E = E_1 \cup E_2 \cup \cdots \cup E_m \cup \{(Root, Root_i)| \ i = 1, 2, \ldots, m\}$, the threshold of *Root* is *threshold*$_w$, and the weight of each branch (*Root*, *Root*$_i$) is α_i.

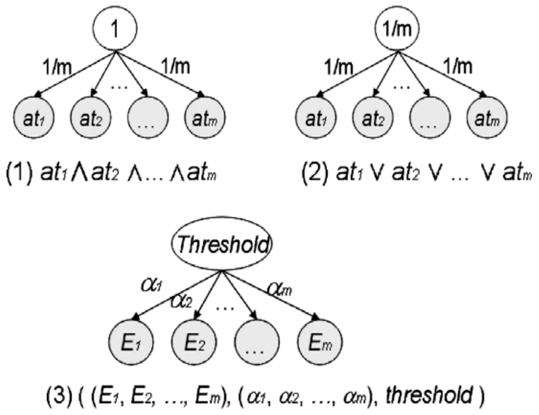
1
1/m
1/m
at_1
at_2
…
at_m
(1) $at_1 \wedge at_2 \wedge \ldots \wedge at_m$
1/m
1/m
1/m
at_1
at_2
…
at_m
(2) $at_1 \vee at_2 \vee \ldots \vee at_m$
Threshold
α_1
α_2
α_m
E_1
E_2
…
E_m
(3) ((E_1, E_2, …, E_m), (α_1, α_2, …, α_m), threshold)

Fig. 4. *Condition Tree* for logic and weighted conditions.

Finally we discuss the historical condition tree. As discussed in Section 3, a history condition can be regarded as a weighted condition with a set of *condition vectors*. Each *condition vector* could be further divided into sub weighted conditions. So, we can express a historical condition as a combined condition tree by iteratively using translation rule *TR_Weight*.

So far, we have been able to express all kinds of conditions into a unified data structure, that is, the *condition tree*, by using the above translation rules. In the following, we discuss how to efficiently evaluate conditions expressed as condition tree.

4.2. Key nodes and strong nodes

According to the definition of a condition tree, the overall evaluation of the tree can be computed based on the evaluation of all its children atoms. However, we do not need to follow such approach all the time. Let us first consider the example shown in Fig. 5, which is the condition tree representing the following compound condition *Cpdc*:

$Cpdc = ((at\text{-}e_1, at\text{-}e_2, codc_1), (0.3, 0.3, 0.4), 0.8)$
$Cpdc_1 = ((cpdc_2, at\text{-}e_3, at\text{-}e_4, at\text{-}e_5), (0.34, 0.2, 0.3, 0.16), 0.6)$
$Cpdc_2 = (at\text{-}e_6 \vee at\text{-}e_7)$

In such condition each $at\text{-}e_i$ is an atomic condition, and each $Cpdc_j$ is a compound condition. *Cpdc* can be rewritten in terms of atomic conditions as follows:

$Cpdc = ((at\text{-}e_1, at\text{-}e_2, ((at\text{-}e_6 \vee at\text{-}e_7), at\text{-}e_3, at\text{-}e_4, at\text{-}e_5), (0.34, 0.2, 0.3, 0.16), 0.6), (0.3,0.3,0.4),0.8))$

From this example, we can see that atom conditions $at\text{-}e_1$ and $at\text{-}e_2$ are more relevant to the final evaluation of the whole condition *Cpdc* than atoms $at\text{-}e_3$ and $at\text{-}e_5$ since if either $at\text{-}e_1$ or $at\text{-}e_2$ is FALSE, the overall evaluation of the tree is FALSE and thus it is not necessary to evaluate other conditions. However, even if we know any result of $at\text{-}e_3$ or $at\text{-}e_5$, no matter TRUE or FALSE, we still need to evaluate other conditions so as to achieve the final result. So we argue that $at\text{-}e_1$ and $at\text{-}e_2$ are important nodes to accelerate the evaluation of such condition tree. The next questions are that whether there are other such *important nodes* and that how to find them.

For this purpose, we should analyze again the above example and look for the reason what makes those atomic nodes more important than other nodes. We observe that if the atomic node $at\text{-}e_1$ were FALSE, the sum of its sibling weights would be less than its parent threshold and result in the parent to be always evaluated to FALSE. Furthermore, if such case happens to the nodes on the path from an atomic node n_i to the root, n_i being FALSE will result in the final evaluation of the whole tree being FALSE. Similarly, we consider the contrary side of the problem. That is if one atom were TRUE the final result of condition is TRUE immediately. Atoms $at\text{-}e_2$ and $at\text{-}e_6$ of the example in Fig. 6 show such case.

To formally describe above two types of node, we introduce concepts *key node* and *strong node*. Both of them can greatly influence the final result of a condition, namely half of their instances could immediately result in the final evaluation of a condition. The formal definitions and implement algorithm are given below.

Definition 15 (*Key node, abbr. Knode*). A key node *Knode* is an atom node of a condition tree such each branch on the path $path_k$ from *Knode* to the root is associated with a weight larger than the complement of its parent threshold in 1. More precisely, the following inequality holds:

$$Branchwgt_j > (1\text{-}threshold_j)$$

where $j = 1, 2, \ldots, k$ and k is the number of branches in $path_k$, $Branchwgt_j$ and $threshold_j$ are the weight of each branch and the threshold of its associated parent.

Definition 16 (*Strong node, abbr. Snode*). A strong node *Snode* is an atom node of a condition tree such on the path $path_s$ from *Snode* to the root each branch is associated with a weight larger than the threshold of its parent. More precisely, the following inequality holds:

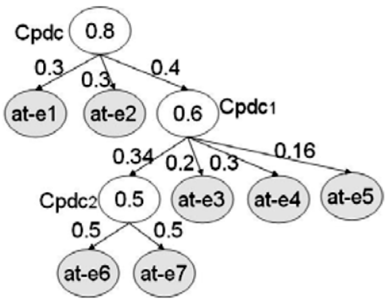

Fig. 5. An example of a condition tree.

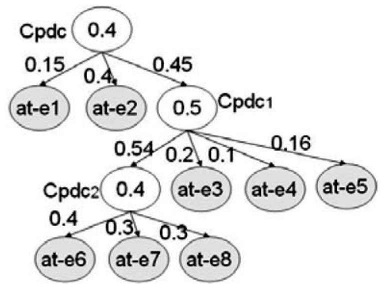

Fig. 6. Strong nodes in a condition tree.

$$Branchwgt_j \geqslant threshold_j$$

where $j = 1, 2, \ldots, k$ and k is the number of branches in $path_s$, $Branchwgt_j$ and $threshold_j$ are the weight of each branch and the threshold of its associated parent, respectively.

The detection process of key nodes and strong nodes is given in Procedure *FindKeyAndStrongNodes*. Since this is a process from a leaf node to the root, the maximum complexity of procedure is equal to the height of a condition tree. Generally, key node and strong node more possibly appear for a sub tree rather than for a whole condition tree, and with larger weight in a condition. Thus we would mark it as a *sub-class* important node so as to take a different priority for calculation. Additionally, one node would be only either key node or strong node. Obviously, taking into account above aspects could efficiently accelerate the determination of a condition.

Procedure for finding key nodes and strong nodes.

```
Procedure FindKeyAndStrongNodes ($V$, $E$, $root$)
Input: compound condition tree ($V$, $E$, $root$)
1.  for each atom node $n_i \in V$, do step 2–12
2.    $tag = 0$; $q = n_i$
3.    find parent $p$ of $q$
4.    while $q$ != $root$ and $branch_q > 1$-$threshold_p$ do
5.      {$q = p$
6.        find parent $p$ of $q$; $tag = 1$;}
7.    if $q = root$ then mark $q$ as key node
8.    if $tag = 0$
9.    then while $q$ != $root$ and $branch_q \geqslant threshold_p$ do
10.       {$q = p$
11.         find parent $p$ of $q$}
12.     if $q = root$ then mark $q$ as strong node
13. end procedure
```

Now we analyze how much important nodes could improve the efficiency of calculation. Suppose that there are n atom nodes in a condition tree. Since each atom node has the same probability to be TRUE or FALSE, the result of a key node or strong node can induce the final result of the whole condition with the probability of 0.5. So, if there were totally *k key nodes* and *strong nodes* together for the whole tree, the probability of an immediate final determination is $(1/2 + 1/2^2 + \cdots + 1/2^k)$. From another view, the probability of indetermination after we calculate all important nodes is left only $1/2^k$. If we consider more *sub-class* important nodes, the efficiency of computation would continually improve.

4.3. Determination process

After specifying the policy, we translate each restraint rule into a condition tree and mark *key nodes* and *strong nodes*. Then the policy is enforced by determining the condition in each restraint rule, which takes into account the transaction data and application context. In this section we would discuss the process of condition determination.

First, we one by one check the result of *key nodes* and *strong nodes*. If any *key node* is FALSE or any *strong node* is TRUE, the final result is determined immediately. Otherwise, we continue calculating other atoms until we reach the final result of the whole condition. The complete algorithm is illustrated in Algorithm *ConditionDetermination* below.

Algorithm for condition determination.

```
Algorithm ConditionDetermination($V$, $E$, $root$)
Input: compound condition tree ($V$, $E$, $root$)
1.  for each key node $kn \in V$, do step 2–4
2.    calculate the associated atom condition $AT_{kn}$
3.    if the result of $AT_{kn}$ is FALSE
4.        then return FALSE
5.  for each strong node $sn \in V$, do step 6–8
6.    calculate the associated atom condition of $AT_{sn}$
7.    if result $AT_{sn}$ is TRUE
8.        then return TRUE
9.  for each node $n \in V$ without mark
10.     calculate the associated atom condition of$AT_n$
11.  post-order travel the condition tree do step 12–14
12.     for each inter node $n_j \in V$ compute the sum of contribution from its each child $E_i$: $Con = \sum_{i=1}^{m} \alpha_i * E_i$
13.          if $Con < threshold_j$
14.             then return FALSE
15. return TRUE
```

5. Case study

In this section, we would present a comprehensive example to illustrate how to specify and enforce a flexible security policy by our method.

Suppose that in a supply chain management system, a supermarket enterprise, like Wal-Mart, is planning to enforce following flexible security policies in transaction databases and application systems:

1. Access rights to sensitive information are assigned to users according to the roles they have. Senior roles are assigned more rights than juniors.
2. Users are authorized different roles depending on their qualification, such as attributes, certifications, and underlying transactions etc.

Name	ID	Description
at1	A001	Sys.data < deadline
at2	A002	user.certification=ISO9000
at3	A003	user.certification=MC
at4	A004	user.rank < 500
at5	A005	T.amount > 10000
at6	A006	T.Sale > 5500000
cp1	C001	((at2, at3),(0.5, 0.5),0.5)
cp2	C002	((at5,at6),(0.5,0.5),0.5)
cp3	C003	((at5,at6,at2),(0.3,0.3,0.4),0.6)
cp4	HC01	((cp2,cp2,cp2,cp2),(0.4,0.3,0.2,0.1),0.7)
cp5	C006	((at1,at2,at3,at4),(0.5,0.2,0.2,0.1),0.6)
cp6	C007	((at1, cp1, at4),(0.3, 0.3, 0.4),1)
……		

(a) Condition Table

Name	ID	Type	Description	Target	Mark
ES1	R001	Rupdt	(ES,cp2)	SES	at5, at6
ES2	R002	Rupdt	(Sph,cp3)	SPhS	
ES3	R003	Rupdt	(SES,cp4)	VIP	
AS1	R004	Pasgn	(p1,cp6)	RGT	at1,at3
AP1	R005	Ractv	(at1)	p1	at1
UR1	R006	Uauth	(NorM, cp5)	RGT	
……					

(b) Restraint Rule Table

Name	ID	Rules	Description	…
P1	1101	AS1， AP1	Submit bid	
P2	1102	AP2	Modify bid	
……				

(c) Permission Table

Name	ID	Description	Roles	Rules	…
AnM	G110001	anonymous	N	UR1	
GE	E320801	ElecAppl	VIP		
Haier	E320802	ElecAppl,	SES		
Philip	E320803	ElecAppl	ES		
……					
CVS	P507601	Pharmacy	SPhS		
CuiThy	P507602	Pharmacy	PhS		
BES	P507701	Pharmacy	PhM		
……					

(d) User Table

Name	ID	Permissions	Parents	Rules	Description	…
VIP	901	p1,p2,p3,p4,p5,…p30	NULL		VIP.supplier	…
SES	101	p1,p2,p3,p4,p5	VIP	ES3	Senior.Elec.supplier	…
ES	111	p1,p2,p5	SES	ES1	Elec.supplier	
SPhS	201	p1,p2,p6,p7,p8	VIP	…	Senior.Phar.supplier	
PhS	211	p1,p2,p7	SPhS	ES2	Phar.supplier	
RGT	001	P10,p12,p13,p15	PhS, ES	…	Registered user	…
……						

(e) Role Table

Fig. 7. A comprehensive example.

3. The roles that a user takes on can be updated by associated restriction rules.
4. Some access rights are restrained by application context, like time, date, address and other condition.

To implement the above policies in detail, we should specify them as concrete restraint rules by using our proposed *policy specification language*. We adopt a unified form of weighted conditions to synthetically consider different factors. The details are as follow.

We would first specify the atomic and compound conditions, which are shown in Fig. 7a *condition table*. Then we define the restraint rules, which are described in the following and summarized in the *restraint rule table* of Fig. 7b.

$$Rupdt :: ES1 = (ES, cp2) \rightarrow (SES)$$

This is a role update rule that defines the prerequisite that how a user could take on a senior role *SES* (senior electric appliance supplier) on the assumption that it has already been assigned role *ES* (electric appliance supplier). Such qualification is *cp2*, namely the trade amount should exceed 5.5 million dollar or the sale quantity is larger than 10,000 pieces.

$$Rupdt :: ES2 = (PhS, cp3) \rightarrow (SPhS)$$

This is also a role update rule. Like ES1, it defines how role *PhS* (pharmacy supplier) can be updated to role *SPhS* (senior pharmacy supplier).

$$Rupdt :: ES3 = (SES, cp4) \rightarrow (VIP)$$

This is a role update rule and describes how the role *SES* can be updated to role *VIP*. Unlike the above role update rules, it is associated with a weighted historical condition and takes into account transactions in the past four years. Although the consideration of each year is the same condition *cp2*, the impact weights of them are different.

$$Pasgn :: AS1 = (p1, cp6) \rightarrow (RGT)$$

This is an assignment rule that gives the prerequisite for normal registered role *RGT* to acquire permission *p1*. It defines the following conditions: the submit time should be within deadline, a user should hold a certification of ISO9000 or be certified by Commercial Ministry, as well as the rank is within top 500.

$$Pactv :: AP1 = (at1) \rightarrow (p1)$$

This is a permission activation rule that defines the revocation prerequisite of permission *p1*, namely the modification of a bid is restricted before deadline.

$$Uauth :: UR1 = (NorM, cp5) \rightarrow (RGT)$$

This is a user authorization rule that presents the qualification for an anonymous browser to acquire role *RGT* (registered user). After the rules established, key nodes and strong nodes are detected and marked in the *restraint rule table*. With the help of the *condition table*, we can reuse the specified atomic or compound conditions when defining other rules.

To enforce the restraint rules, some necessary modifications on traditional data should be made. The added information is depicted as the shaded columns in the*permission table*, *user table* and *role table* of Fig. 7c–e, respectively.

6. System overview

In this section, we would discuss how to integrate the proposed method into legacy systems. Generally, a security administrator of an enterprise is responsible for specifying restraint rules according to security requirements and business missions. The system architecture to implement the proposed model is presented in Fig. 8, in which black thin arrow lines denote commands while thick arrow lines denote data flows. To support interoperation among heterogeneous platform, we adopt XACML to express security policies. XACML is the extensible access control language specified by OASIS Standard bodies.

Suppose a manufacturer is domain A in Fig. 8 while its partner is domain B. There are three main components that are parts of the system architecture. The first component is the application system, which denotes an underlying legacy system processing operational business transactions, like the ERP system in an enterprise. The second component is the platform to specify and manage restraint rules, denoted as *security specification* in Fig. 8. The third component is the access control system that processes authorization requests, referred to as *security decision*. The security management is summarized as two phases:

Policy generation phase. This phase is represented as circle nodes in Fig. 8. The *policy specification and administration* module is responsible for supporting the specification and management of restraint rules. It periodically queries the transaction database and publishes the XACML security policies according to the specified restraint rules.

Authorization decision phase. This phase is represented as diamond nodes in Fig. 8. When the system receives an access request, the *policy decision* module matches the request parameters with the activation conditions of permission. If the conditions are related with system attributes, the *system attributes acquirement* module is invoked and returns the system parameters. If all conditions are satisfied, the *policy decision* module transfers the request to the application system. Otherwise, it refuses the request. The corresponding 4 steps are shown as diamond nodes in Fig. 8.

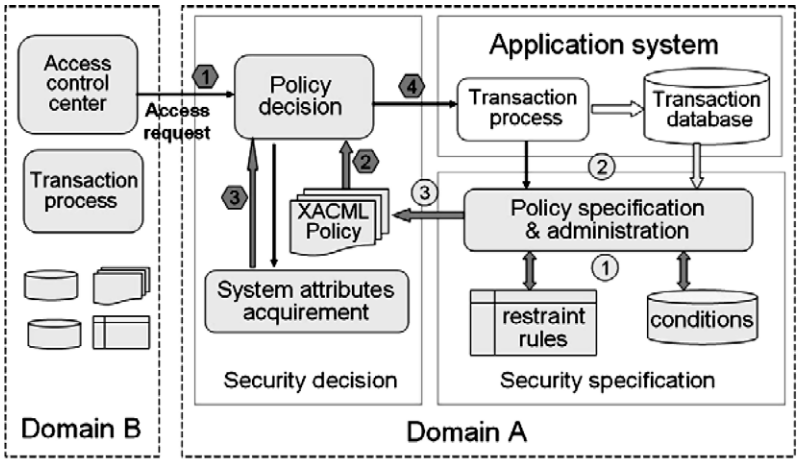

Fig. 8. System architecture.

The above discussion shows that the proposed model can provide active access control to enterprise resources for flexible policies and collaborations. As an example, some of our research results have been applied to a practical system of "*Property Rights Exchange System*" [21]. Details are omitted here.

7. Conclusions and future work

In this paper we have proposed a novel approach to specify and enforce flexible security policies for active cooperation. It extends the RBAC model with the notion of *restraint rules* that are enforced by authorization processes. To support flexible policy specification, we introduce the concept *impact weight* in the conditions of restraint rules. We have also presented the *condition tree* data structure that efficiently supports condition evaluation. Furthermore, we have discussed the system architecture implementing above approach and supporting interoperation among heterogeneous platforms.

As part of future work, we plan to extend this work to address the issue of optimization of restraint rules. We also plan to investigate to combine trust and delegation with this model so as to meet special authorizations. Moreover, we plan to introduce the notion of semantic authorization in our approach to support universal access control on web.

Acknowledgements

The authors would like to sincerely appreciate Dr. Ninghui Li, from CERIAS and Department of Computer Science of Purdue University, for his suggestions and comments on this paper. We also thank the anonymous reviewers for their valuable suggestions. Part of the first author's work was completed when she was as a visiting scholar at Purdue University of US. This work is supported by the National High Technology Research and Development Program (863 Program) of China (2006AA01A113), by the US NSF Grant 0712846 "IPS: Security Services for Healthcare Applications", by the NSF grant of Shandong Province of China (Y2008G28), and by the sponsor of CERIAS.

References

[1] R. Adaikkalavan, S. Chakravarthy, Active authorization rules for enforcing role-based access control and its extensions, in: Proceedings of the 21st International Conference on Data Engineering Workshops, ICDEW, 2005, p. 1197.
[2] N.R. Adam, V. Atluri, E. Bertino, E. Ferrari, A content-based authorization model for digital libraries, IEEE Transactions on Knowledge and Data Engineering 14 (2) (2002) 296–315.
[3] M.A. AI-Kahtani, R.S. Sandhu, A model for attribute-based user-role assignment, in: Proceedings of 18th Annual Computer Security Applications Conference, 2002, pp. 353–362.
[4] M.A. AI-Kahtani, R. Sandhu, Induced role hierarchies with attribute-based RBAC, in: Proceeding of ACM SACMAT, Como Italy, 2003, pp. 142–148.
[5] J. Bacon, K. Moody, W. Yao, A model of OASIS role-based control and its support for active security, ACM Transaction on Information and System Security (TISSEC) 5 (4) (2002) 492–540.
[6] E. Bertino, P. Samarati, S. Jajodia, An extended authorization model, IEEE Transactions on Knowledge and Data Engineering 9 (1) (1997) 85–101.
[7] Matthew Bishop, Computer Security: Art and Science, Addison-Wesley, 2003.
[8] J. Biskup, S. Wortmann, Towards a credential-based implementation of compound access control policies, in: Proceedings of the Ninth ACM Symposium on Access Control Models and Technologies, New York, USA ACM SACMAT'04, 2004, pp. 31–40.
[9] S. Busch, B. Muschall, G. Pernul, T. Priebe, Authrule: a generic rule-based authorization module, in: Proceedings of the 20th Annual IFIP WG 11.3 Working Conference on Data and Application Security, 2006, pp. 267–281.
[10] D.F. Ferraiolo, S. Gavrila, V. Hu, D.R. Kuhn, Composing and combining policies under the policy machine, in: Proceedings of ACM SACMAT, Stockholm, 2005, pp. 11–20.
[11] A. Herzberg, Y. Mass, J. Mihaeli, Access control meets public key infrastructure, or: assigning roles to strangers, in: Proceedings of the IEEE Symposium on Security and Privacy, 2000, pp. 2–14.

[12] A. Kern, Advanced features for enterprise-wide role-based access control, in: Proceedings of the 18th Annual Computer Security Applications Conference, Las Vegas, Nevada, USA, December 2002, pp. 333–342.
[13] A. Kern, A. Schaad, J. Moffett, An administration concept for the enterprise role-based access control model, in: Proceedings of SACMAT'03, June 1–4, Como, Italy, 2003, pp. 3–11.
[14] P. McDaniel, On context in authorization policy, in: Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies, Como, Italy, 2003, pp. 80–89.
[15] J.S. Park, K.P. Costello, T.M. Neven, J.A. Diosomito, A composite RBAC approach for large, complex organization, in: Proceedings of ACM SACMAT, 2004, pp. 163–171.
[16] T. Priebe, W. Dobmeier, N. Kamprath, Supporting attribute-based access control with ontologies, in: Proceedings of the First IEEE International Conference on Availability, Reliability and Security (ARES'06), 2006, pp. 465–472.
[17] B. Shafiq, J.B.D. Joshi, E. Bertino, A. Ghafoor, Secure interoperation in a multidomain environment employing RBAC policies, IEEE Transaction on Knowledge and Data Engineering 17 (11) (2005) 1557–1577.
[18] R.S. Sandhu, D. Ferraiolo, The NIST model for role-based access control: towards a unified standard, in: Proceedings of the Fifth ACM Workshop on RBAC, 2000, pp. 47–63.
[19] B. Shields, O. Molloy, G. Lyons, J. Duggan, Using semantic rules to determine access control for web services, in: Proceedings of the 15th International Conference on World Wide Web, 2006, pp. 913–914.
[20] E. Song, R. Reddy, R. France, I. Ray, G. Georg, R. Alexander, Verifiable composition of access control and application features, in: Proceedings of SACMAT, Stockholm, 2005, pp. 120–129.
[21] Y.Q. Sun, P. Pan, PRES—a practical flexible RBAC workflow system, in: Proceedings of the Seventh International Conference on Electronic Commerce, Xi'an, China, August 2005, pp. 653–658.
[22] L.Y. Wang, D. Wijesekera, S. Jajodia, A logic-based framework for attribute based access control, in: Proceedings of the FMSE'04, ACM Workshop on Formal Methods in Securing Engineering, US, 2004, pp. 45–55.
[23] Y. Zhong, B. Bhargava, M. Mahoui, Trustworthiness based authorization on WWW, in: Proceedings of IEEE Workshop on Security in Distributed Data Warehousing, New Orleans, 2001, pp. 1–6.