Comput Syst Sci & Eng (2008) 3: 193–206
© 2008 CRL Publishing Ltd

International Journal of
Computer Systems
Science & Engineering

Request-driven role mapping framework for secure interoperation in multi-domain environments

Ruixuan Li, Zhuo Tang, Zhengding Lu and Jinwei Hu

College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, Hubei, China
E-mail: rxli@hust.edu.cn, hust_tz@smail.hust.edu.cn, {zdlu, jwhu}@hust.edu.cn

This paper proposes a request-driven role mapping framework for secure interoperation in multi-domain environments. To support flexible policy expression and inter-domain policy mapping, we present a convenient and effective method to perform the privilege query in general hybrid role hierarchies for special external requests based on the minimal unique set (MUS). Role mappings are the basic approach for the interoperation among multiple individual domains. To describe the relationships between roles practically, role mappings are divided into three types: *I-mapping*, *A-mapping* and *IA-mapping*. These mappings denote the forms of the different role hierarchies respectively. Role mappings are the major causes for various types of conflicts and inconsistencies in multi-domains. This paper analyses the reasons for generating these conflicts and presents the algorithms to resolve them. Compared to other researches, this method can ensure that the external user requests will be satisfied and the local role hierarchies will be furthest preserved. Finally, the study of the instance for interoperation among the various offices of a county shows the validity of this role mapping framework.

Keywords: Role mapping, privilege query, conflict resolution, minimal unique set, secure interoperation, multi-domain.

1. INTRODUCTION

An important requirement of emerging systems is information sharing and interoperation with each other [1]. However, unauthorized accesses, particularly by insiders, constitute a major security problem for enterprise application systems in multi-domain environments [2]. How to manage the users' privileges effectively during the process of interoperation among numerous domains is a great challenge. Especially, when a user requests accessing the resources in another domain, which roles can be assigned to the user? To address these problems, this paper proposes a request-driven policy framework for secure interoperation in multi-domain environments.

In a typical multi-domain interoperating environment, the domains can be classified into two types: the external domain and the local domain. The external domain is responsible for mediating accesses to individual systems in local domains through maintaining global policies. When an external subject requests the local resources, we assume the process is established by the trust negotiation between the external and the local domains [3]. The external subject accesses the local domains through the authentication. However, the existing works on trust negotiation don't address the policy mapping [4]. The current software for the single sign-on (SSO) [5], such as the central authorization service (CAS) [6], does not support the fine-grained access control for the local domains. For example, the CAS can decide whether the user is capable of accessing the domains, but may not determine which special resources the user could access.

This paper proposes an interoperation framework using role mappings among multiple domains. In this framework, when the local domain receives a request from the external domain, it will search the role hierarchy for the suitable nodes to be mapped by the external. J. B. D. Joshi et al. [7] introduce the concept of unique activable set (UAS) to facilitate the analysis of hybrid hierarchies and simplify the process for determining the activation and permission-acquisition sets. A UAS is a set each element of which is a set of roles that can be activated in a single session by a specific user. In other words, a user assigned to the role in a hierarchy can only activate those set of roles that occur in a UAS associated with her [8]. Because the UAS set is the activable role set by a user which is assigned to the root role in a single session, for a special external request, the local UAS can not afford the optimum role set to be mapped by the external domain. As an extension of UAS, this paper proposes a new concept of the minimal unique set (MUS) to resolve this problem. Furthermore, we propose a convenient algorithm to compute the MUS for a general hybrid role hierarchy in the local domain. Through searching in the MUS, the external subjects can know which roles in the local domain are the most suitable ones for a special request. Through considering three different cases in this paper, we can provide the most minimal role set for an external request.

The role mappings describe the relationships between two roles in different domains. These relationships can be divided into three types: *I-hierarchy*, *A-hierarchy* and *IA-hierarchy* following the generalized temporal role-based access control (GTRBAC) [9]. These three relationships of the roles can be used to describe the relationships of role mappings in multi-domain environments as well. In this paper, we classify the role mappings into three types correspondingly: *I-mapping*, *A-mapping* and *IA-mapping*. These kinds of mappings describe the forms of the hierarchy between the mapping roles respectively. Through discussing how to establish the role mappings in multi-domain environments, we can resolve the conflicts for the role mappings between different domains effectively.

The proposed policy framework is based on the following assumptions: (i) All security domains in the interoperation environments employ the role-based access control (RBAC) polices. (ii) The external subjects trying to access the local domains have already passed the authentication, and different domains have undergone the trust negotiation.

The roles in the external and local domains need to support the role activation. Through trust negotiation, the users in the external domains can know which privileges in the local domain can be acquired by her [10]. The proposed request-driven policy framework generates a secure interoperation policy in two steps. Firstly, a multi-domain policy is composed by defining role mappings across multi-domains. Such mappings enable inter-domain information and resource sharing via the mapped roles. The framework allows the security administrators to map cross-domain roles based on the interoperability requirements of the collaborating domains. The role mappings are composed of the roles in the local domains and those in the external domains. For a special privilege request, it is to find out the appropriate roles for the external users.

In the second step, we can select the adaptive subject roles in the external domain to be mapped to the local domain and select an adaptive type of mappings to resolve the conflicts. The role mappings in multi-domains may not be consistent and may not satisfy the security constraints [11]. As discussed in Section 5, some of them arise from an inconsistent role mapping [12][13][14]. Separation of duties (SoD) [15] prevent two or more subjects from accessing an object that lies within their conflict of interests or disallow a subject from accessing conflicting objects or permissions. These mainly include: role-assignment violation, role-specific *SoD* violation, and user-specific *SoD* violation. Cyclic inheritance makes the role be able to access the resources which are not permitted previously [16].

The contributions of this paper are as follows:

1. This paper presents a new concept of the minimal unique set (MUS) to perform the privilege query in general hybrid role hierarchies for special external requests, and proposes an algorithm to compute the MUS. The MUS can be used to find out which roles should be mapped in the local domains. Furthermore, we analyze all cases of the special external requests. Especially, when the external request is not satisfied exactly, how to create a suitable new role in the local domain.

2. To solve the conflicts when establishing the interoperations among multi-domains, this paper classifies the role mappings into three types: *I-mapping*, *A-mapping* and *IA-mapping*. The different types reflect the different hierarchy relationships of the roles in the mappings. Through formalizing these three types of role mappings, we analyze the reasons of the conflicts frequently raised by role mappings between different domains, and present the resolution algorithms for each type of conflicts.

3. This paper also proposes a request-driven role mapping framework for establishing the interoperation in multi-domain environments. The approach of the role mappings based on user requests is suitable to the open distributed systems. It supports the flexible policy expression and inter-domain policy mapping.

The rest of the paper is organized as follows. Section 2 describes the basic concepts about interoperation policies. Especially, the new ways of role mappings are proposed. Section 3 introduces three cases for permission request in multi-domain environment. Section 4 presents the approach of minimal unique set (MUS) and the algorithms for request-driven privilege query in general hybrid role hierarchy. Section 5 describes how to establish role mappings among multi-domains, and focuses on the resolution of conflicts arising from role mappings. Section 6 illustrates the proposed request-driven role mapping framework and gives a running example. The related work is presented in Section 7, followed by the conclusions and future work in Section 8.

2. BASIC CONCEPTS

This section formally describes the basic concepts and syntaxes of our policy for the interoperation in multi-domains. To make our approach applicable to general situation, we do not make any assumption on the subjects, objects, or actions with respect to which authorization specifications should be stated. In order to illustrate our approach, we assume reference to some arbitrary, but fixed, set of roles R, objects O, actions A, and domains D. Depending on the application context and the policy to be enforced, objects could be files, relations, XML documents, classes [17]; actions represent the access modes, such as read, write, append and update; and domains are the abstract conceptions of the distributed applications.

Definition 1 (*Authorization Term*) An authorization term is a 2-tuple of the form: <object, action>, which is shortening as <*O*, *A*>. It is the basic form of the permission. The set of authorization terms is denoted as P. That is $P = \{<O, A>\}$.

Definition 2 (*Permission Set*) A permission set represents all permissions of some subject, which is the set of the authorization terms. We denote it as *PS*.

For example, we can describe all permissions of a role r_1 as: $PS\ (r_1) = \{<file_1, read>, <file_2, write>\}$. That is to say, the users associated with the role r_1 can read *file*$_1$ and write *file*$_2$. The denotation $PS(u)$ can also express the permission set of the user u. Obviously, if a role r is assigned to a user u, then $PS(u) \supseteq PS(r)$.

The denotation role (u) represents the role set that is assigned to user u. The BNF definition for permission sets is as follows.

$$PS = PS|PS \cup PS|PS \cap PS|PS - PS|SoD(PS,\ PS);$$

For example, set $PS_1(r) = \{<O_1,\ A_1>,\ <O_2,\ A_2>\}$, $PS_2(r) = \{<O_2, A_2>, <O_3,\ A_3>\}$, then

$$PS_1(r) \cup PS_2(r) = \{<O_1, A_1>, <O_2, A_2>, <O_3, A_3>\};$$
$$PS_1(r) \cap PS_2(r) = \{<O_2, A_2>\};$$
$$PS_1(r) - PS_2(r) = \{<O_1, A_1>\};$$

$SoD(PS_1(r), PS_2(r))$ returns $PS_1(r)$ or $PS_2(r)$, but it can not return the $PS_1(r)$ and $PS_2(r)$ simultaneity.

Role-based access control (RBAC) [18] is widely used for the security specification of most commercial applications. The RBAC model consists of the following four basic components: a set of users, a set of roles, a set of permissions, and a set of sessions. When a user logs in the system, the user establishes a session through activating a set of enabled roles that the user is entitled to activate at that time. If the activation request is satisfied, the user will obtain all the permissions associated with the requested roles [19].

The above operations form the basis of the interoperation policies. Our aim is to establish the mechanism to make the users be able to acquire the required privileges in other domains. In the proposed interoperation framework, the communications between two domains are mainly created by role mappings. The formalized definition about role mapping is as follows.

Definition 3 (*Role Mapping*) The role mapping is a 5-tuple: $<r_1, d_1, r_2, d_2, m>$,$r_1$is a role in domain d_1, and $r_2$is a role in domaind_2 respectively. In general, d_1 is the external domain, and d_2 is the local domain. The fifth parameter *m* is the mapping modes *I*, *A* or *IA*, which denotes the hierarchy relation of the two roles r_1 and r_2.The denotation *I* denotes the *I-hierarchy*, which means that the permissions belonging to the role r_2 also belong to the role r_1. The denotation*A* denotes the *A-hierarchy*, which means that if the role r_1 can be activated by a user, the role r_2 can also be activated by the same user. *IA* denotes the *IA-hierarchy*, which represents the relationship between the two roles r_1 and r_2 is *IA-hierarchy.* The parameter *m* divides the mapping into three types, which are called as *I-mapping*, *A-mapping* and *IA-mapping*. In the following section, we will discuss them in detail.

As Definition 3 represents, the mapping modes add a temporal constrain to the role mappings. In this way, we can formalize that a role in a special domain can be enabled or activated by a role in another domain.

As Figure 1 shows, there are two domains D_1 and D_2. The solid arrows represent the hierarchies and the broken arrows represent the mapping relationship between the two domains respectively. We can formalize this instance as two 5-tuple forms: $<r_{A2}, D_1, r_{B2}, D_2, A>$, $<r_{B1}, D_2, r_{A3}, D_1, I>$. According to GTRBAC [20], $<r_{A2}, D_1, r_{B2}, D_2, A>$ means that, if the role r_{A2} can be activated by a user, then the role r_{B2}can also be activated by the same user. And $<r_{B1}, D_2, r_{A3}, D_1, I>$ means that the permissions of the role r_{A3} in domain D_1 are inherited by the role r_{B1}.

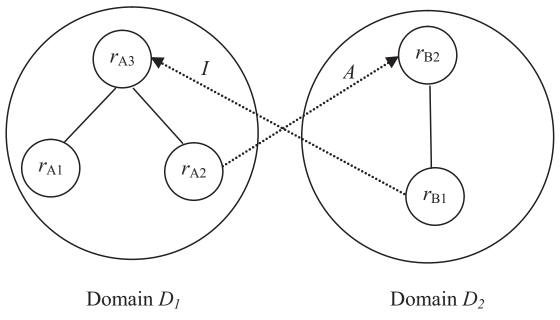


Figure 1 An example of role mapping between two domains.

In Table 1, we summarize the predicates of the role mappings in multi-domain environments with temporal constrains, where x, y denote the roles in different domains. The predicate $x \rightarrow_I y$ indicates that the users assigned to role x can acquire the permissions of the role y. The predicate $x \rightarrow_A y$ indicates that if a user can activate the role x, then it can also activate the role y. And the third predicate contains the two prior predicates. If $x \rightarrow_{IA} y$, it means $x \rightarrow_I y$, and $x \rightarrow_A y$.

Table 1 The predicates in multi-domain environment.

Predicate	**Notation**	**Formalization**
I-mapping	$x \rightarrow_I y$	$\forall p, p \subseteq PS(y) \rightarrow p \subseteq PS(x)$
A-mapping	$x \rightarrow_A y$	$\forall u, can\_activate(u, x) \rightarrow can\_activate(u, y)$
IA-mapping	$x \rightarrow_{IA} y$	$(x \rightarrow_{IA} y) \leftrightarrow (x \rightarrow_I y) \wedge (x \rightarrow_A y)$

3. PERMISSION REQUESTS IN MULTI-DOMAINS

When an external subject makes a request for accessing the objects in local domains, the request can be formalized as a temporal sequence: *REQ* (*req*(1), *req*(2),……*req*(n)). A special request of an external subject can be denoted as $REQ = \{(\{<O,\ A>\})\}$. The element of the set, $req(i) = \{<O,\ A>\}$, means that the subject wants the permission set $\{<O,\ A>\}$ in the local domain. For a special access request, the local domain should afford relevant privileges to the user as far as possible.

Firstly, we consider a single request $req(i)$, and assume that the $req(i)$ includes a set of request for the permissions. That is, $req(i) = \{<O,\ A>\}$.

We proceed to discuss how the local domain finds or creates roles to satisfy the request. In general, for an external access request and the requested permission set $req(i)$, the following cases may arise in a local domain:

(i) There is a set of roles in the local domain that can exactly satisfy the request $req(i)$ of the external domain. These roles may be hierarchically related.

(ii) There is a set of roles in the local domain, whose subset can exactly satisfy the request $req(i)$ of the external domain. That is, the permission set of the role set contains the permissions requested by the $req(i)$.

(iii) There is a set of roles in the local domain can be supported for the external domain, through which only a subset of permissions for $req(i)$ can be acquired.

For a special request, we can create different types of mappings from the external entity to the role set in local domain for different types of above cases. Figure 2 illustrates a simple example for the first case – Case (i). The solid line represents the *I-hierarchy* while the broken line represents the *A-hierarchy* between two roles. In the local domain, we assume that $req(i) = \{p_1\}$. p_1 equals to $PS(r_1)$ exactly. It is obvious that this request can be satisfied if the external entity is mapped to the role r_1. It is the first scene which we discuss above. But if $req(i) = \{p_1,\ p_2,\ p_3\}$, and in the local domain, $p_1 = PS(r_1)$, $p_2 = PS(r_2)$, $p_3 = PS(r_3)$ exactly, we have to establish a one-to-many mapping. That is to say, the roles $\{r_1, r_2, r_3\}$ can be mapped to the external entity. If the senior role r_4 is not assigned with any other permission, it is the I-senior role for the roles r_1 and r_2. For this case, the permissions of r_4 are $\{p_1,p_2\}$. If the role r_5 has no other permissions to be assigned, the roles $\{r_5\}$ will be selected to be mapped to the external entity. Otherwise, the mapped roles are $\{r_4, r_3\}$.

It can be easily seen that the pivotal step is to find the most adaptive roles whose privileges matching to the external request exactly. In general, to reduce the number of the role mappings, the criterion of the "most adaptive" is the "smallest in magnitude". In other words, for a special external privilege request, we try to find the smallest role set in the local role hierarchy to satisfy the request. In a special role hierarchy, the elements in the MUS are the minimal unique role set for a set of privileges. So, for the first case, we can acquire the most adaptive roles for the external request through searching in the MUS set. Section 4.2 proposes an algorithm to compute the MUS set.

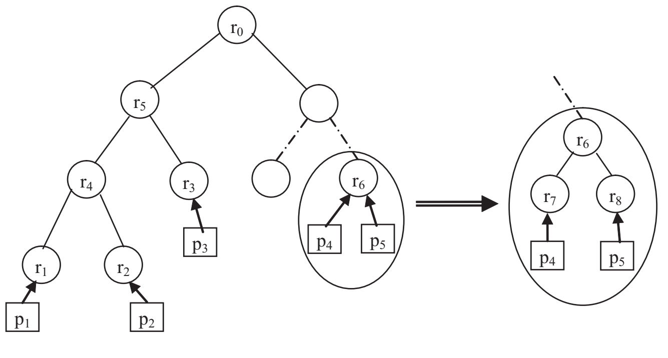


Figure 2 An example for the first case - Case (i).

The second case (Case (ii)) is to indicate that the $req(i)$ is a subset of permissions which can be acquired by the external domain, but we can not find a set of role whose permissions equal to the $req(i)$ exactly. We need to do some additional transformation, such as creating new roles, to get a role set that exactly provides the required privileges to the external entities. In Figure 2, if $req(i) = \{p_1,\ p_4\}$, because $PS(r_6) = \{p_4, p_5\}$, role r_6 can not be mapped to the external entity because that will allow the external entity to acquire an extra permission p_5. One possible solution is to split the role r_6. Two new roles r_7 and r_8 will be created as the junior roles of r_6, where r_7 is assigned with permission p_4, and r_8 is assigned with permission p_5. So the new role set $\{r_1, r_7\}$ can satisfy the request $req(i)$.

In the second case, for the roles containing the required permissions, at least one of which contains one or more extra permissions. These roles can be called as "original role". We have two transformations to satisfy the requirements in the second case.

1. Split the original role. Through splitting the original role to several roles, one of them can provide the external required permissions, while the others provide the extra permissions.

2. Augment local hierarchy. We can create a new *I-junior* role for the original role. Then the new one is assigned with external required permissions.

The third case (Case (iii)) is more complex than the other two. Only part of the requested permissions can be satisfied. For the request $req(i)$, the method is to find out the maximal permission set which can be provided by the local hierarchy firstly. Then, we can resolve the similar problem on the maximal permission set according to the methods in the first and second cases.

For the temporal sequence: $REQ(req(1), req(2), \ldots, req(\mathrm{n}))$, we propose a method to optimize the query efficiency through using the permission request buffer. This buffer is maintained by each individual domain. The permission request buffer is like a hash map, which stores the privileges and the corresponding role sets. When initializing the multi-domain environment, each buffer will be cleaned. Once a user's request is processed, the requested permissions and the corresponding roles for the request will be set to the buffer according to a hash function. When an external subject requests the same permissions, the local domain can query the hash map to get the corresponding roles without searching in the whole role hierarchy. The structure of the request buffer is shown as Table 2.

In the structure of the request buffer, the column "Times" denotes the times of the corresponding permissions being accessed. Because the size of the buffer is limited, we must remove some items in the buffer in order to allow the new request records to be inserted. Like the page model in the memory management [21], the least frequently accessed privilege items will be

Table 2 The structure of the request buffer.

Requests	Requested permissions	Roles	Times
$req(1)$	p_1,p_2,p_3	r_1, r_2, r_3	2
$req(2)$	p_1,p_4	r_1, r_7	1
$req(3)$	p_1	r_1	5
…	…	…	…

removed from the buffer. When the local domain receives an external request, it will search in the request buffer first. If the buffer misses, it will search in the original local role hierarchy. Algorithm 1 shows how to process the permission requests.

Algorithm 1. Processing the permission requests.
Input: permission request *req*(i), and the request buffer.
Output: the roles match the permission requests.

```
processPermissionRequest()
1.  search in the request buffer;
2.  if there is a record matches the request then
3.     parameter time pluses 1;
4.     return the corresponding roles;
5.  else
6.     search in the original local role hierarchy;
7.     if there are roles whose permissions match the external
       request then
8.        if the request buffer has free space then
9.           add the new request and its corresponding role set
             to the buffer;
10.          else replace the item which is the least frequently
             accessed in the buffer with the new
             request and its corresponding role set, and the
             parameter time is set to 1;
11.       end if
12.       return the matched roles;
13.    else
14.       return no matched roles.
15.    end if
16. end if
```

4. REQUEST-DRIVEN PRIVILEGE QUERY

In this section, we will propose a method for the privilege query in general hybrid role hierarchies.

4.1 Minimal unique set (MUS)

In an arbitrary hybrid hierarchy, maintaining permission acquisition and role activation semantics becomes a quite challenging job. S. M. Chandran and J. B. D. Joshi [22] introduce the concept of uniquely activable set (UAS) to facilitate the analysis of hybrid hierarchies and simplify the process of determining the activation and permission acquisition sets. Because each element of UAS is a set of roles that can only be activated in a single session by a specific user which is assigned only to the root role, it can not satisfy the interoperation requirements in a multi-domain environment. For secure interoperation in multi-domains, we need to find an adaptive role set for a special external request. These roles will be mapped to the external entities. Hence, we can not restrict the roles to be activated in a single session and only by a single user assigned the root role.

To find out all minimal role set whose privileges are unique in a hybrid hierarchy, this paper proposes a new concept of the minimal unique set (MUS). MUS is the set of role set. For arbitrary privileges that can be provided by a role hierarchy, the minimal role set satisfying the privilege request must be included in the set MUS. In the following sections, we will present three theorems to describe the proposed methods.

The MUS can be seen as an extension of the UAS defined in [23].

Definition 4 (*Minimal Unique Set*) Let $H = (R, F)$ be a rooted hybrid hierarchy. Then, $MUS(H) = \{Y_1, Y_2, \ldots, Y_m\}$, where $\emptyset \subset Y_i \subseteq R$ for each $i \in \{1, 2, \ldots, m\}$, is the Minimal Unique Set (MUS) of role sets of H if the following conditions hold:

- $\forall i, j \in \{1, 2, \ldots, m\}$ and $i \neq j$, $PS(Y_i) \neq PS(Y_j)$, and
- $\forall Z \subseteq Rs.t.Z \notin MUS(H)$, if $PS(Y) = PS(Z)$ for a $Y \in MUS(H)$, then $(|Y| < |Z|)$; where $|A|$ is the cardinality of set A.

4.2 MUS computing algorithms

Following the above principles, we present an algorithm *ComputeMUS* to find out a role set that satisfies the requested permission set *req*(i). In this algorithm, the breadth-first search in the local role hierarchy ensures that the order of the local role sets, which is denoted as R, is from up to down. [24] proposes an algorithm to find role set to satisfy the external request. But it just aims at a special user in a single session, and does not consider how the roles are mapped to the external entities. Obviously, this method can not meet the requirements of the interoperation in multi-domains.

For an arbitrary general role hierarchy H, we can split all *I-hierarchy* (*IH*) from the hierarchy H. Each *I-hierarchy* contains part of roles of the hierarchy H. Based on these sub-hierarchy, we can compute the MUS for the general hierarchy Has algorithm *ComputeMUS*.

Firstly, we introduce some operations:

$$N_1 \otimes N_2 = \{\{x_1 \cup x_2\} | x_1 \in N_1 \text{ and } x_2 \in N_2\}$$

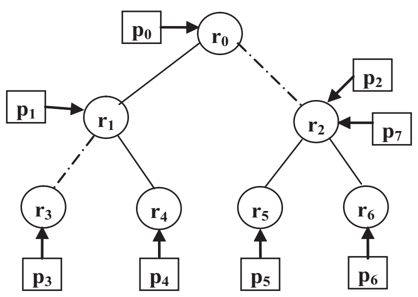
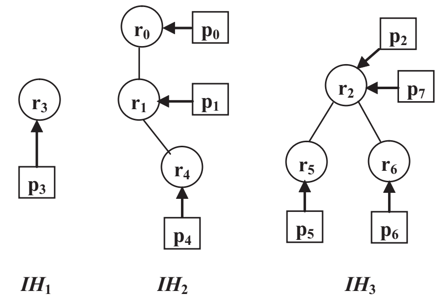
Analogously, assuming x_i is the arbitrary element of the set N_i, $x_i$can be an empty set $\emptyset$.

$$N_1 \otimes N_2 \otimes \ldots \otimes N_m = \{\{x_1 \cup x_2 \cup \ldots \cup x_2\} | x_1 \in N_1, x_2 \in N_2 \ldots x_m \in N_m\}$$

Assuming $M = \{N_1, N_2, \ldots, N_m\}$, we have:

$$\Theta M = N_1 \otimes N_2 \otimes \ldots \otimes N_m$$

We propose the algorithms to compute the MUS for an arbitrary hybrid role hierarchy H, as shown in Algorithm 2, Algorithm 3 and Algorithm 4. The computing approach is enlightened by the algorithm proposed in [23]. In the algorithms, we firstly decompose the hybrid role hierarchy into several pure *I-hierarchies*. As an example of the hybrid role hierarchy shown in Figure 3, we can split it into three *I-hierarchies*: $\{IH_1, IH_2, IH_3\}$ as shown in Figure 4. Through producing a set of roles R' by selecting arbitrary roles where there is no *I-hierarchy* relation between them in every hierarchy IH_i, the power set of the R' is added to the MUS of the hierarchy H. For instance, we select $\{r_0, r_2, r_3\}$ in Figure 4 as R', then the set

 p0 r0 p1 r1 p2 r2 p7 r3 r4 r5 r6 p3 p4 p5 p6	 r0 p0 p2 r3 r1 p1 r2 p7 r4 r5 r6 p3 p4 p5 p6 IH1 IH2 IH3
Fig. 3. An example of the hybrid role hierarchy H	**Fig. 4.** Decomposing the hybrid role hierarchy into *I-hierarchies*

{{r0},{r2}, {r3},{r0,r2},{r0,r3},{r2,r3},{r0,r2,r3}} will be added to the MUS. Once we complete selecting all roles in each hierarchy IH_i, the MUS is formed. The algorithms are as follows.

Algorithm 2. Computing the MUS for an arbitrary hybrid role hierarchy H.
Input: an arbitrary hybrid role hierarchy H.
Output: the MUS of the role hierarchy H.

```
ComputeMUS (H)
1.  initialize MUS =∅;
2.  IH ←I-hierachyGeneration(H);
        // IH={IH_1, IH_2,...... IH_m}, Algorithm 3
3.  IH'←NonInheritSetGeneration(IH); // Algorithm 4
4.   foreach IH"∈2^{IH'} do
5.     if (|IH"|==1) then
6.       foreach r' ∈IH" do
7.         MUS = MUS ∪ {r'};
8.     else
9.         MUS = MUS ∪ ΘIH";
10.   return MUS.
```

Algorithm 3. Computing all *I-hierarchies* for the hierarchy H.
Input: an arbitrary hybrid role hierarchy H.
Output: *IH*: all *I-hierarchies* for the hierarchy H.

```
I-hierachyGeneration(H)
1.   initialize TempR = ∅;
         //the list for the role that has been searched
2.   initialize IH = ∅;
3.   R = Roles(H);
         // R is the list of all roles in the hierarchy H
4.   while R ≠ ∅ do
5.     from r start DFS Search under I-hierachy, the roles
         passed are added to the set TempR;
6.     when the DFS Search completed, produce a sub- I-
         hierachy: IH_i
7.     IH = IH ∪ IH_i;
8.     R=R—TempR;
9.   end while
10.  return IH.
```

Algorithm 4. Computing all role set without inherit hierarchy for the *I-hierarchy IH*.
Input: *I-hierarchy IH*.
Output: *IH'*: all role set without inherit hierarchy for the *I-hierarchy IH*.

```
NonInheritSetGeneration(IH)
1.  initialize IH' as the power set of the roles in the IH;
2.   foreach R' ∈IH' do
3.     if there is a inherit path between the arbitrary roles in
         the set R' then
4.       IH'= IH'—R';
5.   return IH'.
```

4.3 Correctness and complexity analysis

As the MUS set is created based on the above process, we have the following theorem.

Theorem 1 The set of the role set created by Algorithm 2, Algorithm 3 and Algorithm 4 is the MUS for the hybrid role hierarchy.

Proof. Without loss of generality, we assume that the permissions belonging to the different roles in the same *I-hierarchy* are unique. That is to say, the direct assigned permissions for different roles are acquiescently different. So, when we produce a set of role sets R' by selecting arbitrary role in every *I-hierarchy* IH_i, the permissions of all subset of the power set of R' is different. It satisfies the first condition of the Definition 4.

Because the elements in R' are the role sets whose roles come from different *I-hierarchies* (it is denoted as Y_i in Definition 4), there are no "inclusion" relationship among permission sets of each role in *Yi*. Assuming Y_i $=\{r_i, r_j\}$, if $PS(Y_i)= PS(Y_j)$ and $|Y_i|>|Y_j|$, assume Y_j $=\{r_j\}$, then $PS(r_i) \subseteq PS(r_j)$. In the general case, if two roles belong to different *I-hierarchies* respectively or there are no "subordinate" relationship between the two roles in the same *I-hierarchies* (such as r_5 and r_6), their privileges can not include each other. So, it satisfies the second condition in the Definition 4. Hence, the set of the role sets produced by algorithm *ComputeMUS* is the *MUS* for the hybrid role hierarchy.

Example 1 Consider the hybrid hierarchy shown in Figure 3. If we visit the tree through using the breadth first search, then

$R = \{r_0, r_1, r_2, r_3, r_4, r_5, r_6\}$. The *MUS* for the hierarchy is as follows:

{{r$_0$}, {r$_1$}, {r$_2$}, {r$_3$}, {r$_4$}, {r$_5$}, {r$_6$}, {r$_0$, r$_2$}, {r$_0$, r$_5$}, {r$_0$, r$_6$}, {r$_1$, r$_2$}, {r$_1$, r$_5$}, {r$_1$, r$_6$}, {r$_2$, r$_4$}, {r$_4$, r$_5$}, {r$_4$, r$_6$},{r$_0$, r$_3$}, {r$_1$, r$_3$}, {r$_3$, r$_4$},{r$_2$, r$_3$}, {r$_3$, r$_5$}, {r$_3$, r$_6$},{r$_5$, r$_6$}, {r$_0$, r$_2$, r$_3$}, {r$_0$, r$_3$, r$_5$}, {r$_0$, r$_3$, r$_6$}, {r$_1$, r$_2$, r$_3$}, {r$_1$, r$_3$, r$_5$}, {r$_1$, r$_3$, r$_6$}, {r$_2$, r$_3$, r$_4$}, {r$_3$, r$_4$, r$_5$}, {r$_3$, r$_4$, r$_6$},{r$_0$, r$_5$, r$_6$},{r$_1$, r$_5$, r$_6$},{r$_4$, r$_5$, r$_6$},{r$_3$, r$_5$, r$_6$}}

The permission sets of each role set in the *MUS* are:

PS({r$_0$})={p_0, p_1, p_4}, *PS*({r$_1$})={p_1, p_4}, *PS*({r$_2$})={p_2, p_5, p_6, p_7}, *PS*({r$_3$})={p_3}, *PS*({r$_4$})={p_4}, *PS*({r$_5$})={p_5}, *PS*({r$_6$})={p_6}, *PS*({r$_0$, r$_2$})={p_0, p_1,p_2, p_4, p_5, p_6, p_7}, *PS*({r$_0$, r$_5$})={p_0, p_1, p_4,p_5}, *PS*({r$_0$, r$_6$})={p_0, p_1, p_4,p_6}, *PS*({r$_1$, r$_2$})={p_1,p_2,p_4,p_5, p_6, p_7}, *PS*({r$_1$, r$_5$})={p_1, p_4,p_5}, *PS*({r$_1$, r$_6$})={p_1, p_4,p_6}, *PS*({r$_2$, r$_4$})={p_2,p_4, p_5, p_6, p_7}, *PS*({r$_4$, r$_5$})={p_4,p_5}, *PS*({r$_4$, r$_6$})={p_4,p_6}, *PS*({r$_0$, r$_3$})={p_0, p_1,p_3, p_4}, *PS*({r$_1$, r$_3$})={p_1,p_3, p_4}, *PS*({r$_3$, r$_4$})={p_3, p_4}, *PS*({r$_2$, r$_3$})={p_2,p_3, p_5, p_6, p_7}, *PS*({r$_3$, r$_5$})={p_3, p_5}, *PS*({r$_3$, r$_6$})={p_3, p_6}, *PS*({r$_5$, r$_6$})={p_5, p_6}, *PS*({r$_0$, r$_2$, r$_3$})={p_0, p_1,p_2,p_3,p_4, p_5, p_6, p_7}, *PS*({r$_0$, r$_3$, r$_5$})={p_0, p_1,p_3,p_4, p_5}, *PS*({r$_0$, r$_3$, r$_6$})={p_0, p_1,p_3,p_4, p_6}, *PS*({r$_1$, r$_2$, r$_3$})={p_1,p_2,p_3,p_4, p_5, p_6, p_7}, *PS*({r$_1$, r$_3$, r$_5$})={p_1,p_3, p_4,p_5}, *PS*({r$_1$, r$_3$, r$_6$})={p_1, p_3, p_4,p_6}, *PS*({r$_2$, r$_3$, r$_4$})={p_2,p_3,p_4, p_5, p_6, p_7}, *PS*({r$_3$, r$_4$, r$_5$})={p_3, p_4, p_5}, *PS*({r$_3$, r$_4$, r$_5$})={p_3, p_4, p_6}, *PS*({r$_0$, r$_5$, r$_6$})={p_0,p_1, p_4, p_5,p_6}, *PS*({r$_1$, r$_5$, r$_6$})={p_1, p_4, p_5,p_6}, *PS*({r$_4$, r$_5$, r$_6$})={p_4, p_5,p_6}, *PS*({r$_3$, r$_5$, r$_6$})={p_3, p_5,p_6}.

Consider the main operation Θ in Algorithm 2 *computeMUS*: $\Theta M = N_1 \otimes N_2 \otimes \ldots\ldots \otimes N_m = \{\{x_1 \cup x_2 \cup \ldots\ldots \cup x_2\} \mid x_1 \in N_1, x_2 \in N_2 \ldots\ldots x_m \in N_m\}$, where x_i is the arbitrary element of the set N_i, and x_i can be an empty set. That is to say, the denotation Θ is to compute the power set of the sets whose elements are from each set N_i respectively. Hence, the complexity of this algorithm is O(2^m), where *m* is the number of *I-hierarchies* which is generated by the Algorithm 3 *I-hierarchyGeneration*. Apparently, the complexity is NP-complete.

In the best situation, the hybrid hierarchy does not contain any *A-inherit* relations. That is to say, the role hierarchy in the local domain is *I-hierarchy*. The *IH* only contains one sub-tree. The complexity of this algorithm is up to the process of the depth-first search (DFS), and it can be considered as polynomial: O(n^2), where *n* is the number of the roles in the local domain.

In the worst case, the hybrid hierarchy does not contain any *I-inherit* relations. The number of *I-hierarchies* which is generated by the Algorithm 3 *I-hierarchyGeneration* is equal to the number of the roles. So, the complexity of this algorithm is O(2^n), where *n* is the number of the roles in the local domain.

In a word, compared to the algorithm of computing the UAS of a hybrid hierarchy [21], the Algorithm 2 *ComputeMUS* in this paper is also non-polynomial solutions. To optimize this algorithm and propose polynomial solutions, we need further research considerations.

The MUS supports the first case (Case (i)) of permission request above commendably. If *req*(i)={p_1, p_4,p_6}, through searching the permission sets of each role set in the MUS, we can find the minimal role set for supporting these permissions: {r_1, r_6}. If we can not find a set of local roles whose permissions meet the request *req*(i) exactly, the possible solutions are to split the local role or create the new roles.

Theorem 2 *rm For a special hybrid hierarchy, MUS is the set of role sets that contain no I-Inherit path.*

Proof. According to the computing process in the algorithm *ComputeMUS*, we can reach this conclusion easily.

Theorem 3 For a given permission set P, if a role hierarchy $H = (R, F)$ can provide all privileges of the permission set P, then:

$\exists Z \subseteq R$, *if* $Z \notin MUS(H)$ and $PS\ (Z) = P$, then $\exists R' \in MUS\ (H), PS\ (R') = P$, and$R \subseteq Z$

That is to say, for arbitrary privilege provided in a role hierarchy, the minimal role set which can satisfy these privileges must be included in the set *MUS*.

Proof. For an arbitrary given permission set P, if a role hierarchy H=(R, F) can provide all privileges of the permission set P, we assume that there is a role set Z can satisfy: $Z \subseteq R, PS\ (Z) = P, Z \notin MUS(H)$. Because $Z \notin MUS(H)$, according to Theorem 2, we can get that there is an *I-inherit* path $\{r_{i1}, r_{i2} \ldots\ldots r_{in}\}$ in the roles of Z. That is to say, the role r_{i1} inherits the privileges of the roles $r_{i2} \ldots\ldots r_{in}$. $PS\ (r_{i1}) = PS\ (\{r_{i1}, r_{i2} \ldots\ldots r_{in}\})$. So, we can replace the path $r_{i1}, r_{i2} \ldots\ldots r_{in}$ with the role r_{i1} in the role set Z. In this step, the permissions provided by $Z$are unchanged. Through dealing with all *I-inherit* paths in the set Z, we can get a new set R', and apparently, $PS\ (R') = P$ and $R \subseteq Z$.

For the second case (Case (ii)) of the external request, we can not find an exact role set in MUS to satisfy the requested permissions. In Example 1, assuming the request *req*(i)={p_1, p_4,p_7}, there are no local roles containing the permission p_7 merely. The method in this paper is to find an appropriate role to be decomposed. The result of the decomposition will produce a new role which contains the adaptive permissions for the external request. For the instance shown in Figure 5, the role set in the *MUS* whose permissions satisfy the request are: {r_0, r_2}, {r_1, r$_2$}, {r_0, r_2, r_3}, {r_1, r_2, r_3}. Because the set {r_1, r_2} is assigned the least permissions, and *PS*(r_1)={p_1, p_4}, *PS*({r_2})={p_2, p_5, p_6, p_7}, *PS*(r_1) $\subseteq$ *req*(i), *PS*(r_2) $\not\subseteq$*req*(i), the role r_2 will be selected to decompose. For the stabilization of the *MUS*, we prefer to use the selected role as the direct senior role of the new role in each adjustment of the local hierarchy. In this instance, we can create a new role r', *PS*(r') = {p_7}. Because the external request is to activate some local roles to acquire the appropriate permissions, the relationship between roles $r_2$and $r'$is *I-hierarchy*.

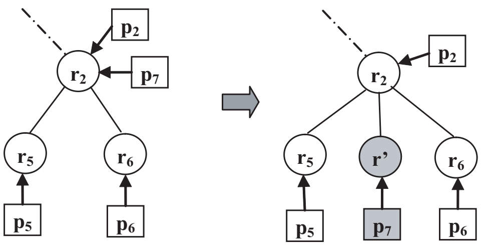


Figure 5 An instance for role decomposition.

The resolution for the third case (Case (iii)) is similar to the second. we can split *req*(i) into two subsets *req1*(i) and *req2*(i) such that *req1*(i) relates to the second case and *req2*(i) refers to the permission set that are not available through any existing role.

5. CONFLICT RESOLUTION FOR ROLE MAPPINGS

When the privilege query is completed in the local domain for a special external request, the following step is to create the role mappings between two individual domains. This section will discuss which type of mappings is suitable for a special external request. When the requisite roles in the local domain satisfy the external request, the mappings should be established from the external roles of the subject to these local roles.

5.1 Security requirements in role mappings

In Figure 6, the hierarchy of the local domain is the same as the Example 1. Suppose the *req*(i) of the external user u_1 is $\{p_1, p_4, p_6\}$. Through searching the MUS of the local domain, we know that the roles $\{r_1, r_6\}$ will satisfy the request and be mapped by the external domain. In the external domain, the role r_c that has two *A-junior* roles is assigned to the user u_1. We have to select a role from $r_c$or its juniors to be mapped to the local roles $\{r_1, r_6\}$.

When the roles in the local domain satisfying the external request are ready for mapping, the following problems will arise.

1. Which roles associated with the request subject in the external domain should be mapped to the roles in the local domain?

2. Which types of mapping should be selected for these roles? That is to say, which hierarchy is appropriate for the two roles between the external and local domains?

As we know, secure interoperation requires enforcement of the following two principles: autonomy principle and security principle [25]. In this paper, the selection of the mapping type and the original roles in the external domain is up to these principles.

The mapping policies in multi-domains may be inconsistent and not satisfy the security constraints of collaborating domains. Separation of duties (*SoD*) prevents two or more subjects from accessing an object that lies within their conflict of interests or disallows a subject from accessing conflicting objects or permissions [26]. For example, the same managers cannot authorize payments and sign the payment checks [16]. Violations of *SoD* constraints may occur in an interoperation policy because of the interplay of various policy constraints across domains.

There are various conflicts existing in role mapping among multiple domains, such as role-assignment violation, role-specific *SoD* violation, user-specific *SoD* violation, and cyclic inheritance. These conflicts can be resolved by removing some of the mapping links specified in the role mapping as presented in [25]. But this method may lead the request of the external user to be not satisfied. Furthermore, it is difficult to determine which mappings should be reserved and which should be removed. In this paper, we propose a new method for selecting the mapping type and the original roles in the external domain that can avoid these conflicts and inconsistencies.

5.2 Violation of role-specific SoD

Role mappings cause violations of role-specific *SoD* constraints of domain k if it allows a user to simultaneously access any two conflicting roles r_i and r_j of domain k in the same session or in concurrent sessions. For example, Figure 7 shows that the roles r_e and r_f are the conflicting roles in the external domain. That is to say, these two roles can not be accessed by a same user concurrently. As the role r_c is assigned to the user u_1, the user u_1 can activate the roles r_e and r_f, but cannot activate them concurrently. We assume that user u_1 activates the roles r_c and r_e concurrently, that is to say, the user u_1 can inherit all permissions of the role r_e: $PS\ (u_1) \supseteq PS\ (r_c)$, $PS\ (u_1) \supseteq PS\ (r_e)$. As mentioned above, if the user u_1 requests the permissions $\{p_1,\ p_4, p_6\}$ in the local domain, the mapping must be created between the roles $\{r_c, r_e, r_f\}$ and the local roles $\{r_1, r_6\}$. Now, consider the *I-mapping* $< r_c, E, r_1, L, I>$. The role r_c can inherit all permissions of the role $r_1$in domain L through this mapping: $PS\ (u_1) \supseteq PS\ (r_1) \supseteq PS\ (r_4)$. If there is already an *I-mapping* from role r_4 to role r_f, the role r_c can inherit all permissions of the role r_f through r_4. $PS\ (u_1) \supseteq PS\ (r_c) \supseteq PS\ (r_4) \supseteq PS\ (r_f)$. So, $PS\ (u_1) \supseteq (PS\ (r_e) \cup PS\ (r_f))$. Thus the user u_1 can acquire the permissions of the role r_f, which has the *SoD* conflict with the role r_e.

Supposing denotations U and R represent the set of users and roles in a domain respectively, the violation of role-specific *SoD* can be formalized as: $\{\exists u \in U,\ r_1, r_2 \in R,\ SoD(r_1, r_2) | PS(u) \supseteq PS(r_1) \cup PS(r_2)\ \}$

In this case, if the user u_1 requests the permissions $\{p_1, p_4, p_6\}$, and the user u_1 activates the roles r_c and r_e concurrently, we have two methods to resolve this problem.

1. We can remove the *I-mapping* $<r_c, E, r_1, L, I>$ and establish the mapping $<r_f, E, r_1, L, A>$. In this way, there is an *A-inherit* path from the role r_c to r_1 through the role r_1. So the user u_1 can activate the role r_1. For the same reason, we can also establish the mappings $<r_e, E, r_1, L, A>$ or $<r_c, E, r_1, L, A>$. With the constraint that the user u_1 can not activate the role r_e and r_1 concurrently, the request of the user u_1 can be satisfied.

2. We can transform the existing *I-mapping* $<r_4, L, r_f, E, I>$ to $< r_4, L, r_f, E, A>$. In this way, the role $r_4$does not inherit the permissions of role r_f any longer. So, if the *I-mapping* $< r_c, E, r_1, L, I>$ is established, the user u_1 can not acquire the permissions of the role r_f. Therefore, the role specific violation of *SoD* is resolved.

The reason for violations of the role-specific *SoD* is that a user simultaneously accesses any two conflicting roles r_i and $r_j$of the domain in the same session or in concurrent sessions. If the role mapping between different domains is the pivotal cause of this violation, the solution is to forbid these roles to be activated by the user concurrently through *A-mapping*. In general, the

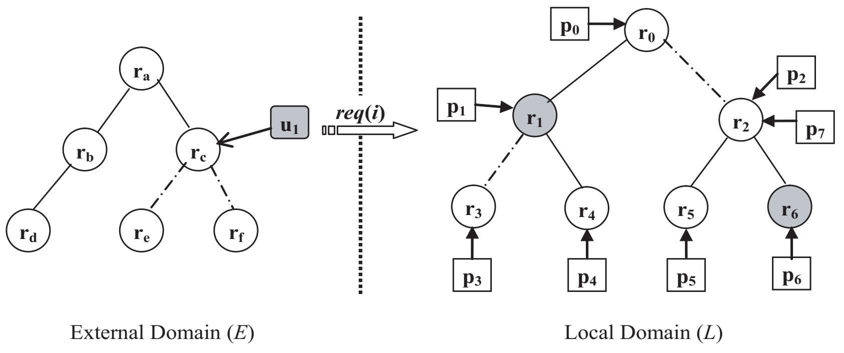


Figure 6 An example of the role mapping between two domains.

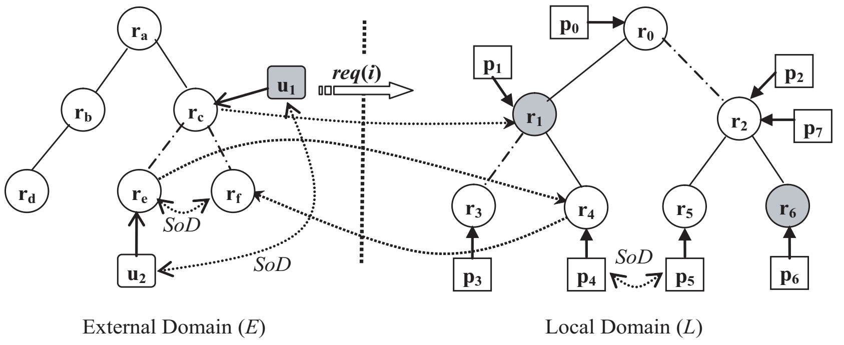


Figure 7 An example of *SoD* between two domains.

following algorithm can be used to detect and resolve this type of conflict, as shown in Algorithm 5.

Algorithm 5. The resolution for violation of role-specific *SoD*.

```
Input: The role hierarchies with role mappings and authorization assignment in a special multi-domain environment HS.
Output: RM: The role mappings without violation of role-specific SoD.
NonVRSGeneration(HS)
1.  foreach user $u \in U$ do
2.    get all roles associated with the user $u$: $Roles(u)$;
3.    if there are SoD constrains in $Roles(u)$ then
4.      select the conflicting roles into a set SoD_R;
5.    else return;
6.     foreach role $r \in$ SoD_R do
7.       find all I-inherit paths from $u$ to $r$;
8.       if there is an I-mapping in the I-inherit path then
9.           transform the I-mapping into the relevant A-mapping;
10.          modify the start role of the mapping with the user's direct assigned role or the A-inherit junior role;
11.       end if
12.      end foreach
13.     forbid the user $u$ to activate the roles in SoD_R concurrently;
14.    end foreach
15.   form these A-mappings into the role mappings RM;
16.   return RM.
```

5.3 Violation of user-specific SoD

Role mappings cause violation of user-specific *SoD* constraint of domain *k* if it allows any two *SoD* users to hold the same role *r* in the same session or in concurrent sessions. For example, as Figure 7 shows, the user u_1 and u_2 are a pair of *SoD* users. The role sets assigned to them should not have a common set. As mentioned above, if the user u_1 requests the permission $\{p_1, p_4, p_6\}$, the mappings must be established between the roles $\{r_c, r_e, r_f\}$ and the local roles $\{r_1, r_6\}$. We assume that there is already a mapping $<r_e, E, r_4, L, I>$ between the two domains before the mappings are established. Now, consider the *I-mapping* $<r_c, E, r_1, L, I>$. Through this mapping, the user u_1, which is assigned the role r_c, can acquire all the permissions of r_1: $PS(u_1) \supseteq PS(r_c) \supseteq PS(r_1)$. As r_1 is the senior role of r_4, the permission set of the role r_4 is inherited by the role r_1. That is to say, the user u_1 can acquire the permissions of the role r_4, $PS(u_1) \supseteq PS(r_4)$. On the other hand, there is a mapping from role r_e to role r_4. As the role r_e is assigned to the user u_2, the user u_2 can also access the role r_4. So, the user u_2 can also acquire the permissions of the role r_4 in the same way: $PS(u_2) \supseteq PS(r_4)$. Thus, the conflicting users, u_1 and u_2, can access the same permissions of the role r_4. This is a representative case of violation of the user-specific *SoD*.

Supposing denotations *U* and *R* represent the set of the users and roles in a domain respectively, the violation of the user-specific *SoD* can be formalized as: $\{\exists u_1, u_2 \in U, \exists r_1, r_2 \in R|\ SoD(u_1, u_2) \wedge r_1 \in role(u_1) \wedge r_2 \in role(u_2) \wedge Containned(r, r_1) \wedge Containned(r, r_2)\}$.

In this case, if the user u_1 and u_2 are a pair of *SoD* users in the external domain and the user u_1 requests the permission $\{p_1,$

p_4,p_6}, we can transform the existing *I-mapping* <r_e, E, r_4, L, I> to <r_e, E, r_4, L, A> firstly. In this way, the role r_e does not inherit the permissions of role r_4 any longer. Than we can add the *A-mapping* from the role r_c or its juniors to r_1. For instance, the *A-mapping* <r_c, E, r_1, L, A>. Because the user u_1 and $u_2$can not acquire the same permissions of the role r_4, the following constraint must be enforced:

$$\{SoD(u_1, u_2) \wedge \text{can\_activate}(u_1, r_1, t_1)\wedge \text{can\_activate}(u_2, r_4, t_2) \wedge (t_1 \wedge t_2 = \emptyset)\}$$

When violation of the user-specific *SoD* occurs, the appropriate way is to establish the *A-mapping* between the two domains, and the conflicting users are prohibited from activating the same junior roles in the *A-hierarchy* path concurrently. In general, the following algorithm can be used to detect and resolve the violations of the user-specific *SoD*, as shown in Algorithm 6.

Algorithm 6. The resolution for violation of user-specific *SoD*.

```
Input: The role hierarchies with role mappings and authorization assignment in a special multi-domain environment HS.
Output: RM: The role mappings without violation of user-specific SoD.
NonVUSGeneration(HS)
1.  foreach role $r \in R$ do
2.      get all users associated with the role $r$: $Users(r)$;
3.      if there are SoD constrains in $Users(r)$ then
4.          select the conflicting users into a set SoD_U;
5.      else return;
6.      foreach user $u \in$ SoD_U do
7.          find all I-inherit paths from $u$ to $r$;
8.          if there is an I-mapping in the I-inherit path then
9.              transform this I-mapping into the relevant A-mapping;
10.             modify the start role of the mapping with the user's direct assigned role or the A-inherit junior role;
11.         end if
12.     end foreach
13.     forbid the users in SoD_U to activate the role $r$concurrently;
14. end foreach
15. form these A-mappings into the role mappings RM;
16. return RM.
```

5.4 Violation of policy assignment SoD

The policy assignment *SoD* means that the conflicting policies are prohibited from being authorized to the same role. Generally speaking, role mappings will not lead to violation of policy assignment *SoD* directly. For instance, as the Figure 7 shows, in the local domain L, p_4 and p_5 are the conflicting policies. If the external user requests the two privileges concurrently, this request can not be satisfied completely from the *I-mapping*. This type of violation usually arises from the *I-hierarchies* and the role mapping between two domains. We can formalize it as: $\{\exists p_1, p_2 \in P, \exists r \in R, p_1$=<$r_1$, *PS, d*>, p_2=<r_2, *PS, d*>$|SoD(p_1, p_2)\wedge Containned(r_1, r) \wedge Containned(r_2,r)\}$.

We can resolve this problem through *A-mapping*. If user u_1 requests the conflicting permissions $\{p_4, p_5\}$, the roles r_4 and r_5 will be selected for mapping by searching in the MUS of the local domain. Since the users in the external domain can not own the permissions p_4 and $p_5$concurrently, we can create the *A-mapping* <r_e, E, r_4, L, A> and <r_e, E, r_5, L, A>. Through these two mappings, the user $u_1$can activate the local role $r_4$or r_5, but can not activate them concurrently.

When violation of policy assignment *SoD* occurs, the best way is to establish the *A-mappings* from external to the requested roles. Through *A-mappings* to every requisite role, we can make these roles hold the *SoD* relations. The user can only activate a single role of them in a single session. In this way, the conflicting permissions are prohibited from being authorized to the same user. In general, the following algorithm can be used to detect and resolve the violation of the policy assignment *SoD*, as shown in Algorithm 7.

Algorithm 7. The resolution for violation of policy assignment *SoD*.

```
Input:The role hierarchies with role mappings and authorization assignment in a special multi-domain environment HS.
Output:RM: The role mappings without violation of policy assignment SoD.
NonVPSGeneration(HS)
1.  foreach role $r \in R$ do
2.      get all permissions of the role$r$: $Perms(r)$;
3.      if there are SoD constrains in $Perms(r)$ then
4.          select the conflicting permissions into a set SoD_P;
5.      foreach permission $p \in$ SoD_P do
6.          get the role $r'$ which is assigned the permission $p$ directly;
7.          if there is an I-mapping from external domain to $r'$then
8.              transform the I-mapping into the relevant A-mapping;
9.      end foreach
10.     forbid the users in external domain to activate the role $r'$concurrently;
11. end foreach
12. form these A-mappings into the role mappings RM;
13. return RM.
```

5.5 Conflicts of cyclic inheritance

Conflicts of cyclic inheritance mainly occur in interoperation of systems employing multilevel security policies, such as lattice-based access control (LBAC) and role-based access control (RBAC) [13] [27].

The consequence of cyclic inheritance is that the junior roles obtain the senior role's permissions. Even the permission set of junior role will be larger than the senior one. Hence, the cyclic inheritance violates the security principle which is mentioned above. It enables the roles to access the resources that are not permitted in the domain. It is the typical conflict in multi-domain environment which we try to avoid.

In the example shown in Figure 8, if user u_1 requests the permission $\{p_1, p_2\}$, traditional role mapping $<r_{A2}, D_1, r_{B2}, D_2, I>$ will cause the circulation in role hierarchy. The junior roles r_{A2} and r_{B2}will inherit the permission of their seniors. If we replace this *I-mapping* with an *A-mapping*: $<r_{A2}, D_1, r_{B2}, D_2, A>$, the user $u_1$can acquire the permission $\{p_1, p_2\}$ through activating the role r_{B2}. Since the permissions of the role r_{B2} are not inherited by the role r_{A2}, the cyclic inheritance is avoided.

Cyclic inheritance usually arises from the circulation in the *I-hierarchy*. As *A-mapping* denotes the activation relation between the mapped roles, there is no privilege inheritance in *A-mapping*. In the multi-domain interoperation, *A-mapping* can be used to resolve the conflict of the cyclic inheritance. In general, the following algorithm can be used to detect and resolve this type of conflict, as shown in Algorithm 8.

Algorithm 8. The resolution for conflicts of cyclic inheritance.

Input:The role hierarchies with role mappings and authorization assignment in a special multi-domain environment *HS*.

Output:*RM*: The role mappings without conflicts of cyclic inheritance.

```
NonVCIGeneration(HS)
1.  foreach role $r \in R$ do
2.      get all senior role of $r$: $R'$;
3.      get all permissions of role $r$: $Perms(r)$;
4.      foreach arbitrary role $r' \in R'$ with directly assigned permissions
5.          if $Perms(r) \supseteq Perms(r')$ then
6.              if $r$ is associated with an I-mapping then
7.                  transform the I-mapping into the relevant A-mapping;
8.      end foreach
9.  end foreach
10. form these A-mappings into the role mappings RM;
11. return RM.
```

In this section, we discuss why the conflicts arise in the interoperation among multi-domain environments. For each type of conflicts, we propose the corresponding algorithms to automatically resolve conflicts. These algorithms are all polynomial solutions. For a special multi-domain with n roles, m users, k mappings and q permissions, the complexity analysis of the conflict resolution algorithms is shown in Table 3.

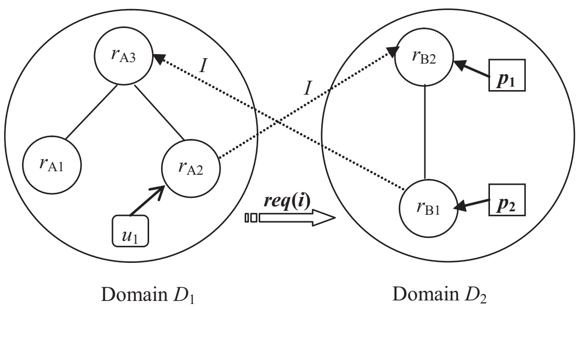


Figure 8 An example of cyclic inheritance between two domains.

Table 3 The complexity analysis of the conflict resolution algorithms.

Algorithm No	Algorithm Name	Complexity
Algorithm 5	*NonVRSGeneration*	$O(m \times n^2 \times k)$
Algorithm 6	*NonVUSGeneration*	$O(m^2 \times n \times k)$
Algorithm 7	*NonVPSGeneration*	$O(n \times q^2 \times k)$
Algorithm 8	*NonVCIGeneration*	$O(n^3 \times q \times k)$

6. ILLUSTRATION AND ANALYSIS

In this section, we illustrate the proposed request-driven policy framework through considering interoperation among various offices of a county. The application scenario is also illustrated by B. Shafiq [25]. Figure 9 shows the role hierarchies of the clerk and treasure departments.

There are two domains in Figure 9: *County Treasurer Office* (*CTO*) and *County Clerk Office* (*CCO*). The roles in the domain *CTO* are as follows: *Tax Collection Manager* (*TCM*), *Tax Assessment Clerk* (*TAC*), *Tax Billing Clerk* (*TBC*), *Tax Collection Clerk* (*TCC*), *Junior Tax Collection Clerk* (*JTCC*) and *Tax Collection Assistant Clerk* (*TCAC*). There is a *SoD* constraint between roles *TAC* and *TBC*. There are only two roles in the domain *CCO*: *Property Tax Manager* (*PTM*) and *Property Tax Clerk* (*PTC*). For the relationship among the roles, the solid line represents the *I-hierarchy* while the broken line represents the *A-hierarchy*.

In this example, the user *TOM* in the domain *CCO* requests the privilege $\{p_2, p_3, p_4\}$ in the domain *CTO*. For this request, there are four roles combination: $\{\{TCC\}, \{TBC, JTCC, TCAC\}, \{TCC, TCAC\}, \{TCC, JTCC, TCAC\}\}$. Following the algorithm *ComputeMUS* (see Algorithm 2), we can get the MUS of the domain *CCO* without the above role sets:

$\{\{TAC\}, \{TCM\}, \{TBC\}, \{TCC\}, \{JTCC\}, \{PTC\}, \{TAC, TBC\}, \{TAC, TCM\}, \{TAC, TCC\}, \{TAC, JTCC\}, \{TAC, TCAC\}, \{TAC, JTCC, TCAC\}, \{TBC, TCM\}, \{TBC, TCC\}, \{TBC, JTCC\}, \{TBC, TCAC\}, \{JTCC, TCAC\}, \{TBC, JTCC, TCAC\}, \{TAC, TBC, TCM\}, \{TAC, TBC, TCC\}, \{TAC, TBC, JTCC\}, \{TAC, TBC, TCAC\}, \{TAC, TBC, JTCC, TCAC\}\}$.

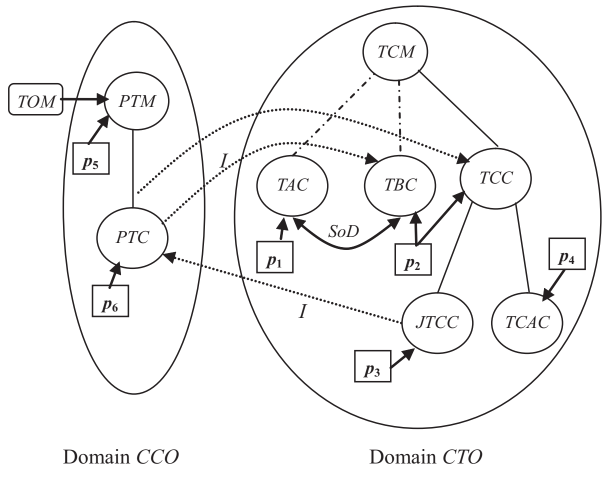


Figure 9 The role hierarchies of the departments of clerk and treasure.

So, for the request $\{p_2, p_3, p_4\}$, we can find out the roles: $\{TCC\}$ or $\{TBC, JTCC, TCAC\}$. The reason for this situation is that the direct assigned permissions for role *TBC* and *TCC* are the same. But in our algorithm, the direct assigned permissions for different roles are acquiescently different. In this case, we can select the role set $\{TCC\}$ to be mapped by the external roles. Since the role mapping <*JTTC*, *CTO*, *PTC*, *CCO*, *I*> exists before the requesting, if we establish the role mapping <*PTC*, *CCO*, *TCC*, *CTO*, *I*>, it will cause the cyclic inheritance. To resolve this problem, we can replace the mapping between *PTC* and *TCC* with *A_mapping* < *PTC*, *CCO*, *TCC*, *CTO*, *A*>.

If the user *TOM* requests the permissions $\{p_1, p_2\}$, through searching in the MUS, the role set $\{TAC, TBC\}$ can be mapped. We assume that the mapping <*PTC*, *CCO*, *TBC*, *CTO*, *I*> is already established. That is to say, the user can acquire the permission $p_2$through the junior role *PTC*. Now the user should have the right of the role *TAC* to acquire the permission p_1. However, violation of the role-specific *SoD* will be raised in such circumstances, no matter we create the mapping between *PTM* or *PTC* and *TAC*. The solution could be removing the mapping <*PTC*, *CCO*, *TBC*, *CTO*, *I*> and establishing two *A-mappings*: <*PTM*, *CCO*, *TAC*, *CTO*, *A*> and <*PTM*, *CCO*, *TBC*, *CTO*, *A*>. We can not use the *A-mappings* <*PTC*, *CCO*, *TAC*, *CTO*, *A*> and <*PTC*, *CCO*, *TBC*, *CTO*, *A*> since the user *TOM* is not assigned with the role *PTC* directly and can not activate the roles *TAC* or *TBC*.

The existing approaches to resolve these conflict problems in this example are usually to sacrifice some user requests [18]. For instance, the priorities of the permissions can be used to select the requested roles. Once the user requests two conflict permissions, the permission with higher priority will be acquired and the one with lower priority will be rejected. In the above example, since there is a *SoD* constrain between the roles *TAC* and *TBC*, if *TOM* requests the permissions $\{p_1, p_2\}$, the role with higher priority permissions, say *TAC* with p_1, will be selected. But the method proposed in this paper is to establish the relationships between the user and all the requested roles. Through these relationships, the existing constrains can be remained and reflected in the role mappings, and no privilege requests will be abnegated.

To illustrate the proposed methods, we implemented an authorization administration system with request-driven policy framework. The system can manage the resources, privileges, roles, users and relationships of multi-domains expediently. The associations among the roles in different domains can be established under the request of the users in this system. The authorization and privilege validation are implemented as web services. Through implementing the interoperation and privilege query among the individual systems, this framework can provide the centralized authorization service for numerous web applications.

Figure 10 shows the snapshot of the above instance in the Multi-Domain Access Control System. It is a service deployed in the web application server complying with the J2EE standard. The table "map to" states the role mappings from local domain to the external domains. Accordingly, the table "map from" states the role mappings from external domains to the local domain. These role mappings are worked out automatically according to the privilege request. When users access the resources in the external domains, the system will validate the authorizations of the users according to the role mapping table.

7. RELATED WORK

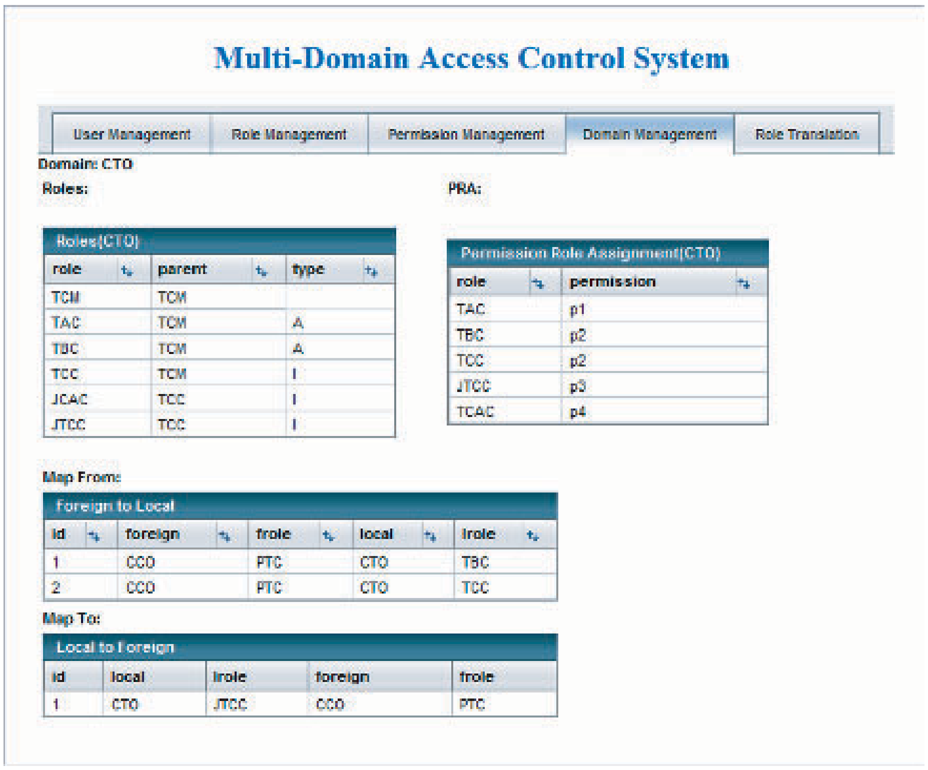
Several research efforts [13][27][28] have been devoted to the topic of policy composition and secure interoperation in multi-domain environment. In [29], M. Shehab et al propose a distributed secure interoperability protocol that ensures secure interoperation of the multiple collaborating domains without compromising the security of collaborating domains. In [8], the authors propose a breadth-first search based algorithm for policy mapping between two loosely coupled interacting domains for sharing resources. However, the proposed algorithm does not perform an exhausted search; instead, it creates new roles even if it is possible to combine roles in the local domain to satisfy the requested permissions.

More relevantly, J. B. D. Joshi et al [23] have tried different approaches to facilitate the administration of role hierarchy by constructing the actual UAS set. Though the first approach in [23] is slightly better in terms of time complexity, both approaches are non-polynomial solutions. The role mapping is to resolve the privilege request between different domains. It provides a method for the external users to access the roles in the local domains. Because the UAS set is the activable roles set by a user that is assigned to a role in a single session, the local UAS can not provide the optimum roles set to be mapped for a special external request. As an extension of the UAS, the MUS can be considered as the minimal unique set for a hybrid role hierarchy. For arbitrary privileges provided by a role hierarchy, the external subjects can know which roles in the local domain are the most suitable ones for a special request through searching in the MUS.

B. Shafiq et al [25] propose a policy integration framework for merging heterogeneous role-based access control (RBAC) policies of multiple domains into a global access control policy. An integer programming (IP)-based approach is proposed as the optimal resolution for such conflicts. The method to resolve most conflicts arising from role mappings in his paper is to remove some of the mapping links specified in the role mappings. But it is difficult to decide which links should be removed. Moreover, some requests of the external users will not be satisfied if some mappings are canceled.

There are several researches concerning the resolution of the conflicts among role mappings. E. C. Lupu et al [16] focuses on the problems of conflict detection and resolution for policy conflicts, including authorization policies and obligation policies. They discuss various precedence relationships that can be established between policies in order to allow inconsistent policies to coexist within the system and present a conflict analysis tool that partially forms a role-based management framework. The main idea for conflict resolution is to set the priorities of the policies. But it is also difficult to set the priorities of the policies in this approach. That is, it is too hard to judge which conflicting permissions will be acquired preferentially according to the user requests. Through our analyses and comparison, selecting adaptive types of role mappings for a special external request is an easy and effective method to avoid the conflicts.

Time-based secure interoperation [30] has not been addressed by earlier models. Differing from the TRBAC [31] model which addresses the role enabling constraints only, J. B. D. Joshi et al [9] propose a generalized temporal role-based access control



Domain: CTO

Roles:

Roles(CTO)

role	parent	type
TCM	TCM	
TAC	TCM	A
TBC	TCM	A
TCC	TCM	I
JCAC	TCC	I
JTCC	TCC	I

PRA:

Permission Role Assignment(CTO)

role	permission
TAC	p1
TBC	p2
TCC	p2
JTCC	p3
TCAC	p4

Map From:

Foreign to Local

id	foreign	frole	local	lrole
1	CCO	PTC	CTO	TBC
2	CCO	PTC	CTO	TCC

Map To:

Local to Foreign

id	local	lrole	foreign	frole
1	CTO	JTCC	CCO	PTC

Figure 10 The snapshot of the Multi-Domain Access Control System.

(GTRBAC) model capable of expressing a wider range of temporal constraints. GTRBAC allows describing periodic as well as duration constraints on roles, user-role assignments, and role-permission assignments. In GTRBAC, role activation can further be restricted by numerous activation constraints, including cardinality constraints and maximum active duration constraints.

As an important extension of the GTRBAC model, J. B. D. Joshi [32] presents design and implementation of X-GTRBAC Admin, an administration model that aims at enabling administration of role-based access control (RBAC) policies with constraints and coordinates conflict resolution in a multi-domain environment.

The hybrid hierarchy represents the most prevalent relationship among the roles. Other earlier work related to hybrid hierarchy that highlights its importance can be found in [7]. In our research, we focus on the role mappings among different hybrid role hierarchies. The relations of the roles in the mappings between different domains are also hybrid hierarchies. Hence, our approach presented in this paper can also be considered as the implementation of the GTRBAC model for the interoperation in a multi-domain environment.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we propose a request-driven policy framework for interoperation in a multi-domain environment. While hybrid hierarchy is important to make the RBAC approach generic enough to support flexible policy expression and inter-domain policy mapping, we propose a more convenient and effective method to perform the permission query in a local domain based on the minimal unique set (MUS). Through this method, the suitable role set will be figured out for the external privilege requests, and the role mappings between the external and local domains will be established based on the query result.

The role mapping in different domains is a cause of the various types of conflicts and inconsistencies. In this paper, the violation of role-specific *SoD*, violation of user-specific *SoD*, violation of policy assignment *SoD* and cyclic inheritance conflicts can be resolved by selecting adaptive roles and role mappings between the external and local domains. This paper analyses the reasons and presents the resolutions for these conflicts. Compared to other researches, this method can ensure that that the external user requests will be satisfied and the local role hierarchies will be furthest preserved.

In a special role hierarchy, the elements in the MUS are the minimal unique role set for a set of privileges. The MUS is essential for the privilege query. At present, the algorithm to compute MUS in this paper is non-polynomial solutions. Algorithm optimization and polynomial solutions may be the further research considerations.

9. ACKNOWLEDGEMENTS

This work was supported by National Natural Science Foundation of China under Grant 60403027, 60773191, 70771043, National High Technology Research and Development Program of China under Grant 2007AA01Z403, China Postdoctoral Science Foundation under Grant 20060400846, Natural Science Foundation of Hubei Province of China under Grant 2005ABA258, Open Foundation of State Key Laboratory of Software Engineering under Grant SKLSE05-07. The authors would like to thank the anonymous reviewers for their valuable comments.

REFERENCES

1. J. B. D. Joshi, A. Ghafoor, W. Aref, and E. H. Spafford, "Digital Government Security Infrastructure Design Challenges," *IEEE Computer*, vol. 34, no. 2, pp. 66–72, 2001.
2. R. Power, "Tangled Web: Tales of Digital Crime from the Shadows of Cyberspace," *Que/Macmillan Publishing*, August 2000.
3. P.-P. Lu, B. Li, M.-L. Xing, and L. Li, "An Automated Trust Negotiation Framework Based on Subjective Trust Model," *Proceedings of the 6th International Conference on Machine Learning and Cybernetics (ICMLC'07)*, Hong Kong, pp.19–22, 2007.
4. O. Ajayi, R. Sinnott, and A. Stell, "Trust Realisation in Multi-domain Collaborative Environments," *Proceedings of the 6th IEEE/ACIS International Conference on Computer and Information Science (ICIS'07)*, Melbourne, Australia, pp. 906–911, 2007.
5. G. Zhao, D. Zheng, and K. Chen, "Design of single sign-on," *Proceedings of the 2004 IEEE International Conference on E-Commerce Technology for Dynamic E-Business (CEC-East'04)*, Beijing, China, pp.253–256. 2004.
6. JA-SIG,"*ITS Central Authentication Service,"* http://www.yale.edu/tp/cas/.
7. J. B. D. Joshi, E. Bertino, and A. Ghafoor, "Temporal Hierarchies and Inheritance Semantics for GTRBAC," *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT'02)*, Monterey, California, USA, pp. 74–83, 2002.
8. S. Piromruen, and J. B. D. Joshi, "An RBAC Framework for Time Constrained Secure Interoperation in Multi-domain Environments," *Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'05)*, Sedona, Arizona, USA, pp. 36–48, 2005.
9. J. B. D. Joshi, E. Bertino, and U. Latif, "A Generalized Temporal Role-Based Access Control Model," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 1, pp. 4–23, 2005.
10. S. Baselice, P. A. Bonatti, and M. Faella, "On interoperable trust negotiation strategies," *Proceedings of the Eighth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'07)*, Bologna, Italy, pp. 39–50, 2007.
11. H. Chen, and N. Li, "Constraint Generation for Separation of Duty," *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT'06)*, Lake Tahoe, California, USA, pp. 130–138, 2006.
12. P. Bonatti, S. D. C. Vimercati, and P. Samarati, "An Algebra for Composing Access Control Policies," *ACM Transactions on Information and System Security*, vol. 5, no. 1, pp. 1–35, 2002.
13. S. Dawson, S. Qian, and P. Samarati, "Providing Security and Interoperation of Heterogeneous Systems," *Distributed and Parallel Databases*, vol. 8, no. 1, pp. 119–145, 2000.
14. E. Bertino, F. Buccafurri, E. Ferrari, and P. Rullo, "A Logical Framework for Reasoning on Data Access Control Policies," *Proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW'99)*, Mordano, Italy, pp. 175–189, 1999.
15. R. T. Simon, and M. E. Zurko, "Separation of Duty in Role-Based Environments," *Proceedings of the 10th IEEE Computer Security Foundations Workshop (CSFW'97)*, Rockport, MA, USA, pp.183–194, 1997.
16. E. Lupu, and M. Sloman, "Conflicts in Policy-Based Distributed Systems Management," *IEEE Transactions on Software Engineering*, vol. 25, no. 6, pp. 852–869, 1999.
17. M. Jaume, and C. Morisset, "Formalisation and implementation of Access control models," *Proceedings of the 2005 International Conference on Information Technology: Coding and Computing (ITCC'05)*, pp. 703–708, 2005.
18. R. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role Based Access Control Models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.
19. R. Sandhu, "Role Activation Hierarchies," *Proceedings of the 3rd ACM Workshop Role-Based Access Control*, Fairfax, VA, USA, pp. 33–40, 1998.
20. J. B. D. Joshi, E. Bertino, and A. Ghafoor, "Hybrid Role Hierarchy for Generalized Temporal Role Based Access Control Model," *Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC'02)*, Oxford, England, pp. 951–956, 2002.
21. Z. Hu, J. Zhang, and X. Luo, "Design and Realization of Efficient Memory Management for Embedded Real-Time Application," *Proceedings of the 6th International Conference on ITS Telecommunications (ITST'06)*, Chengdu, China, pp. 174–177, 2006.
22. S. M. Chandran, and J. B. D. Joshi, "Towards Administration of a Hybrid Role Hierarchy," *Proceedings of the 2005 IEEE International Conference on Information Reuse and Integration (IRI'05)*, Las Vegas, Nevada, USA, pp. 500–505, 2005.
23. J. B. D. Joshi, E. Bertino, and A. Ghafoor, "Formal Foundation for Hybrid Hierarchies in GTRBAC," *ACM Transactions on Information and System Security*, to be published, 2008.
24. S. Du, and J. B. D. Joshi, "Supporting Authorization Query and Inter-domain Role Mapping in Presence of Hybrid Role Hierarchy," *Proceedings of the 11th ACM Symposium on Access Control, Models and Technologies (SACMAT'06)*, Lake Tahoe, California, USA, pp. 228–236, 2006.
25. B. Shafiq, J. B. D. Joshi, and E. Bertino, "Secure Interoperation in a Multidomain Environment Employing RBAC Policies," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 11, pp. 1557–1577, 2005.
26. J. B. D. Joshi, E. Bertino, B. Shafiq, and A. Ghafoor, "Dependencies and Separation of Duty Constraints in GTRBAC," *Proceedings of the 8th ACM Symposium on Access Control, Models and Technologies (SACMAT'03)*, Como, Italy, pp. 51–64, 2003.
27. L. Gong, and X. Qian, "Computational Issues in Secure Interoperation," *IEEE Transactions on Software Engineering,* vol. 22, no.1, pp. 43–52, 1996.
28. P. A. Bonatti, M. L. Sapino, and V. S. Subrahmanian, "Merging Heterogeneous Security Orderings," *Proceedings of the 4th European Symposium on Research in Computer Security: Computer Security (ESORICS'96)*, Rome, Italy, pp. 183–197, 1996.
29. M. Shehab, E. Bertino, and A. Ghafoor, "SERAT: Secure Role mapping Technique for Decentralized Secure Interoperability," *Proceedings of the ACM Symposium on Access Control, Models and Technologies (SACMAT'05)*, Stockholm, Sweden, pp. 159–167, 2005.
30. J. B. D. Joshi, E. Bertino, and A. Ghafoor, "An Analysis of Expressiveness and Design Issues for the Generalized Temporal Role-Based Access Control Model," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 2, pp. 157–175, 2005.
31. E. Bertino, and P. A. Bonatti, "TRBAC: A Temporal Role-Based Access Control Model," *ACM Transactions on Information and System Security*, vol. 4, no. 3, pp. 191–223, 2001.
32. J. B. D. Joshi, "X-GTRBAC Admin: A Decentralized Administration Model for Enterprise-Wide Access Control," *ACM Transactions on Information and System Security*, vol. 8, no. 4, pp. 388–423, 2005.