HAL
open science

Secure Interoperation in Heterogeneous Systems based on Colored Petri Nets

Hejiao Huang, Helene Kirchner

To cite this version:

Hejiao Huang, Helene Kirchner. Secure Interoperation in Heterogeneous Systems based on Colored Petri Nets. 2009. inria-00396952

HAL Id: inria-00396952
https://inria.hal.science/inria-00396952

Preprint submitted on 19 Jun 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Secure Interoperation in Heterogeneous Systems based on Colored Petri Nets

Hejiao Huang[1,2] Hélène Kirchner[2]
1. Harbin Institute of Technology Shenzhen Graduate School, China
2. INRIA Bordeaux Sud-Ouest, France
hejiao0@yahoo.com.cn; Helene.Kirchner@inria.fr

Abstract

In a multi-domains application environment, where distributed multiple organizations interoperate with each other, the local access control policies should correspondingly be integrated in order to allow users of one organization to interact with other domains. One of the key challenges of integrating policies is conflict detection and resolution while preserving policy consistency. This paper addresses several types of potential conflicts and consistency properties with a systematic and rigorous approach. In the approach, graph theory, network flow technology and colored Petri nets are applied for specifying and verifying a secure interoperation design. The component-based integration of policies is applicable for both static and dynamic multi-domains environments.

Keywords: *secure interoperation, conflict resolution, colored Petri nets, dynamic environment.*

I. Introduction

Security policies are one of the most fundamental elements of computer security. In order to address the requirement of multi-domains interoperation, current security policy design trends to the composition of components and interactions between them. Consequently, in a component-based specification and verification of a policy, the composition of local policies must consistently preserve some properties of security policies. A rigorous and systematic way to predict and assure such critical properties is crucial.

Composition of the local access control policies into a global coherent security policy is a challenging problem since many kinds of violations may occur during the composition. In a RBAC policy, the potential conflicts appearing in an interoperation can be considered in the following aspects:

1) role inheritance violation: in an interoperation policy, the cross domain hierarchy relationship may introduce a path in the interoperation domain enabling a role A which has no inheritance relation with B in their local domain to assume the permission of B. The cyclic inheritance addressed in some literature [1], [2], [3] belongs to the role inheritance violation;
2) cardinality for roles, users and shared resources: they are numerical restrictions on access control entity assignments. Concretely, a role r with cardinality r_i implies that the number of users simultaneously assigned with r cannot exceed r_i; a user u with cardinality u_j implies that the number of roles simultaneously assigned to u cannot exceed u_j; a resource o with cardinality o_m implies that the number of users simultaneously accessing to o cannot exceed o_m;
3) separation of duty (SoD): in a local policy or in an interoperation policy design, some conflicting roles are not permitted to be assigned to a same user; some roles with conflicting privileges are not permitted to be activated simultaneously in a session; and some conflicting users cannot be assigned with the same role;
4) resource sharing: in an interoperation policy, local resources are permitted to be shared by the collaborated domains. Because of resource cardinality restriction, the case of "circular waiting" for resources may occur and result in system deadlock.

Besides conflicts resolution, the following problems should also be addressed in order to get an efficient and consistent interoperation policy.

1) user-role assignment and optimization: in each session, how to assign users to their qualified roles is an

This work was supported in part by National Natural Science Foundation of China with Grant No. 10701030 and by INRIA.

issue to be handled well in order to make the policy more efficient. This optimization problem is to find a user-role assignment with as many (user, role) pairs as possible without cardinality violation;

2) dynamic environments: the interoperation policy design should permit the addition of new domains and new users, deletion of domains from the collaboration, and evolution of local domain's policies.
3) consistency verification: the interoperation policy should be consistent with the constituent policies. This was expressed in [2], [3] by the autonomy and security principles. Autonomy principle means, if an access is permitted in the local policy, it must also be permitted in the global policy; security principle means, if an access is not permitted in the local policy, it must not be permitted under secure interoperation.

As described above, conflict resolution and consistency verification are the main concerns for dealing with secure interoperation in multi-domains environments. The problems have been partly addressed in literature. [3] resolves the role assignment violation and SoD violation with a 0-1 integer programming (IP) approach, where the objective is to maximize the inter-domain role accesses. The violation constraints are modeled as the constraints of the IP problem. The domain of integer programming provides some approximation algorithms such as Lagrangian relaxation, tabu search and simulated annealing for solving IP problem. But the approach is not applicable for a real large policy system because the constraints are too many. For example, in the illustrative example shown in the literature, there are almost 1500 constraints for a small system. Another difficulty for the IP approach is that, the selection of the weight for the objective function is itself an open research issue. When the interoperation occurs in a dynamic environment, where addition of new domains and removal of domains of collaboration are permitted, the IP approach meets even more challenges to be rephrased.

In [4], the authors apply a modified Petri net, called a Conflict Petri Net, to analyze the interoperation and evaluate the possibility of security violations. They detect conflicts concerning separation of duty, temporal constraints, and cardinality constraints. With these conflict Petri nets and the defined firing rules, although some violation can be detected, the resolution of conflicts is hard and policy consistency verification is even more challenging.

The paper [5] presents a goal-oriented and model-driven approach to analyze the security features of local policies and a guideline for integrating them to fulfill the security goals of the global policy. The proposed analysis procedure leads to the discovery of three classes of security interoperability conflicts: different definitions of security, different implementation mechanisms and conflicting authorization policies, which helps to determine whether or not the domain policies should be used together. How to resolve such conflicts is not the point of the paper.

The paper [6] handles resource sharing in a mediator-free scheme. In the scheme, every local domain creates and maintains cross-domain policies; any security violation to any domain can be detected by the domain itself and each domain finding a security violation raises a negotiation among domains to remove the violation; each domain is responsible for making the access decisions when entities from other domains access to its own resources. Since the paper focuses on resource sharing, conflict detection and resolution are not considered in detail.

The paper [7] proposes a security violation detection method for RBAC based interoperation. This method reduces complexity by decreasing the amount of roles involved in detection.

There is much earlier related work [1], [2], [8], [9] concerning cyclic inheritance, conflict resolution and SoD violation resolution [9], [10]. Policy consistency is also considered in [11]. A brief review of these papers can be found in [3].

As analyzed above, up to date approaches contain some shortages when considering the above secure interoperation problem. Handling policy composition in rigorous and systematic way is yet a challlenge.

Taking the viewpoint of policy system design, this paper addresses a component-based systematic approach for specifying and verifying an interoperation policy by integrating the constituent policies. The approach and the main contribution are roughly outlined as follows: first, the role inheritance hierarchy is modeled with role inheritance graphs. Some crossing arcs between these inheritance graphs are applied for specifying the interoperation. When a role inheritance violation (RIV) occurs, a depth-first search algorithm is provided for resolving the RIV problem; then, role-privilege assignment and user-role assignment are specified with a colored Petri net (CPN); based on this CPN model, cardinality constraints are specified with the number of tokens in the corresponding places, and control places are applied for solving the SoD constraints; the deadlock-free resource sharing problem is handled by applying place merging operators; in order to get an efficient access control ability, how to activate a session is converted into the max-flow problem which can be solved by existing algorithms. The final model for interoperation is a colored Petri net and an access decision corresponds to a reachable marking in the colored Petri net specification. The CPN model applied in this paper is quite simple and equivalent to ordinary Petri nets. Hence, the reachability analysis techniques adopted for ordinary Petri nets are also applicable to our CPN model. During the process of integration, the previously mentioned conflicts

and principles are considered for specification, resolution and verification. Conflict resolution and consistency can be guaranteed in this approach.

The motivation of applying Petri nets for the specification and verification of interoperation policy is that Petri nets are well-known for their graphical and analytical capabilities for the specification and verification of concurrent and distributed systems. They have two main features particularly convenient for our methodology of interoperation security policy design:

1) Petri net representations are analytical and flexible. Analysis and logical reasoning can be performed on their representations and on their properties. They are compatible with a compositional approach via operators for compositions, refinements and reductions, and their functional purposes and characteristics are accurately and logically reflected.
2) Many Petri net-based techniques are available for verification, including reachability analysis or mathematical programming, as well as for characterization and transformations (see [12], [13] for a review). There are also abundant results concerning property preserving operators.

Motivated by these advantages, much work about applying Petri net for the policy design has appeared in the literature. In [14], a colored Petri net (CPN) based framework is presented for verifying the consistency of RBAC policies. In [15], CPN is used to specify a real industrial example. Based on the CPN model, the Design/CPN tool is applied for the implementation of automatic code generation. [16] defines task based access control as a dynamic workflow and then specifies the workflow with Petri nets. [17], [18] model Chinese wall policy and Strict Integrity Policy, respectively, with CPN and apply coverability graph for the verification. [19] uses CPN for the specification of mandatory access control policies and occurrence graph is applied for verification. [20] applies Predicate/Transition net for the modeling and analysis of software security system architectures.

We have to mention that the above Petri net based work considers policy design in a single local domain environment. [4] tried to handle conflict problems in an interoperation environment, but the approach is neither complete nor rigorous as analyzed above. The approach presented in this paper is, to the best of our knowledge, among the first efforts on systematic integration and analysis of security policies for both static and dynamic multi-domains environments in the literature.

The remaining part of the paper is organized as follows: Section II introduces some related terminology about Petri nets and a discussion of our model; Section III is about the detailed policy specification and conflict resolution; the verification technique is presented in Section IV and Section V illustrates the policy design technology with an example; some conclusive remarks are given in Section VI.

II. Basic Terminology about Petri Nets

This section briefly introduces some basic terminologies about Petri nets and colored Petri nets applied in this paper. More information can be found in [21], [22].

Definition 1 (Petri nets): A Petri net (N, M_0) is a net $N = (P, T, F, W)$ with an initial marking M_0 where P is a finite set of places; T is a finite set of transitions such that $P \cap T = \emptyset$, and $P \cup T \neq \emptyset$; $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation; W is a weight function such that $W(x, y) \in \mathcal{N}^+$ if $(x, y) \in F$ and $W(x, y) = 0$ if $(x, y) \notin F$; M_0 is a function $M : P \to N$ such that $M(p)$ represents the number of tokens in place $p \in P$.

The idea of colored Petri net (CPN) is to introduce the notion of token types. Tokens are differentiated by colors, which may be arbitrary data values. Each place has an associated type determining the kind of data that the place may contain. The precise definition can be found in [22].

Definition 2 (Colored Petri nets (CPN)): A colored Petri net is a tuple $CPN = (\Sigma, P, T, A, V, C, G, E, I)$, where

1) *Σ is a finite set of non-empty types, also called color sets. The set of types determines the data values and the operations and functions that can be used in the arc expressions, guards and initialisation.*
2) *P is a finite set of places.*
3) *T is a finite set of transitions.*
4) *A is a finite set of arcs such that: $P \cap T = P \cap A = T \cap A = \emptyset$*
5) *V is a node function $V : A \mapsto (P \times T) \cup (T \times P)$.*
6) *C is a color mapping $C : P \mapsto \Sigma$. The color function C maps each place p to a type $C(p)$, which means that each token on p must have a data value that belongs to $C(p)$.*
7) *G is a guard function. It is defined from T into expressions such that $\forall t \in T : Type(G(t)) = Boolean$ and $Type(Var(G(t)) \subseteq \Sigma$. For any transition $t \in T$, the guard of t is a boolean expression, where all variables have types that belong to Σ.*
8) *E is an arc expression function. It is defined from A into expressions such that: $\forall a \in A : Type(E(a)) = C(p)_{MS}$ and $Type(Var(E(a)) \subseteq \Sigma$, where p is the place of $V(a)$. For any arc $a \in A$, the type of $E(a)$ is the multiset $C(p)_{MS}$, which means that $E(a)$ must evaluate to multisets over the type of the adjacent place p. Moreover all variables in an arc expression $E(a)$ have a type in Σ.*
9) *I is an initialization function. It is defined from P into closed expressions such that: $\forall p \in P : Type(I(p)) =$*

$C(p)_{MS}$. I maps each place $p \in P$ into a closed expression that must be of type of $C(p)_{MS}$.

In this paper, colored Petri nets are much simpler than those of Definition 2, since we do not consider guard functions for transitions. Hence, the firing rule for a CPN is similar to that for an ordinary Petri net, except that the markings and weights are represented as multiple dimensional vectors. Our CPN is just a folding of several ordinary Petri nets with the same net structure. Hence, our CPN is equivalent to an ordinary Petri net and the reachablity analysis techniques for ordinary Petri nets are also applicable for our CPN model. The meaning of equivalence is defined below.

Definition 3 (equivalent nets): [23] *Two nets (N_1, M_{10}) and (N_2, M_{20}) are said to be equivalent if there exists a 1-1 mapping ρ between the markings of these two nets satisfying: 1) $M_{20} = \rho(M_{10})$ and 2) any two markings $M_{11}, M_{12} \in R(N_1, M_{10})$ satisfy $M_{11}[N_1, \sigma_1\rangle M_{12}$ iff $\rho(M_{11}), \rho(M_{12}) \in R(N_2, M_{20})$ and $\rho(M_{11})[N_2, *\rangle\rho(M_{12})$ in (N_2, M_{20})*

Usually, colors in the CPN are used to distinguish different types of data. In this paper, besides some ordinary tokens, there are three types of color sets: $Users = \{u_1, u_2, \cdots, u_i\}$, $Roles = \{r_1, r_2, \cdots, r_j\}$, and $O = \{o_1, o_2, \cdots, o_m\}$ for distinguishing different users, roles and objects, respectively.

III. Specification of Secure Interoperation Policy

In this section, a formal specification methodology of secure interoperation is addressed. Each of the potential conflicts is considered for specification and resolution in an independent subsection.

A. Specification of domain-integration and conflict resolution

In this paper, we use notation G_D to denote the role inheritance graph in a domain D. We borrow a definition from [6] for defining our role inheritance graph in D, where the similar graph is called domain policy graph with much more complex notations.

Definition 4 (role inheritance graph): Given a policy defined on a domain D, a role inheritance graph is a graph $G_D = (V_D, A_D)$, such that each node v in V_D corresponds to a role in the domain D, and an arc $(u, v) \in A_D$ iff role u is immediately senior to role v in the policy.

Definition 5 (interoperation graph): Let $G_{D_1} = (V_{D_1}, A_{D_1})$ and $G_{D_2} = (V_{D_2}, A_{D_2})$ be two role inheritance graphs in domains D_1 and D_2, respectively. An interoperation graph $G_D = (V_D, A_D)$ in the interoperation domain $D = D_1 * D_2$ is created by integrating G_{D_1} and G_{D_1} as follows: $V_D = V_{D_1} \cup V_{D_2}$, $A_D = A_{D_1} \cup A_{D_2} \cup A_{12} \cup A_{21}$, A_{12} and A_{21} are sets of arcs from V_{D_1} to V_{D_2} and from V_{D_2} to V_{D_1} built in the following way: an arc $(u, v) \in A_{12}$ (resp., A_{21}) iff $u \in V_{D_1}$ (resp., V_{D_2}) is immediately senior to $v \in V_{D_2}$ (resp., V_{D_1}) in the interoperation policy of D.

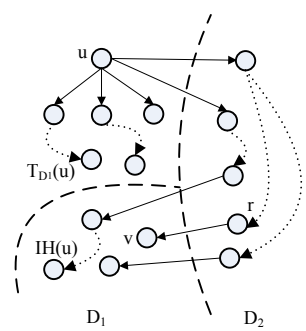

Fig. 1: Shortest path tree $T_D(u)$.

In this construction process, role inheritance violation may occur.

Definition 6 (role inheritance violation (RIV)): RIV occurs when a user u in domain D is allowed to access a local role v even though u is not directly assigned to v or to any of the roles that are senior to v in the role hierarchy of domain D.

Role inheritance violation occurs when a user u in D_1 that cannot access v in D_1 may access a role r in D_2 which is senior to role v in the interoperation domain $D = D_1 * D_2$ (Fig. 1). It is more general than cyclic inheritance violation addressed in [1], [2], [3]. When role u cannot inherit the permissions of role v, u may be either junior to v (the inducement of cyclic inheritance violation) or independent of v in their local domain.

Property 1 *If an interoperation graph G_D has directed cycles, then a role inheritance violation occurs.*

Proof. If there exists a directed cycle in G_D, then any two roles in the cycle can mutually inherit the privilege of each other, which obviously results in RIV. □

The well-known breadth first search algorithm [24] can be applied to generate a shortest path tree rooted at node s in graph G_D. The output of the algorithm is denoted as $T_D(s) = (V_{T_D}(s), A_{T_D}(s))$. Assume that the shortest path tree rooted at node u in G_{D_1} is $T_{D_1}(u) = (V_{T_{D_1}}(u), A_{T_{D_1}}(u))$ and in G_D is $T_D(u) = (V_{T_D}(u), A_{T_D}(u))$. Let $IH(u) = (V_{T_D}(u) - V_{T_{D_1}}(u)) \cap V_{D_1}$, then $IH(u)$ is the set of nodes which may cause role inheritance violation for node u. In order to avoid the role inheritance violation for node u, a set of

arcs $A' \subset A_{12} \cup A_{21}$ should be deleted from G_D.

Property 2 *(Fig. 1) Let us consider a node $u \in V_{D_1}$ in graph $G_{D_1} = (V_{D_1}, A_{D_1})$, and let $D = D_1 * D_2$. The role inheritance violation occurs for u iff $IH(u) = (V_{T_D}(u) - V_{T_{D_1}}(u)) \cap V_{D_1} \neq \emptyset$. Furthermore, for each node $v \in IH(u)$, there exists a node r in V_{D_2} such that r is senior to v in the interoperation policy.*

Proof. By definition of $IH(u)$, the nodes in $IH(u)$ do not belong to the path rooted at node u in D_1, which implies that they are either senior to u or independent of u in the local domain policy. RIV occurs for u iff there exists a node v in $IH(u)$ such that u can access to v, i.e. $IH(u) \neq \emptyset$. Furthermore, u will access v by inheriting the permissions of some node r in D_2 which is senior to v. □

For each arc $(u, v) \in A_{12} \cup A_{21}$, let us assign to the arc (u, v) a weight $e_{uv} = |V_{T_{D_i}}(v)|$ which is the number of nodes in the shortest path tree $T_{D_i}(v)$, where $v \in V_{D_i}, i = 1$ or 2. A weight with a higher value implies more inheritance privileges by crossing domain.

For any $w \in IH(u)$, if $u \notin V_{T_D}(w)$, let us then create an arc (w, u) and denote the set of all these arcs as $A_u = \cup_{w \in IH(u)}(w, u)$. Obviously, the graph $(V_D, A_D - A')$ has no role inheritance violation for node u iff $(V_D, A_D \cup A_u - A')$ has no role inheritance violation for node u.

Let $A_U = \cup_{u \in V_D} A_u$. Based on the graph $G = (V_D, A_D \cup A_U)$, the Resolve RIV algorithm (Fig. 2) is used to break cycles in G in order to avoid role inheritance violation in the interoperation policy. Once there is a directed cycle in the graph G, RIV will exist. In order to break a cycle, one and only one arc needs to be deleted. In order to maximize the resource accessibility and preserve the autonomy of local policies, we always delete a lower weighted arc in $A_{12} \cup A_{21}$.

The time complexity of the depth first search algorithm is $O(m + n)$, where m and n are the number of arcs and nodes in the graph G. For each node u in V_{D_i}, we need to run a breadth first search algorithm [24] with time complexity $O(m + n)$ in order to compute $IH(u), T_D(u), T_{D_i}(u)$ and e_{vu}. The total complexity for resolving the inheritance violation is $O(mn + n^2)$.

It is known that the RIV problem is equivalent to the feedback set problem which is generally NP-hard [2]. In the remaining part of this subsection, the RIV problem in two interoperating domains is considered. More technical details about this problem can be found in [25].

Definition 7 (RIV Problem in two domains): For a graph $G = (V_D, A_D \cup A_U)$ created as above, is there a set of arcs $A' \subset A_{12} \cup A_{21}$ such that: 1) $G = (V_D, A_D \cup A_U - A')$ has no directed cycles; and 2) the sum of the weights for these arcs in A' is minimum.

```
depth first search algorithm
Input a graph G = (V_D, A_D∪A_U)                          /G may contain cycles
begin
for each node s∈V_D, and IH(s)≠∅  do
  begin
    unmark all nodes in graph G
    mark node s                                           /{the initial node}
    pred(s) = 0                                           /{it has no predecessor}
    next:=1                                               /{next is a counter}
   order (s): =s
    LIST ={s}
    while LIST  do
      begin
        select a node i in LIST
        if node i is incident to an inadmissible arc (i, j)      /{a cycle is found}
          begin
            select two arcs (l_i, k_i), i = 1, 2 satisfies that, l_1, k_2 ∈ V_D1, l_2, k_1 ∈ V_D2,
            k_i, l_i ∈ LIST satisfying
            order (j) ≤ order (l_1) ≤ order (k_1) ≤ order (l_2) ≤ order (k_2) ≤ order (i)
            or
            order (j) ≤ order (l_2) ≤ order (k_2) ≤ order (l_1) ≤ order (k_1) ≤ order (i)
              if e_l1k1 < e_l2k2    {G = (V_D, A_D − {(l_1, k_1)})}
              else {G = (V_D, A_D − {(l_2, k_2)})}
            LIST = LIST − {j, …, l_i, …, k_i, …, i}
          end
        else if node i is incident to an admissible arc (i, j) then
          begin
              mark node j;
              pred(j):=i;
              next = next + 1
              order (j):= next;
              add node j to LIST;
            end
        else
        delete node i from LIST
      end
  end
end
output graph G                                            /G does not contain cycles


Greedy algorithm
input G =  (V_D, A_D∪A_U − A_12∪A_21)
order the arcs in A_12∪A_21 in non-decreasing order to weight as e_1≥ e_2≥, …, ≥ e_k
for i =1 to k, do
      if G+{e_i} does not have a directed cycle then G:=G +{e_i}
output G
```

Fig. 2: Resolve RIV algorithm.

The RIV in two interoperation domains is a special case of the feedback arc set problem. In the following, we will convert the RIV problem into a minimum weight vertex cover problem in a bipartite graph and solve it with polynomial-time complexity algorithms. Before that, let us recall the definition for the vertex cover problem and consider a property which is applicable to the problem conversion.

Definition 8 (Vertex cover problem): A vertex cover for an undirected graph $G = (V, E)$ is a subset S of its vertices such that each edge has at least one endpoint in S. The vertex cover problem is the optimization problem of finding a smallest vertex cover in a given graph. The Weighted Vertex Cover Problem is defined as follows: given a simple graph $G = (V, E)$ and a weight function $w : V \to \mathcal{R}^+$, find a vertex cover of minimum total weight.

Property 3 *(Fig. 3) In the graph $G = (V_D, A_D \cup A_U)$ defined above, if there exists a directed cycle C_1 such that $|C_1 \cap (A_{12} \cup A_{21})| > 2$, then there is an arc pair (a_i, b_j) which belong to both C_1 and another directed cycle C_2 in G. Furthermore, $C_2 \cap (A_{12} \cup A_{21}) = \{a_i, b_j\}$.*

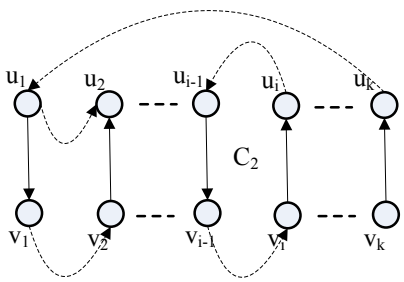

Fig. 3: An inheritance cycle.

Proof. Suppose $C_1 \cap (A_{12} \cup A_{21}) = \{(u_i, v_i), (v_j, u_j)\}, i, j = 1, 2, ..., k$. Let $C_1 = u_1 v_1 ... v_2 u_2 ... v_k u_k ... u_1$. For the vertex u_1, if $u_2 \in IH(u_1)$, then $C_2 = u_1 v_1 ... v_2 u_2 ... u_1$ contains an arc pair (a_i, b_j), where $a_i = (u_1, v_1)$ and $b_j = (v_2, u_2)$ and the property is proved. If $u_2 \notin IH(u_1)$, then let us consider u_2 and repeat the above steps. Since there exists inheritance violation in G, there must exist a $u_i \in IH(u_{i-1})$. By the above steps, an arc pair $((u_{i-1}, v_{i-1})(v_i, u_i))$ belongs to both cycles and satisfies $C_2 \cap (A_{12} \cup A_{21}) = \{(u_{i-1}, v_{i-1})(v_i, u_i)\}$. □

Since cyclic inheritance should be avoided in each local policy design before policy composition for interoperation, we may assume that there is no directed cycle in each local inheritance graph and we have the following property.

Property 4 *Suppose u_i, u_j belong to domain D_1, v_i, v_j belong to domain D_2 and $D = D_1 * D_2$. For two arcs a_i, b_j, where $a_i = (u_i, v_i), b_j = (v_j, u_j)$ in $G = (V_D, A_D \cup A_U)$, there exists a directed cycle C satisfying that $C \cap (A_{12} \cup A_{21}) = \{a_i, b_j\}$ iff $u_j \in IH(u_i)$ (or $u_i \notin IH(u_j)$) and $v_j \notin IH(v_i)$ (or $v_i \in IH(v_j)$.*

Proof. If there exists a directed cycle C satisfying $C \cap (A_{12} \cup A_{21}) = \{a_i, b_j\}$, then there exists a path in D_1 connecting u_j to u_i and hence $u_j \in IH(u_i)$ (or $u_i \notin IH(u_j)$), and a path in D_2 connecting v_i to v_j and hence $v_j \notin IH(v_i)$ (or $v_i \in IH(v_j)$. On the other hand, if $u_j \in IH(u_i)$ (or $u_i \notin IH(u_j)$) and $v_j \notin IH(v_i)$ (or $v_i \in IH(v_j)$, then there exists a path in D_1 connecting u_j to u_i and a path in D_2 connecting v_i to v_j. These two paths together with arcs a_i and b_j form a directed cycle C in G. □

Based on the above properties, for each directed cycle C, we can find a special arc pair (a_i, b_j) in C, which is the intersection of a cycle (that may or may not be C) and the arc set $(A_{12} \cup A_{21})$. Hence, we can construct a bipartite graph as follows: let $B = (U \cup V, E)$ be a bipartite graph with two independent vertex sets U and V. E is the set of edges in B. Let $G = (V_D, A_D \cup A_U)$ be defined as above. Each arc $a_i \in A_{12}$ in the graph G corresponds to a vertex a_i in U with weight $w(a_i)$ and each arc $b_j \in A_{21}$ corresponds to a vertex b_j in V with weight $w(b_j)$. An edge $(a_i, b_j) \in E$ is in B iff there is a directed cycle C satisfying that $C \cap (A_{12} \cup A_{21}) = \{a_i, b_j\}$ in the graph G. In other words, an edge $(a_i, b_j) \in E$ is in B iff $u_j \in IH(u_i)$ (or $u_i \notin IH(u_j)$) and $v_j \notin IH(v_i)$ (or $v_i \in IH(v_j)$ in G, where $a_i = (u_i, v_i)$ and $b_j = (v_j, u_j)$ in G.

Property 5 *Let the graph $G = (V_D, A_D \cup A_U)$ and the bipartite graph B be defined as above. Then, resolving RIV in G is equivalent to finding the minimum weight vertex cover in B.*

Proof. By construction of the bipartite graph B, each edge in B corresponds to at least one directed cycle in G. Resolving RIV in G amounts to delete some arcs in $(A_{12} \cup A_{21})$ with minimum sum of weights. A vertex cover in B corresponds to a set of arcs which cover all directed cycles in G. Hence, the minimum weight vertex cover corresponds to the minimum sum of weighted arcs that should be deleted from G in order to resolve RIV. □

In the following, our focus is on solving the minimum vertex cover problem in the bipartite graph B. Based on graph theory, we show a 1-1 correspondence between minimum vertex cover and minimum cut in a bipartite graph. In order to state this property, we need to introduce a few definitions and constructions.

Based on the undirected bipartite graph $B = (U \cup V, E)$, let us define $\vec{E}$ as a directed version of E, and let us construct a directed network $N = (U \cup V \cup \{s, t\}, A)$, where s is the source vertex, t is the sink vertex, $A = \{(s, a_i), (b_j, t)\} \cup \vec{E}$, $a_i \in U, b_j \in V$; an arc $(a_i, b_j) \in \vec{E}$ is in N iff an edge $(a_i, b_j) \in E$ is in B for $i, j = 1, 2, 3, ...$ The capacity for each arc in N is defined as follows: $C(s, a_i) = w(a_i), C(b_j, t) = w(b_j), C(a_i, b_j) = \infty, i, j = 1, 2, 3, ...$

Definition 9 ($s - t$ cut): A cut of a directed graph $G = (V, A)$ is a partition of the vertices set V into two parts, S and $\bar{S} = V - S$. Each cut defines a set of arcs consisting of those arcs that have one endpoint in S and another endpoint in $\bar{S}$. An $s - t$ cut is defined with respect to two distinguished vertices s and t, and is a cut $(S, \bar{S})$ satisfying the property that $s \in S$ and $t \in \bar{S}$.

Definition 10 (Capacity of a $s - t$ cut): The capacity $C(S, \bar{S})$ of an s-t cut $(S, \bar{S})$ is defined as the sum of the capacities of the arcs from S to $\bar{S}$ in the cut. That is, $C(S, \bar{S}) = \sum_{(u,v) \in (S, \bar{S})} C(u, v)$.

The network flow and the cut have the following relation which is useful for our purpose.

Max-flow min-cut theorem [26]: The maximum value of the flow from a source vertex s to a sink vertex t in a capacitated network equals the minimum capacity among all $s - t$ cuts.

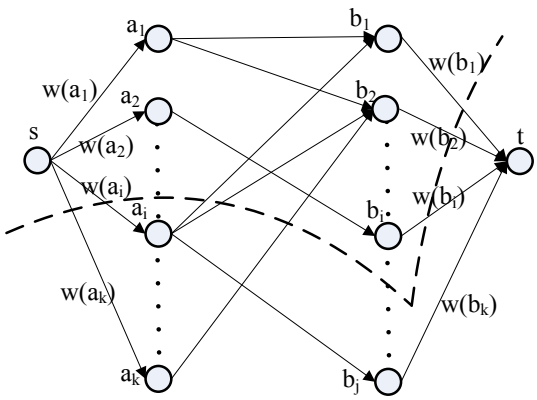

Fig. 4: A $s-t$ cut and the vertex cover for graph B.

The following two properties give a 1-1 correspondence between $s-t$ cut and vertex cover. Consequently, based on the minimum $s-t$ cut, a minimum weighted vertex cover can be found.

Property 6 (Fig. 4) *Let $B = (U \cup V, E)$ and $N = (U \cup V \cup \{s,t\}, A)$ be defined as above. Let $(S, \bar{S})$ be a $s-t$ cut in N satisfying $(S, \bar{S}) \cap \vec{E} = \emptyset$ and let $cov = (U \backslash S) \cup (V \cap S)$. Then cov is a vertex cover of B and the weight $w(cov) = C(S, \bar{S})$, where $C(S, \bar{S})$ is the capacity of the cut $(S, \bar{S})$.*

Proof. Let us first prove that cov is a vertex cover of B: since $U \backslash S \subset cov$, all arcs associated with $U \backslash S$ are covered by cov. Other arcs that are associated with $U \cap S$ must also be associated with $V \cap S$ because the $s-t$ cut does not include the arcs in $\vec{E}$. Hence all arcs associated with U are covered by cov and cov is a vertex cover of B. In the following, we prove $w(cov) = C(S, \bar{S})$: (a) $\forall u \in U \backslash S, (s,u)$ is a cut edge and $C(s,u) = w(u)$; (b) $\forall u \in V \cap S, (u,t)$ is a cut edge and $C(u,t) = w(u)$. There are no other cut edges and hence $w(cov) = C(S, \bar{S})$.□

Fig. 8 gives an illustrative example. For clarity, the abbreviated vertices and arcs are not mentioned below. It has a $s-t$ cut $(S, \bar{S})$, where $S = \{s, a_1, a_2, b_1, b_2, b_i\}$ which are above the block broken line and $\bar{S} = \{a_i, a_k, b_j\}$ which are under the block broken line. Then $cov = \{a_i, a_k, b_1, b_2, b_i\}$ is a vertex cover of B and $w(cov) = w(a_i) + w(a_k) + w(b_1) + w(b_2) + w(b_i) = C(S, \bar{S})$.

Property 7 *Let $B = (U \cup V, E)$ and $N = (U \cup V \cup \{s,t\}, A)$ be defined as above. Then $(S, \bar{S})$ is a minimum $s-t$ cut in N iff $cov = (U \setminus S) \cup (V \cap S)$ is a minimum weighted vertex cover in B.*

Proof. We just need to prove 1) the minimum $s-t$ cut satisfies $(S, \bar{S}) \cap \vec{E} = \emptyset$ and hence it corresponds to a vertex cover based on Property 6; and 2) for any vertex cover $cov = U' \cup V'$, where $U' \subseteq U, V' \subseteq V$ in a bipartite graph $B = (U \cup V, E)$, it corresponds to a $s-t$ cut $(S, \bar{S})$ satisfying $(S, \bar{S}) \cap \vec{E} = \emptyset$. The following are the proof details: 1) suppose $(S, \bar{S}) \cap \vec{E} \neq \emptyset$. Based on max-flow min-cut theorem [26], the max-flow for the network N is infinite since the capacity of each arc in $\vec{E}$ is infinite. This contradicts the fact that the max-flow cannot exceed $\sum_i w(a_i)$ based on the construction of N. 2) Let $S = \{s\} \cup U/U' \cup V'$ and $\bar{S} = U' \cup V/V' \cup \{t\}$. Then $(S, \bar{S})$ is a $s-t$ cut. In the following, we prove $(S, \bar{S}) \cap \vec{E} = \emptyset$. Let us consider an arc $(a_i, b_j) \in (S, \bar{S}) \cap \vec{E}$. Then $a_i \in U/U'$ and $b_j \in V/V'$ and the arc (a_i, b_j) cannot be covered by $cov = U' \cup V'$, which contradicts the assumption that cov is a vertex cover of B. From both 1) and 2), it is clear that the weight of cov is equal to the capacity of the corresponding cut based on the proof of Property 6. Hence, $(S, \bar{S})$ is a minimum $s-t$ cut in N iff $cov = (U \setminus S) \cup (V \cap S)$ is the minimum weighted vertex cover in B. □

Based on Properties 6 and 7, finding a minimum weight vertex cover for B is finding a minimum $s-t$ cut in network N. Hence, resolving RIV is converted to solving a minimum $s-t$ cut problem.

It is well-known that any max-flow algorithm can be applied for finding minimum cuts in a network since they are dual problems [26]. Some max-flow algorithms with polynomial-time complexity can be found in [24]. Currently, the time complexity for the fastest max-flow algorithm is somewhat higher than $O(mn)$. For example, the Goldberg-Tarjan algorithm [27] has a time complexity $O(mnlog(n^2/m))$ and excess scaling algorithm has a complexity $O(n^2m^{0.5})$ [24], where m is the number of arcs and n is the number of vertices in the network N.

In this section, we assume that there are weights for each arc in $A_{12} \cup A_{21}$ in order to give a more general solution. When all the arc weights are equal to 1, the problem is much easier to solve. In this case, the minimum vertex cover in a bipartite graph can be solved by applying max-matching algorithms [28] with time complexity $O(mn)$, where m is the number of edges and n is the number of vertices in the bipartite graph B.

To summarize, the steps for designing a secure interoperation policy based on two domains D_1 and D_2 are as follows:

1) Formulate role inheritance graph for each local policy G_{D_i} (we assume the local policies are secure) and for each vertex $v \in V_{D_i}$, find the shortest path tree $T_{D_i}(v), i = 1, 2$ rooted at v based on breadth first search algorithm [24];
2) Based on the cross mapping between the two domains D_1 and D_2, formulate the policy interoperation graph G_D and for each vertex $v \in V_D$ in G_D, find the shortest path tree $T_D(v)$ rooted at vertex v based on breadth first search algorithm [24], and let

$IH(v) = V_{T_D}(v) - V_{T_{D_i}}(v), i = 1, 2;$

3) Based on Property 3 and Property 4, find all arc pairs (a_i, b_i) (cyclic inheritance detection algorithm (Fig. 2) is applied) and construct the weighted bipartite graph $B = (U \cup V, E)$;
4) Based on $B = (U \cup V, E)$, construct the network $N = (\{s, t\} \cup U \cup V, A)$; by applying max flow algorithm for network N, find a minimum $s-t$ cut $(S, \bar{S})$; then $cov = (U \setminus S) \cup (V \cap S)$ is the minimum weighted vertex cover of B.
5) Delete those arcs in cov from the interoperation graph G_D, and a policy without inheritance violation is obtained.

More generally, when there are more than two interoperation domains, the method consists in adding the considered domains iteratively. Hence, the approach is applicable for flexible and large-scale systems.

B. Specification of role-privilege assignment and inheritance

In a policy, each role is assigned with some privileges of accessing some objects (we use object to denote both data and other resources indifferently). In this paper, we use colored Petri net to specify the privilege assignment.

Fig. 5(a) is the Petri net model for a single role-privilege assignment, where place l_i is the entry place of role r_i, if it is initially marked, meaning that role r_i is available. When transition e_i is fired, place r_i gets a new token, meaning that role r_i is ready to be activated. The resource place O represents the objects, where different kinds of objects are represented with different kinds of colored tokens. The number for each kind of object o has an upper bound b_o, meaning that the object o can be shared by at most b_o subjects simultaneously. In the corresponding CPN specification, the number for each color of tokens o has an upper bound b_o. The weight d_i for the arc (O, t_i) is a vector of colors, representing the objects to which role r_i is requested to access. For example, let us suppose that role r_i has a privilege of accessing 3 patients' records data d_{red} in an hospital domain and 2 bank records data d_{blue} in a bank domain. Then $d_i \subseteq \{3d_{red}, 2d_{blue}\}$. If place a_i is marked, it means that the requested objects are being processed by role r_i. Once role r_i finished accessing the resource, the exit transition x_i is fired, it releases all requested objects and the exit place p_i is marked. Firing transition f_i can release the role r_i.

When role r_i inherits the permissions of role r_j, we compose the two CPN models R_i and R_j by adding an arc from transition f_i to place l_j. For example, in Fig. 5(b), r_i inherits the permissions of r_j and r_k, for the specification of inheritance, two arcs (f_i, l_j) and (f_i, l_k) are added to compose the three role-privilege Petri net models. The resultant model is shown in Fig. 5(c).

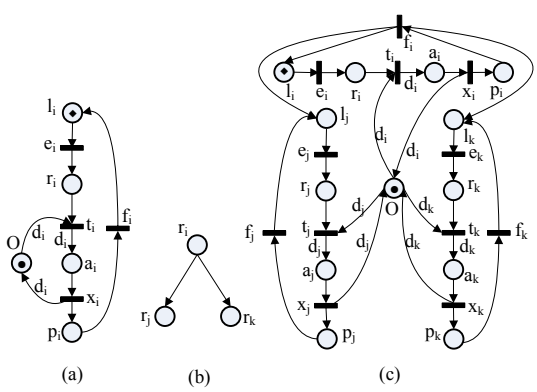

Fig. 5: The role-privilege assignment and role inheritance.

More generally, for the interoperation policy design, the roles in the system are composed by refining each role node in the inheritance-conflict-free interoperation graph (created in Section III-A) with its corresponding Petri net model. After that, the resource place O in each role-privilege Petri net model will be merged into a single resource place in the resultant system. For example, the specification of the role-privilege assignment and resource sharing for the interoperation graph in Fig. 5(b) is shown in Fig. 5(c). More technical details will be introduced in Section III-D.

C. Specification of cardinality, SoD and violation resolution

In an access control system, each user may have many qualified roles but usually not all qualified roles can be assigned to this user simultaneously due to cardinality constraints.

Definition 11 (cardinality violation): Each user u_i can assign with at most u_{si} roles, each role r_j can be assigned to at most u_{jt} users and each resource object O can be accessed by at most k users simultaneously. If any one of these constraints is not satisfied, a cardinality violation occurs in the interoperation policy.

In order to solve the cardinality violation, the following Petri net based specification is applied: based on the user-role mapping, let us suppose that user u_i is assigned with role r_j. In the corresponding Petri net model R_j for the role-privilege assignment specification, we add an input place u_i to the entry transition e_j (Fig. 6(a)), such that place u_i is initially marked with u_{si} tokens, representing that user u_i can be assigned with u_{si} roles. Similarly, we initially assign u_{jt} tokens in the entry place l_j for representing that role r_j can be assigned to u_{jt} users.

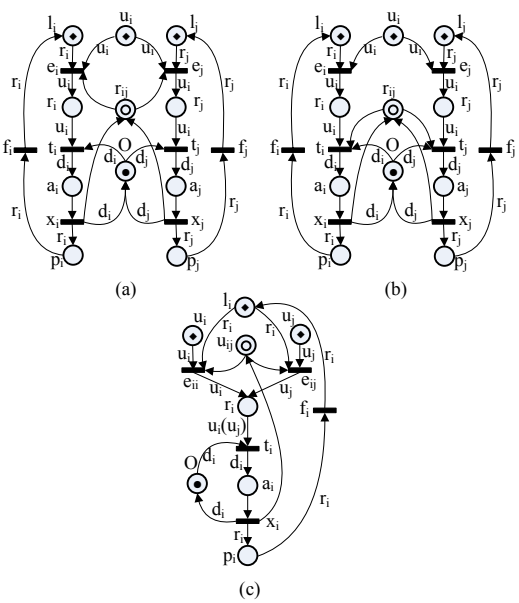

Fig. 6: Modeling SoD.

When two (resp., more) users u_i and u_j are assigned with an identical role r_i, there are two (resp., more) entry transitions e_{ii} and e_{ij} which are associated to the entry place l_i and the role place r_i. The user places u_i and u_j are associated to the two (resp., more) entry transitions respectively as input places (Fig. 6(c)).

According to the specification of resource cardinality, we use k identical colored tokens to represent that the corresponding resource object can be shared by at most k users simultaneously. For example, if there are four types of objects a, b, c, and d in the policy, and if the cardinality for these objects are 1, 2, 3, and 4 respectively, then the resource place is initially marked with 1 token of color a, 2 tokens of color b, 3 tokens of color c and 4 tokens of color d.

For user-role assignment, the following separation of duty may result in SoD violation.

Definition 12 (Separation of Duty (SoD)): Static separation of duty means that two conflicting roles cannot be assigned to a single user simultaneously. Dynamic separation of duty means that two roles with conflicting permissions cannot be activated simultaneously in a session. A user-based separation of duty means that two conflicting users cannot access a same role at the same session.

The Petri net model in Fig. 6 can be applied for the specification of the SoD violation resolution. In Fig. 6(a), a single user u_i cannot be assigned to two conflicting roles r_i and r_j. In the Petri net specification, we use a control place r_{ij} initially marked with a normal token associated to the assignment transitions e_i and e_j, respectively. Even if user u_i, roles r_i and r_j are available, only one of e_i or e_j can be fired and hence only one of the two roles can be assigned to user u_i.

Fig. 6(b) is a specification of dynamic SoD resolution. It is similar to Fig. 6(a). The difference is that the control place r_{ij} is associated to the activation transitions t_i and t_j. Even if user u_i, roles r_i and r_j are available, only one of t_i or t_j can be fired and hence only one of the two roles with conflicting permissions can be activated in a session.

Finally, Fig. 6(c) is a specification of user-based SoD. Let us suppose that u_i and u_j are two conflicting users and cannot be assigned to a same role r_i. For the two entry transitions e_{ii} and e_{ij}, an additional control place u_{ij} with an initially marked normal token is created. Initially places l_i, u_i and u_j are marked with many tokens based on their respective cardinality. Even if the users u_i, u_j and the role r_i are available, only one of transitions e_{ii} and e_{ij} can be fired and hence only one of the users u_i and u_j can be assigned to role r_i in a session.

The legend of the places and transitions are presented in TABLE I.

TABLE I: Legend for Fig. 5 and Fig. 6

l_i, l_j, l_k	role places
r_i, r_j, r_k	roles with users assigned
a_i, a_j, a_k	state of processing objects
p_i, p_j, p_k	state of deactivating roles
u_i, u_j, u_k	user places
r_{ij}, u_{ij}	control places
O	object place
$e_i, e_j, e_k, e_{ii}, e_{ij}$	user-role assignment
t_i, t_j, t_k	activate roles and access privileges
x_i, x_j, x_k	deactivate roles and release objects
f_i, f_j, f_k	release roles

The separation of duty problems can be generally extended and resolved as follows: if the number of conflicting roles (resp., users, permissions) is k, then the control place will be associated to the corresponding k assignment or activation transitions. More generally, if there are n conflicting roles (resp., users, permissions) and k of them cannot be assigned or activated simultaneously, then the control place with initially marked k ordinary tokens will be associated to the corresponding n transitions.

D. Specification of resources sharing and deadlock recovering

In the Petri net model created in the above steps, there are some identical places, e.g., the user places. If a user u_i with cardinality k is qualified with roles r_i, r_j and r_k, then there is a user place u_i initially marked with k tokens

that appears three times in the Petri net model. In order to simplify the Petri net model and implement the cardinality requirement, these identical places should be merged into a single place.

For the specification of resources sharing, a similar technique is applied. That is, the places representing the resources should be merged into a single resource place O. After applying place merging, each role will be associated with the resource place O. This does not imply each role can access all the objects in O. The privilege d_i of each role r_i is assigned in the corresponding arc (O, t_i) between O and the accessing transition t_i. Only when the requested object o_i belongs to d_i and is available in O, t_i can be fired and o_i is accessible. For specifying a cardinality constraint of sharing an object, we use an upper bound for each object. For example, if an object o_i can be shared by at most k_i users simultaneously, then there are k_i identical colored tokens, say, red tokens, in O for representing the object o_i. When there is no cardinality constraint for the object o_j, its upper bound should be large enough, say mn, where m is the number of users and n is the number of roles. Hence, the initial marking of the resource place O is a vector of integers, each item representing the sharing cardinality of the corresponding object. For a reachable marking M in the Petri net model, $M(O)$ is a vector of integers and each item represents the number of remaining objects which can be shared simultaneously.

Note that resources-sharing is a hard problem in many systems design, e.g., in manufacturing system, because it may result in deadlocks. In our interoperation policy design, the deadlock will not occur if there are no controlling relations between the objects (resources) and consequently the inducement "circular waiting" will not occur. When there are some controlling relations between the objects, e.g., accessing object o_i should preprocess object o_j, then deadlock may occur. For example, in Fig. 7, there is a single object o_i and a single object o_j in the resource place O. Let us assume that a user u_i requests o_j, and before accessing o_j, he should preprocess an object o_i; at the same time, a user u_j requests o_i, but before accessing o_i, he should preprocessthe object o_j. In this case, a "circular waiting" may appear and there is a deadlock. For solving deadlock problems, siphon-trap technology [29], [30] and adding control places [31] are efficient methods. The following is a methodology for detecting and removing deadlocks by adding control places in a resource sharing policy design.

Deadlock-free Design

Input: n Petri nets for n roles

Output: A compositional deadlock-free model for these n Petri nets

1) For each pair of Petri nets which have common resource places, do place merging

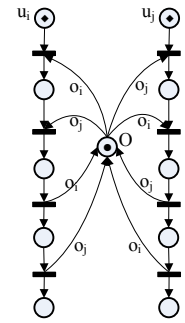

Fig. 7: An example of deadlock for resource sharing.

2) For the compositional net, calculate the next reachable marking $MR(k+1)$ from the initial marking $M(0)$ by $M_R(k+1)^T = M(k)^T L^*_{VP}$.
   If $M_R(k+1) = M_f$, output "There is no deadlock in the two nets" and goto Step 5);
   else find out the dead marking and record the dead transitions and goto Step 3);
3) Add a control place for the dead transitions in this compositional net;
4) Change the weighted place transitive matrix, and check dead marking and dead transitions again
   if there are also dead transitions, continue to add control place until the net is deadlock-free;
   else output "There is no deadlock in the two nets" and goto Step 5);
5) For other pairs of Petri nets sharing resources, do the same work as Step 1) to Step 4), until all the compositional nets are deadlock-free.
6) Merge these n news nets together into a large compositional one by place merging, and a deadlock-free model for the secure interoperation policy is gotten

In the above algorithm, $M_R(k+1)^T = M(k)^T L^*_{VP}$ is a transformation of marking equation based on the place-transitive matrix L^*_{VP} [32], where $M_R(k+1)$ is a vector of nonnegative integer and is a reachable marking from $M(k)$, M_f is a final marking in a role-privilege model R_i such that $M_f(p_i) > 0$. More technical details about how to add control places for solving deadlocks can be found in [31].

E. Optimization of user-role assignment

With the Petri net specification introduced in the previous subsections, a secure interoperation policy without role inheritance violation, cardinality violation, SoD violation nor system deadlock can be designed and an access request can be handled properly. In order to make the policy system more efficient, we now try to find an optimal user-role assignment such that the system can permit as many

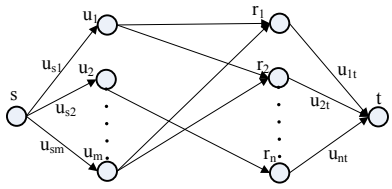

Fig. 8: The network flow model for user-role assignment.

users as possible to safely access simultaneously.

In other words, we address in this section the following problem: in a session, how to assign users to their qualified roles with as many (user, roles) pairs as possible without cardinality violation?

In order to express the problem more clearly, consider the example shown in TABLE II. A possible user-role assignment is $u_1 \rightarrow r_1, u_2 \rightarrow r_2, u_4 \rightarrow r_3, u_5 \rightarrow r_4, u_6 \rightarrow r_5, u_7 \rightarrow r_6, u_8 \rightarrow r_7$; for this assignment, there is no cardinality violation. But u_3 and u_9 have no roles and roles r_2, r_6 and r_7 have space for more users. Obviously, it is not an optimal assignment. A good policy requires as many users as possible to safely access the objects simultaneously without any conflicts.

In order to obtain a user-role assignment such that a maximum number of (user, roles) pairs can be obtained in a session, we apply the following network flow model and use a max-flow algorithm for the solution. Let us consider the network $G = (V, A)$, where V is the set of nodes and A is the set of directed arcs. Each user and each role are represented with a node in V, s is the source node and t is the sink node in the network. There is an arc connecting source node s to each user node u_i and the capacity in the arc (s, u_i) is u_{si} (cardinality of u_i); there is an arc connecting each role node r_j to the sink node t and the capacity in the arc (r_j, t) is u_{jt} (cardinality of r_j). There is an arc with capacity 1 connecting user u_i to role r_j iff user u_i is qualified with role r_j in the interoperation policy (Fig. 8).

Based on the network model, the user-role assignment problem is to solve the following max-flow problem:

Given $G = (V, A)$ with

$$V = \{s, u_i, r_j, t\}, A = \{(s, u_i), (u_i, r_j), (r_j, t)\}$$

where $i = 1, 2, \cdots, m$, $j = 1, 2, \cdots, n$.
Let
x_{ij} = flow on arc (u_i, r_j);
x_{si} = flow on arc (s, u_i);
x_{jt} = flow on arc (r_j, t);
u_{si} = capacity of flow in arc (s, u_i);
u_{jt} = capacity of flow in arc (r_j, t);
s = source node and t = sink node

Find a maximal f under the following constraints:

$$\begin{cases} \sum_{i=1}^{m} x_{si} = f \\ \sum_{j=1}^{n} x_{jt} = f \\ x_{si} - \sum_{j=1}^{n} x_{ij} = 0, \forall i \\ x_{jt} - \sum_{i=1}^{m} x_{ij} = 0, \forall j \\ 0 \leqslant x_{si} \leqslant u_{si} \\ 0 \leqslant x_{jt} \leqslant u_{jt} \\ x_{ij} \in \{0, 1\} \end{cases} \quad (1)$$

The solution for the above max-flow problem is a maximum user-role mapping such that there is a maximum number of roles which can be assigned with users without cardinality conflict. Note that the above integer programming model is just a description of the max-flow problem, and our focus is not to find an approximation algorithm for solving the 0-1 integer programming problem. In fact, our solution is based on the network model shown in Fig. 8. For saving space, the max-flow algorithms will not be introduced here and the interested reader can refer to any textbook about network flow optimization or programming. For instance, the Ford-Fulkerson max flow algorithm, the capacity scaling algorithm and the preflow push algorithm [24] are all applicable for solving this problem.

With the technique introduced in this section, for a given session, an optimal user-role assignment is obtained. Based on an optimal max-flow solution, we can select which roles must be activated by which users in order to permit as many users as possible to safely access objects simultaneously.

F. Specification of dynamic environment

The main characteristic of a dynamic environment of interoperation policy design is to allow the addition of new roles (resp., users, domains) in the collaborative system and the removal of roles (resp., users, domains) from collaboration, as well as sometimes the evolution of local domains access policy.

Based on our approach, each role is designed independently with a CPN, the privileges and users are specified as tokens and assigned to their specific role based on this CPN model. When a new role is added, its corresponding CPN model is integrated into the existing system based on the role inheritance hierarchy; when a role is removed from collaboration, the corresponding subnet can be deleted from the system. Both cases will not affect the remaining

part of the system. Addition and removing of domains can be handled similarly. Since privileges and users are specified as tokens in the CPN, addition and removing of users are processed inside a single subnet and will not affect other parts of the system, either. When an evolution for a local domain's policy is necessary, the corresponding subnet will be handled independently.

IV. Verification of the Interoperation Policy Design

This section considers the verification of conflict and policy consistency for the secure interoperation policy design. State equations and reachability techniques are simply introduced.

A. Conflict verification

Based on the technique introduced in Section III-A, for an inheritance graph G_{D_1}, if role u is senior to role v, then it is also true in the interoperation graph G_D since all the arcs in A_{D_1} remain in G_D; if role u is not senior to role v in the local domain D_1, it still is not senior in the interoperation policy of D, since all those violation arcs are deleted in the algorithm. Hence G_D has no role inheritance violation, and both the autonomy principle and security principle are preserved.

For cardinality specification, the number of tokens in the specific place can guarantee the concurrent firing times in the Petri net model based on the firing rule.

As for SoD violation, which is introduced in Section III-C, based on the Petri net firing rule, the single token in the control place guarantees that the two violation transitions cannot be fired simultaneously, hence obviously the violation can be dealt properly. In the generalized case, there may be n conflicting roles (resp., users and permission) and k of them cannot be assigned or activated simultaneously. Since the control place is initially marked with k tokens and associated with these n conflicting transitions, at most k of them can be fired simultaneously and the SoD problem can be resolved.

B. Reachability verification

With the above steps, an interoperation policy is specified with a colored Petri net. The request-decision process (u_i, d_j) (user u_i requests to access object d_j) amounts to find a firing sequence σ in the Petri net model such that $M_i[N, \sigma\rangle M_j$, where $M_i(u_i) > 0$ and there exists a place p_j satisfying $M_j(p_j) \geqslant d_j$. The following is the analysis and verification details for a specific request.

In a Petri net (N, M_0), any reachable marking M satisfies the state equation [21]. In other words, if the state equation is not true, then the corresponding marking M is not reachable. Hence we have the following property for an interoperation Petri net model.

Property 8 *For a specific request (u_i, d_j) and all the reachable markings M_i and M_j satisfying that $M_i(u_i) > 0$ and $M_j(p_j) \geqslant d_j$ in the interoperation model (N, M_0), if the state equation $M_j = M_i + V\mu$ is not true, then the requested resource is not accessible.*

Proof. If the state equation $M_j = M_i + V\mu$ is not true, then the marking M_j is not reachable form M_i. Correspondingly, there is no path from place u_i to place p_j and hence user u_i cannot access object d_j in the interoperation policy. □

Property 9 *For a specific request (u_i, d_j), let us assume that u_i is qualified to the roles in the set R_i and the resource d_j is included in the privilege of role set R_j. If for any $r_i \in R_i$ and $r_j \in R_j$, there does not exist a directed path from r_i to r_j in the interoperation graph G_D, then the requested resource is not accessible.*

Proof. If the requested resource is accessible, then there exists a firing sequence σ in the Petri net model such that $M_i[N, \sigma\rangle M_j$, where $M_i(u_i) > 0$ and there exists a place p_j satisfying $M_j(p_j) \geqslant d_j$; the firing sequence σ is a transition sequence including some transitions in $r_i \in R_i$ and $r_j \in R_j$. Therefore in the interoperation graph there exists a directed path from r_i to r_j, which leads to a contradiction. □

For a request (u_i, d_j), if either Property 8 or Property 9 is true, then the request is not accessible. If none of the properties is true, the following reachability tree technique will be applied for verification.

A reachability tree is a tree rooted at the initial marking M_0, such that each node in the tree is a reachable marking of the Petri net (N, M_0). For two nodes M_i and M_j in the tree, there is an arc from M_i to M_j iff there is a transition t satisfying $M_i[N, t\rangle M_j$ in the Petri net (N, M_0).

Generally, the reachability tree may encounter the space explosion problem. Fortunately, for our interoperation policy design, each role-privilege assignment Petri net is bounded (the structure is simple, this is easy to verify). After applying place-merging, based on the theoretical results shown in [33], the resultant Petri net is still bounded. As a consequence, the corresponding reachability tree is finite. Hence, for a request (u_i, d_j), if there is a firing sequence σ in the Petri net model such that $M_i[N, \sigma\rangle M_j$, where $M_i(u_i) > 0$ and there exists a place p_j satisfying $M_j(p_j) \geqslant d_j$, M_i and M_j are reachable markings in the Petri net model (N, M_0), then the requested resource is accessible, otherwise it is not accessible.

V. Illustrative Example

The example in this section contains two domains, one is office staff and another is medical staff. In the office staff domain, there are five roles: office manager (r_1), medical representative (r_2), secretary (r_3), accounts payable (r_4) and purchasing (r_5), where payable and purchasing have a separation of duty constraints; and the medical staff domain contains two roles, i.e., doctor (r_6) and nurse (r_7) and they have separation of duty constraints. There are 9 users (denoted as $u_i, i = 1, 2, \cdots, 8, 9$) in the system, where users u_1 and u_3 have a separation of duty constraint. For simplicity, we suppose that the cardinality for all the users is 2. That is, each user can have at most 2 roles in a session. The access privileges for each role, the users' possible assignment and role-cardinalities are shown in TABLE II. There are four types of objects: the prescription records, medicine information, financial information and patients' personal information ; we denote these objects as a, b, c and d respectively. Suppose prescription record data a can be shared by at most 2 users simultaneously, object b can be simultaneously shared by 2 users, object c can be accessed by only one user and object d cannot be shared by two users.

The interoperation rule is as follows: in the office domain, the secretary (r_3) has access to medical domain by inheriting the permissions of the doctor (r_6); in the medical domain, the doctor (r_6) and the nurse (r_7) have access to the office domain by inheriting the permissions of the office manager (r_1) and the secretary (r_3), respectively.

TABLE II: Constraint information for the example

roles	privilege	Possible users	cardinality
Office manager (r_1)	$d_1 = \{a, b\}$	u_1, u_2	1
Medical-representative (r_2)	$d_2 = \{a, b\}$	u_2, u_3	2
Secretary (r_3)	$d_3 = \{b, c\}$	u_1, u_2, u_4	1
Accounts payable (r_4)	$d_4 = \{c\}$	u_5	1
Purchasing (r_5)	$d_5 = \{c\}$	u_6	1
Doctor (r_6)	$d_6 = \{a, b, d\}$	u_7, u_9	2
Nurse (r_7)	$d_7 = \{d\}$	u_7, u_8	2

Step 1. Based on the senior-junior relation between these roles, an inheritance graph G_D for the interoperation domain is illustrated in Fig. 9. After applying the shortest path tree generating algorithm to the graph G_D, we get $IH(r_3) = \{r_1\}, IH(r_7) = \{r_6\}$ and $IH(r_i) = \emptyset$ for $i = 2, 3, 4, 5, 7$. Since $r_3 \notin V_{T_{D_1}}(r_1)$ and $r_7 \notin V_{T_{D_2}}(r_6)$, arcs (r_1, r_3) and (r_6, r_7) are added to the graph G_D (Fig. 9) to create directed cycles. For the three cross domain arcs, the weights are $e(r_6, r_1) = 2, e(r_7, r_3) = 3$ and $e(r_3, r_6) = 1$, respectively. Then based on the Resolve RIV algorithm, the arc (r_3, r_6) is deleted from G_D and an inheritance-conflict-free graph G is obtained (Fig. 9).

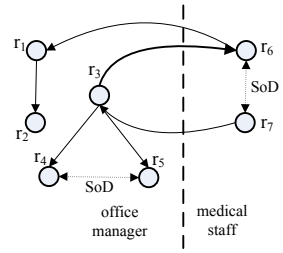

Fig. 9: Generating conflict-free inheritance graph.

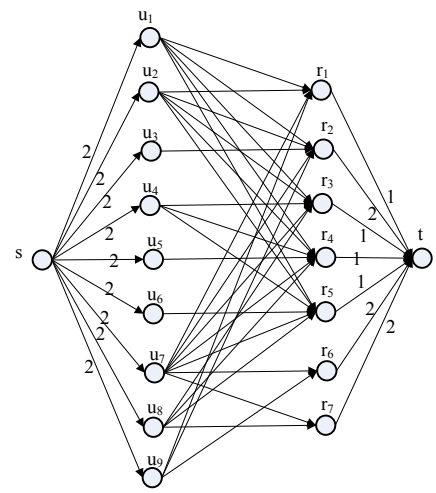

Fig. 10: The max-flow network model for user-role assignment.

Step 2. For each role r_i, the colored Petri net model R_i for specifying the role-privilege assignment is created (refer to Fig. 5(a) and Fig. 11), l_i is initially marked with 1 or 2 tokens according to its cardinality shown in TABLE II.

Step 3. Based on the user-role assignment shown in TABLE II and the inheritance-conflict-free graph G (Fig. 9), the max-flow model (Fig. 10) is obtained as follows: if user u is qualified with role r_i which is senior to r_j, then there are arcs (u, r_i) and (u, r_j) in the network model. Based on the pre-flow push algorithm [24], we get the following user-role assignment such that the maximum number of (user, role) pairs is available. In this example, there are at most 10 users (including identical users) who can be assigned with roles simultaneously based on the role cardinality shown in TABLE II. All the optimal solutions are shown in TABLE III, we select the first solution for illustrating the user-role assignment specification in order to get an efficient interoperation policy design.

Based on the system requirement, there are SoD constraints between users u_1 and u_3, roles r_4 and r_5 and roles r_6 and r_7. Hence, the user-role assignment model (Fig. 11) for composing R_6 and R_7 will have an additional control

TABLE III: All the optimal user-role assignment

r_1	r_2	r_3	r_4	r_5	r_6	r_7
u_1	u_2, u_3	u_4	u_5	u_6	u_7, u_9	u_7, u_8
u_1	u_2, u_3	u_2	u_5	u_6	u_7, u_9	u_7, u_8
u_2	u_2, u_3	u_1	u_5	u_6	u_7, u_9	u_7, u_8
u_2	u_2, u_3	u_4	u_5	u_6	u_7, u_9	u_7, u_8

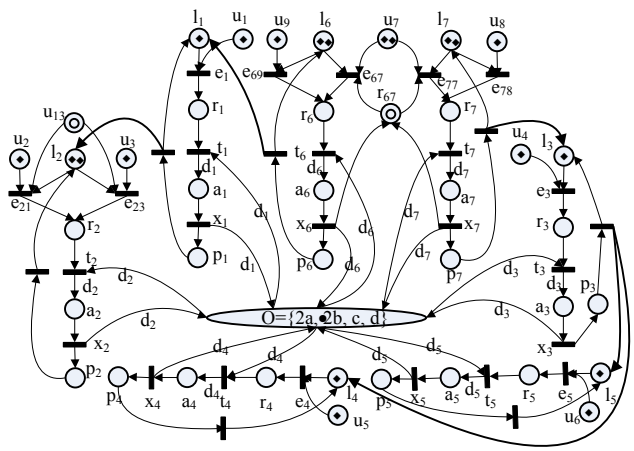

Fig. 11: The colored Petri net specification of the secure interoperation policy.

place r_{67} and the CPN model R_2 has an additional control place u_{13} (Fig. 11). For this example, in order to simplify the structure of the Petri net model, we do not add a control place for composing R_4 and R_5 although there are conflicting constraints between roles r_4 and r_5. The reason is that there are no common users which can be assigned to both r_4 and r_5 based on the information shown in TABLE-II. At last, all the role-privilege assignment models R_i are composed by merging the identical places and the resultant CPN model N for the secure interoperation policy specification is shown in Fig. 11.

The request (u_i, o_j), meaning that user u_i requests to access object $o_j \in O = \{2a, 2b, c, d\}$, is processed as follow: for a marking M_i satisfying $M_i(u_i) > 0$, if there exists a firing sequence σ and a place p_i such that $M_i[N, \sigma\rangle M$, and $o_j \in M(p_i)$, then the decision for the request is "permit", otherwise, the decision is "reject". For example, in Fig. 11, suppose current state is "user u_8 is accessing $d_7 = \{d\}$ and user u_1 is dealing with $d_1 = \{a, b\}$". Then the current marking M_i satisfies that $M_i(u_8) = M_i(u_1) = 0, M_i(a_7) = d$ and $M_i(a_1) = \{a, b\}$. For a new request (u_7, d), there exists a firing sequence $\sigma = e_{77}x_7t_7x_7$ such that $M_i[N, \sigma\rangle M$ satisfies $M(p_7) = d$. It means that the object d is not accessible in O (currently occupied by user u_8 and transition t_7 is not firable) until user u_8 finishes its accessing request and releases the object d (firing transition x_7), since d cannot be shared by two users simultaneously based on the policy requirements. Another new request (u_3, a) is immediately permitted since there exists a firing sequence $\sigma = e_{23}t_2x_2$ such that $M_i[N, \sigma\rangle M$ satisfying that $M(p_2) = a$.

VI. Conclusion and Future Work

This paper mainly considers the conflict resolution problem for a secure interoperation policy design. The contribution and the approach are briefly summarized as follows: globally, the roles and their relations in the interoperation environment are modeled as a directed interoperation graph, based on which, role inheritance violations are detected and resolved using a resolve RIV algorithm, which selects to delete some inheritance relations with the minimum inheritance weight. The resulting graph G does not contain role inheritance violation anymore. Then, for each role r_i in domain D, its privileges are assigned based on a colored Petri net model R_i. Cardinalities are specified as numerical restrictions for the initially marked tokens in the corresponding places in R_i. Next, each role r_i is refined with its corresponding model R_i to the inheritance graph G, and the separation duty constraints are considered by adding control places after the refinement. The user-role assignment problem is modeled as a max-flow problem and solved by a mature max-flow algorithm, e.g, preflow push algorithm [24], in order to give an optimal user-role assignment for each session; at last, the resource sharing problem is specified by resource place merging. When deadlock occurs, the mature siphon-trap technology and addition of control place technology can be applied. Finally, based on the resulting Petri net model, handling a request is to search for a firing sequence in the bounded Petri net model. State equation technology and reachability tree technology can be applied for the decision-making and verification.

Since multi-domains environments are going to change frequently, policies integration may face to much more complex conflicts, that may be for example related to time constraints and partial resource sharing. Finding other technologies for resolving such complex conflicts, and efficient verification techniques for verifying the consistency of complex policy composition are yet challenging issues for future work.

References

[1] Dawson, S., Qian, S., Samarati, P.: Providing security and interoperation of heterogeneous systems. Distributed and Parallel Databases **8** (2000) 119–145

[2] Gong, L., Qian, X.: Computational issues in secure interoperation. IEEE Trans. on Software Engineering **22**(1) (1996)

[3] Shafiq, B., Joshi, J.B.D., Bertino, E., Ghafoor, A.: Secure interoperation in a multidomain environment employing rbac policies. IEEE Transactions on Knowledge and Data Engineering **17**(11) (2005)

[4] Walvekar, A., Kelkar, M., Smith, M., Gamble, R.: Determining conflicts in interdomain mappings for access control. In: Joint Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis (FCS-ARSPA'06). (2006)

[5] Oladimeji, E.A., Chung, L.: Analyzing security interoperability during component integration. In: Proceedings of the 5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse. (2006)

[6] Wang, X., Feng, D., Xu, Z., Hu, H.: Mediator-free secure policy interoperation of exclusively-trusted multiple domains. Lecture Notes in Computer Science **4991** (2008) 248–262

[7] Wang, X., Sun, J., Yang, X., Huang, C., Wu, D.: Security violation detection for rbac based interoperation in distributed environment. IEICE Transactions on Information and Systems **E91-D**(5) (2008) 1447–1456

[8] Bonatti, P., Vimercati, S.D.C., Samarati, P.: An algebra for composing access control policies. ACM Trans. Information and System Security **5**(1) (2002)

[9] Lupu, E., Sloman, M.: Conflicts in policy-based distributed systems management. IEEE Trans. on Software Engineering **25**(6) (1999) 852–869

[10] Joshi, J., Bertino, E., Ghafoor, A.: Temporal hierarchies and inheritance semantics for GTRBAC. In: Proc. Seventh ACM Symp. Access Control Models and Technologies. (2002) 74–83

[11] Gavrila, S.I., Barkley, J.F.: Formal specification for role based access control user/role and role/role relationship management. In: Proc. Third ACM Workshop Role-Based Access Control. (1998)

[12] Huang, H.: Enhancing the Property-Preserving Petri Net Process Algebra for Component-based System Design (with Application to Designing Multi-agent Systems and Manufacturing Systems). PhD thesis, Department of Computer Science, City University of Hong Kong (2004)

[13] Mak, W.M.: Verifying Property Preservation for Component-based Software Systems (A Petri-net Based Methodology). PhD thesis, Department of Computer Science, City University of Hong Kong (2001)

[14] Shafiq, B., Masood, A., Joshi, J., Ghafoor, A.: A role-based access control policy verification framework for real-time systems. In: Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems. (2005)

[15] Mortensen, K.H.: Automatic code generation method based on coloured petri net models applied on an access control system. Lecture Notes in Computer Science **1825** (2000) 367–386

[16] Knorr, K.: Dynamic access control through petri net workflows. In: Proceedings of 16the annual conference on Computer Security Applications. (2000) 159–167

[17] Zhang, Z.L., Hong, F., Liao, J.G.: Modeling chinese wall policy using colored petri nets. In: Proceedings of the 6th IEEE International Conference on Computer and Information Technology. (2006)

[18] Zhang, Z.L., Hong, F., Xiao, H.J.: Verification of strict integrity policy via petri nets. In: Proceedings of the International Conference on Systems and Networks Communication. (2006)

[19] Juszczyszyn, K.: Verifying enterprise's mandatory access control policies with coloured petri nets. In: Proceedings of the 12th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises. (2003)

[20] Deng, Y., Wang, J.C., Tsai, J., Beznosov, K.: An approach for modeling and analysis of security system architectures. IEEE Transactions on Knowledge and Data Engineering **15**(5) (2003) 1099–1119

[21] Murata, T.: Petri nets: Properties, analysis, and applications. Proceedings of IEEE **77**(4) (1985) 541–580

[22] Jensen, K.: Coloured petri nets: Basic concepts, analysis methods and practical use. Springer Verlag **1** (1997)

[23] Jensen, K.: Coloured petri nets: A high level language for system design and analysis. Lecture Notes in Computer Science **483** (1991) 342–416

[24] Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network flows: Theory, algorithms, and applications. In: Network Flows: Theory, Algorithms, and Applications, Prentice-Hall (1993)

[25] Huang, H., Kirchner, H., Liu, S., Wu, W.: Handling inheritance violation for secure interoperation of heterogeneous systems. to appear in International Journal of Security and Networks (2009)

[26] Ford, L.R., Fulkerson, D.R.: Maximal flow through a network. Can. J. Math **8** (1956) 399 – 404

[27] Goldberg, A., Tarjan, R.: A new approach to the maximum flow problem. Journal of the ACM **35** (1988) 921–940

[28] Brualdi, R.A.: Matchings in bipartite graph. In: Introductory Combinatorics, Prentice Hall (2004)

[29] Li, Z., Zhou, M.: Elementary siphons of petri nets and their application to deadlock prevention in flexible manufacturing systems. IEEE Transactions on System, Man, and Cybernetics **34** (2004) 38–51

[30] Li, Z., Wei, N.: Deadlock control of flexible manufacturing systems via invariant-controlled elementary siphons of petri nets. The International Journal of Advanced Manufacturing Technology **33** (2007) 24–35

[31] Huang, H., Li, J., Zhou, W.: Deadlock-free design of jsp with multi-resource sharing. In: Proceedings of the International Conference on Computer and Electrical Engineering. (2008)

[32] Liu, J., Itoh, Y., Miyazawa, I., Sekiguchi, T.: A research on petri net properties using transitive matrix. In: Proceeding IEEE SMC. (1999) 888–893

[33] Jiao, L., Huang, H., Chueng, T.Y.: Property-preserving composition by place merging. Journal of Circuits, Systems and Computers **14**(4) (2005) 793–812