CERIAS Tech Report 2004-34

SECURE INTEROPERATION IN A MULTI-DOMAIN ENVIRONMENT

by Basit Shafiq

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

SECURE INTEROPERATION IN A MULTI-DOMAIN ENVIRONMENT

A Preliminary Report

Submitted to the Faculty

of

Purdue University

by

Basit Shafiq

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

April 2004

TABLE OF CONTENTS

TABLE OF CONTENTS ii

LIST OF TABLES iv

LIST OF FIGURES v

ABSTRACT vii

1 INTRODUCTION 1

2 OVERVIEW OF MULTI-DOMAIN RBAC SYSTEM 5

2.1 Role Based Access Control (RBAC) 5
2.2 The NIST RBAC Model 7
2.3 Graph-based Specification Model for RBAC 8
2.4 Security Requirements in a Multi-domain RBAC System 10

3 MULTI-DOMAIN POLICY INTEGRATION 14

3.1 Information Sharing Policy 14
3.2 Heterogeneity Issues in Policy Integration 16
3.3 RBAC Policy Integration 19
3.3.1 Policy Integration Requirements (PIR) 19
3.3.2 RBAC Policy Integration Algorithm 22
3.3.3 Properties of RBAC-integrate 29
3.3.4 Time Complexity of RBAC-integrate 32

4 OPTIMAL CONFLICT RESOLUTION 34

4.1 IP Formulation of a Multi-Domain RBAC Policy 35
4.1.1 Constraint Transformation Rules 35
4.1.2 Optimality Criteria 37
4.2 Autonomy Consideration 38
4.3 Conflict Resolution Algorithm 41
4.4 An illustrative example 43
4.5 Verification of Multi-domain policy 54

5 CONCLUSION 59

5.1 Summary of Current Work 59
5.2 Future Work 60
5.2.1 Verification of RBAC policy specification 60
5.2.2 Policy Evolution 63
5.2.3 Autonomy and interoperability trade-off 63
5.2.4 Policy partitioning for enterprise splitting 64

6 REFERENCES 65

7 APPENDIX 71

LIST OF TABLES

Table 3.1 Functions/predicates used in this report..... 21

Table 4.1 Description of roles involved in collaboration among county offices..... 49

Table 4.2 Information sharing policy of collaborating domains..... 50

Table 4.3 Cardinality and user assignment of roles used in autonomy loss measurement of Fig. 4.5 51

LIST OF FIGURES

Fig. 1.1 Policy Integration Framework 4

Fig. 2.1 Constraints and hierarchy in RBAC 5

Fig. 2.2 RBAC type graph 9

Fig. 2.3 An example of RBAC graph 9

Fig. 2.4 A multi-domain access control policy defining interoperation between CTO and CCO 12

Fig. 2.5 Example of a cross-domain separation of duty (SoD) constraint 13

Fig. 3.1 An abstract view of inter-domain information sharing 15

Fig. 3.2. Information exchange between the County Treasurer Office and District Clerk Office 17

Fig. 3.3 Hierarchical heterogeneity 19

Fig. 3.4 Policy integration algorithm 23

Fig. 3.5 Procedures used by Role-Integrate during Policy Integration 24

Fig. 3.6 Example of induced SoD 28

Fig. 4.1 IP formulation of multi-domain RBAC policy shown in Fig. 4.2 42

Fig. 4.2 (a) RBAC policy graph of domain A and B in example 4, (b) Integrated RBAC policy defining interoperation between domains A and B 42

Fig. 4.3 Conflict resolution algorithm 43

Fig. 4.4 RBAC policy graphs of collaborating county offices 51

Fig. 4.5 Integrated RBAC policy governing collaboration among the county offices 52

Fig. 4.6 Interoperability versus autonomy loss 53

Fig. 7.1. Cases of role-specific *SoD* violations involving cross-domain paths..... 80

Fig. 7.2. User-specific SoD violation through a cross-domain path 84

ABSTRACT

Shafiq, Basit. Ph.D., Purdue University, April 2004. Optimal Secure Interoperation in a Multi-Domain Environment. Major Professor: Arif Ghafoor.

The rapid proliferation of the Internet and the cost effective growth of its key enabling technologies such as database management systems, storage and end-systems, and networking are revolutionizing information technology and have created unprecedented opportunities for developing large scale distributed applications and enterprise-wide systems. At the same time, there is a growing need for information sharing and resource exchange in a collaborative environment that spans multiple enterprises. Various businesses, government, and other organizations have realized that information and resource sharing is becoming increasingly critical to their success. However, increase in inter-domain information and resource exchange poses new threats to the security and privacy of data. Numerous studies have shown that unauthorized access, in particular by insiders, constitutes a major security problem for enterprise application environments. This problem can get magnified in a collaborative environment where, distributed, heterogeneous, and autonomous organizations interoperate with each other. Collaboration in such a diverse environment requires integration of the access control policies of local domains to compose a global security policy for controlling information accesses across multiple domains. In this proposal, we address the issue of policy integration in a multi-domain system that allows information and resource sharing in a collaborative environment. The proposed policy integration mechanism is a two phase process that first defines a mapping among the cross-domain entities and then resolves the underlying access control policy conflicts. For conflict resolution, we propose an integer programming (IP) based approach that maximizes inter-domain information and data exchange according to some specified optimality criterion. As an

extension to the policy integration framework, we plan to address the problem of access control policy verification and policy evolution in the context of secure interoperation. In addition, we will investigate the problem of semantic partitioning of a single access control policy into multiple independent, autonomous, and functional policies.

1 INTRODUCTION

The rapid proliferation of the Internet and the cost effective growth of its key enabling technologies such as database management systems, storage and end-systems, and networking are revolutionizing information technology and have created unprecedented opportunities for developing large scale distributed applications and enterprise-wide systems. At the same time, there is a growing need for information sharing and resource exchange in a collaborative environment that spans multiple enterprises. Various businesses, government, and other organizations have realized that information and resource sharing is becoming increasingly critical to their success. In the commercial sector, companies collaborate with each other for supply chain arrangements, subcontracting relationships, or joint marketing campaigns [Coh02]. In the public sector, government has taken various initiatives to increase collaboration among government agencies and NGOs in order to provide better public service to citizens, and to make available timely, accurate, and complete information to relevant government agencies and general public. Two major projects initiated in this regard are Digital Government Program and Integrated Justice Information Systems. The aim of the Digital Government Program is to make use of information and communication technologies for empowering citizens with greater access to services and increase their involvement in decision making process, leading to improved citizen-government interaction [Elm01]. Integrated justice is an initiative taken by Department of Justice to improve information management and sharing between justice system agencies at all levels of government [IJIS]. Whether collaboration is solely among government agencies, or incorporates both government and commercial organizations, information and resource exchange beyond the individual domain boundary is crucial to meet the business requirements of organizations in today's world.

With the increase in information and data accessibility, there is a growing concern for security and privacy of data. Many studies show that unauthorized access, in particular by insiders, constitutes a major security problem for enterprise application environments [Pow00], highlighting the need for robust access control management systems. This problem can be highly magnified in a collaborative environment where distributed and heterogeneous organizations, each employing its own security policy, interoperate with each other, allowing highly intensive inter-domain accesses [Jos01b, Gon96]. Collaboration in such a diverse and heterogeneous environment requires integration of local policies to compose a global security policy that governs information and data accesses across domain boundaries. Integration of security policies, local to the collaborating domains, entails various challenges regarding reconciliation of semantic differences, secure interoperability, containment of risk propagation, and policy management etc. [Jos01b]. An access control model that can be used to uniformly represent policies of the individual domains is desirable. Such a model should allow interoperation and information sharing among multiple domains and at the same time guarantee that such inter-domain data accesses do not violate the underlying policies of constituent domains. In particular, secure interoperation should enforce the following two principles [Gon96]:

The *autonomy principle*, which states that if access is permitted within an individual system, it must also be permitted under secure interoperation

The *security principle,* which states that if an access is not permitted within an individual system, it must not be permitted under secure interoperation.

The problem of secure interoperation in a multi-domain environment has been addressed in literature in the context of multi-level security (Bell-Lapadula) model [Gon96, Bon96]. Multi-level security or Bell-Lapadula [Bel73] model is more suitable for environments which have static constraints. For instance, in multi-level security model, all accesses conform to the pre-specified security ordering. Security ordering in this case is a static constraint, even though the security labels of entities may change with time, e.g., declassification of documents after a certain period of time. In a multi-level security model, if a subject *s* with security level *a* is authorized to access an object *o* with

security level *b*, then *s* can access *o* at all times provided the security levels of *s* and *o* never change. Dynamic constraints on the other hand, may not allow subject *s* to access *o* even though their security labels remain unchanged. Separation-of-duty (SoD) and precedence constraints are example of such dynamic constraints and are required in most commercial applications including digital government, e-commerce, health-care systems, and workflow management systems [Ber99]. Traditional multi-level (LBAC) model cannot be used to capture the dynamic constraint requirements of emerging applications and information systems. *Role based access control* (RBAC) models are receiving increasing attention as a generalized approach to access control [Nya99, San98b]. Due to its inherent richness in modeling hierarchical, SoD, cardinality, and dependency constraints, RBAC is emerging as a vital access control model capable of modeling a wide range of access control policies. For the same reason, we use RBAC to express the security policies of collaborating organizations/domains.

In this report, we address the issue of policy integration in a multi-domain system that allows information and resource sharing in a collaborative environment. The policy integration mechanism described in this report is a two phase process as shown in Figure 1.1. In the first phase the role heterogeneity constraints among collaborating domains are resolved and a global access control policy is generated from the given RBAC policies and administrator specified constraints. The global policy generated in the first phase may be conflicting and may allow violation of some of the security requirements. In the second phase, conflicts are resolved by relaxing some of the access constraints. For conflict resolution, we propose an integer programming (IP) [Wol98] based approach that maximizes inter-domain information and data exchange according to some specified optimality criterion.

This report is organized as follows. In Chapter 2, we provide a brief overview of the RBAC model and discuss the basic security requirements in a multi-domain RBAC system. The policy integration phase of Figure 1.1 is described in Chapter 3. Chapter 4 describes the conflict resolution strategy and illustrates the policy integration framework through a detailed example. Chapter 5 provides a formal proof that the multi-domain policy produced by the proposed policy integration framework satisfies the security

requirements of all collaborating domains. Chapter 6 gives a brief description of research problems that we intend to address during the course of this research.

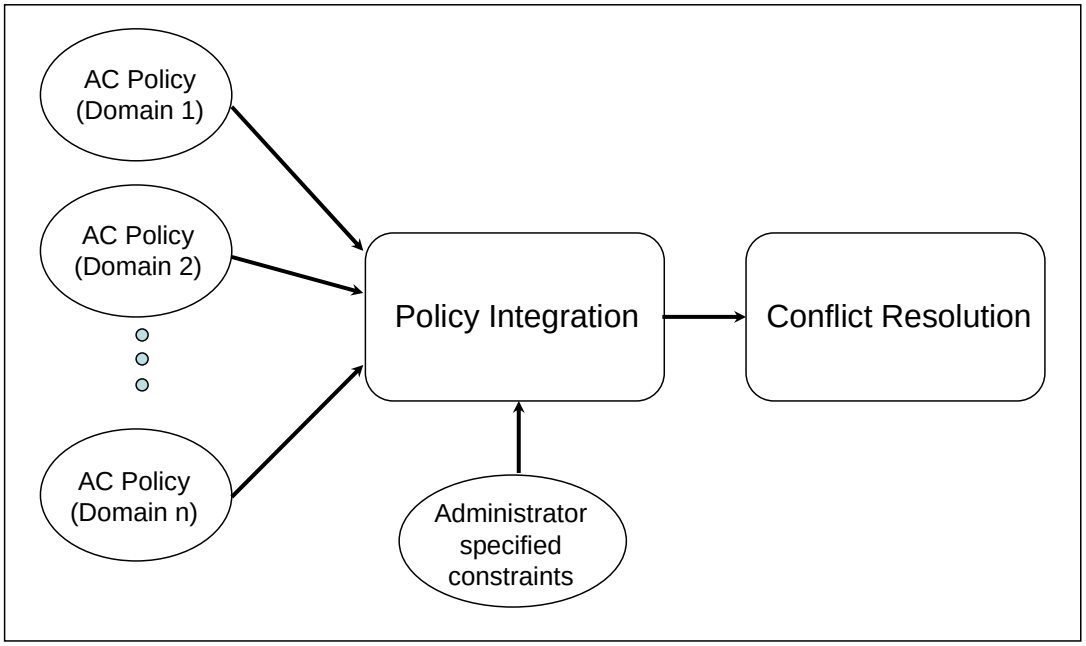

Fig. 1.1 Policy Integration Framework

2 OVERVIEW OF MULTI-DOMAIN RBAC SYSTEM

2.1 Role Based Access Control (RBAC)

Role based access control (RBAC) is a flexible approach that has generated great interest in the security community [Fer01, Giu95, Giu97, Jos01a, Jos01b, Ker02, Nya93, Nya99, Osb00a, San95, San96, San97, San98a, Tar97b]. In RBAC, users are assigned memberships to roles and these roles are in turn assigned permissions as shown in Fig. 2.1. A user can acquire all the permissions of a role of which he is a member. Role-based approach naturally fits into organizational contexts as users are assigned organizational roles that have well-defined responsibilities and qualifications [Fer93].

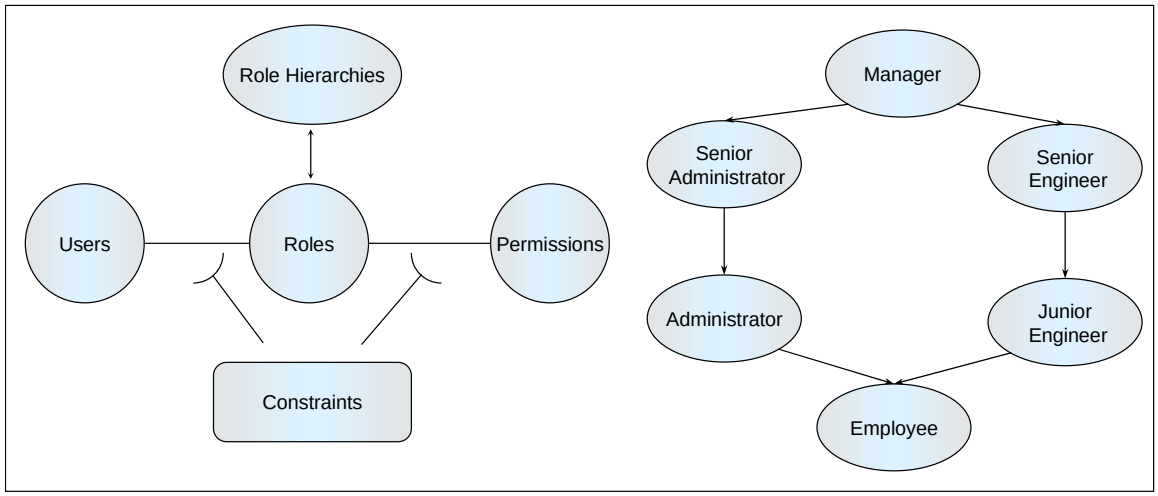

Fig. 2.1 Constraints and hierarchy in RBAC

According to a survey conducted by the U.S. National Institute of Standards and Technology (NIST) [Fer93], RBAC has been found to address many needs of the commercial and government sectors. This study showed that access control decisions in many organizations are based on "*the roles that individual users take on as part of the organization*." Many surveyed organizations indicated that they had unique security requirements and the available products did not have adequate flexibility to address them.

RBAC approach has several advantages, the key among which include [Jos01b, , San96]:

- *Security management*: The *role in the middle* approach to access control removes the direct association of the users from the objects. This logical independence greatly simplifies management of authorization in RBAC systems. For example, when a user changes his role, all that needs to be done is to remove his membership from the current role and assign him to the new role. In case authorizations were specified in terms of direct associations between the user and the individual objects, this change would require revoking permissions granted to all the objects and explicitly granting permissions to the new set of objects. Using a role-based approach, the number of assignments of users to permissions is considerably reduced. Generally, a system has a very large number of subjects and objects, and hence, using RBAC has benefits in terms of managing permissions.
- *Role hierarchy*: Natural role hierarchies exist in many organizations based on the principle of generalization and specialization [San98c]. For example, there may be a general *Employee* role in a Consulting Firm as shown in Fig. 2.1: *Employee, Engineer, Senior Engineer, Administrator, Senior Administrator* and *Manager*. Since everyone is an employee, the *Employee* role models the generic set of access rights available to all. A *Senior Engineer* role will have all the permissions that an *Engineer* role will have, who in turn will have the permissions available to the *Employee* role. Thus, permission inheritance relations can be organized in role hierarchies. This further simplifies management of access permissions. Fig. 2.1 shows a simple hierarchy.
- *Principle of Least Privilege*: RBAC can be configured to assign the least set of privileges from a set of roles assigned to a user when that user signs on. Using least privilege set minimizes the damage incurred to a system if someone not assigned to a role acquires its permissions through other means, or if someone masquerades as another user [Jos01b, sSan96].
- *Separation of Duties*: Separation of duties (SoD) has been considered a very desirable organizational security requirement [Ahn00, Ber99b, Bew89, Kun99, Nya99, San91, Sim97, Tid98]. SoD constraints are enforced mainly to avoid possible fraud in organizations. RBAC can be used to enforce such requirements easily – both statically and dynamically. For example, a user can be prevented from being assigned to two roles to prevent possible fraud by using a *static* SoD which says that a user cannot be assigned to two roles, one of which prepares a check and the other authorizes it.

- *Grouping Objects*: Roles classify users according to the activity or the access needs based on the organizational functions they carry out. Similar classifications can also be possible for objects. For example, a *secretary* generally has access to all the memos and letters in his/her office, whereas an accountant has access to all the bank accounts belonging to his/her organization. Thus when permissions are assigned to roles, it can be based on object classes instead of individual objects [San96]. This further increases the manageability of authorizations.
- *Policy-neutrality*: Role-based approach is policy-neutral and is a means for articulating policy [Jos01b, San96]. Role-based systems can be configured to represent many useful DAC, MAC policies [Nay95, Osb97, Osb00b] and user-defined and organizational security policies.

2.2 The NIST RBAC Model

The NIST 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*. A user is a human being or a process within a system. A role is a collection of permissions associated with a certain job function within an organization. Permission defines the access rights that can be exercised on a particular object in the system. A session relates a user to possibly many roles. When a user logs in the system he establishes a session by activating a set of enabled role that he is entitled to activate at that time. If the activation request is satisfied, the user issuing the request obtains all the permissions associated with the role he has requested to activate. One of the most important aspects of RBAC is the use of role hierarchies to simplify management of authorizations. The original RBAC model supports only *inheritance* or *usage* hierarchy, which allows the users of a senior role to inherit all permissions of junior roles. In order to preserve the *principle of least privilege*, RBAC model has been extended to include *activation hierarchy* which enables a user to activate one or more junior roles without activating senior roles [San98c]. A third type of hierarchy *inheritance-activation hierarchy* can be defined on roles by composing *inheritance* and *activation* hierarchies [Jos02]. From this point onward, we will use the notations *I*, *A*, and *IA* to refer to inheritance, activation and inheritance-activation hierarchies respectively. The symbols $\geq_I^*, \geq_A^*$, and $\geq_{IA}^*$ are used to express *I*, *A*, and *IA*

relationship between two roles respectively. Accordingly, $r_i \overset{*}{\underset{f}{\geq}} r_j$, where $f \in \{I, A, IA\}$, implies that role r_i is senior to r_j and the hierarchical relationship between them can be either *inheritance* only, or *activation* only *or inheritance-activation*. If role r_i is immediately senior to role r_j then the superscript * is omitted from the relation symbol $\underset{f}{\geq}$.

2.3 Graph-based Specification Model for RBAC

A graph based formalism can be used to specify the RBAC policy of a domain. In the graph based model, users, roles, and permissions are represented as nodes and the edges of the graph describe the association between various nodes. In order to capture the RBAC semantics, the nodes cannot be connected in an arbitrary manner. The type graph shown in Figure 2.2, defines all possible edges that may exist between different nodes. An edge between a user node *u* and a role node *r* indicates that role *r* is assigned to user *u*. Self edges on the role node *r* models the role hierarchy. In the type graph, *I-hierarchy*, *A-hierarchy* and *IA-hierarchy* are represented by *solid*, *dashed* and *bold-edges* respectively. There can be edges between role and permission nodes. A permission is a pair (*object*, *access mode*), which describes what objects can be accessed and in which mode (read, write, execute, approve etc).The graph model also supports specification of separation of duty (SoD) constraints. A role specific SoD constraint disallows assignment and/or activation of conflicting roles to same user. Similarly, a user specific SoD constraint prohibits conflicting users from assuming the same role simultaneously. In the graph model, a role-specific SoD constraint between two roles is represented by a double arrow between the corresponding roles. To represent conflicting users u_i and u_j for a role r_k, a double headed edge with a label r_k is drawn between the user nodes u_i and u_j. The label r_k specifies that the corresponding users are conflicting for role r_k and cannot acquire permissions over r_k simultaneously (user specific SoD constraint).

Figure 2.3 shows the graphical representation of an RBAC policy instance. The RBAC graph in Figure 2.3 consists of four roles r_a, r_b, r_c and r_d, with $r_a \underset{A}{\geq} r_c$, $r_a \underset{I}{\geq} r_d$, and $r_d \underset{A}{\geq} r_b$. User u_a is assigned to r_a, u_b assigned to r_b, and u_c assigned to r_c. Note that user u_a although inherits the permissions of role r_d, is not authorized to

activate role r_b which is junior to r_b in the activation hierarchy semantics. There exists a role specific separation of duty (SoD) constraint between role r_b and r_c, shown as a double headed arrow between these two roles in Figure 2.3. Also users u_a and u_c are conflicting users for role r_c and are not allowed to access r_c simultaneously.

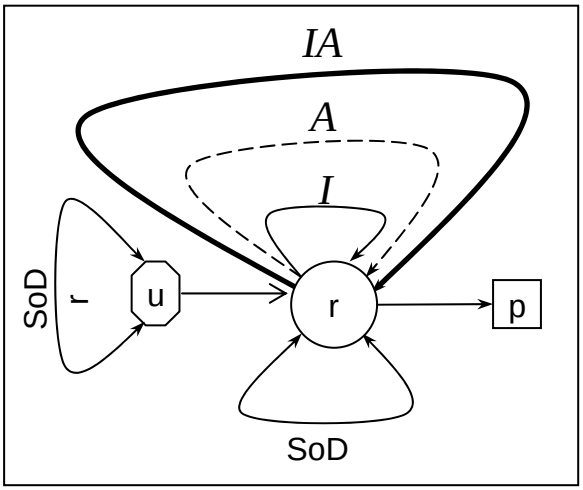

Fig. 2.2 RBAC type graph

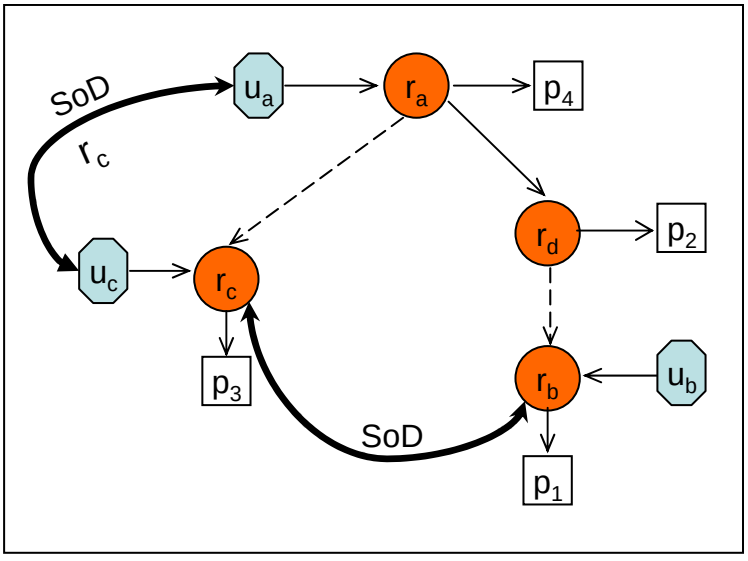

Fig. 2.3 An example of RBAC graph

2.4 Security Requirements in a Multi-domain RBAC System

In a multi-domain environment where distributed and heterogeneous organizations, each employing its own security policy, interoperate with each other, maintaining the security and privacy of data is highly problematic [Jos01a, Jos01b, Gon96]. One key aspect of this complex problem is the integration of diverse security policies and mechanisms of partner organizations into a coherent capability for managing access and use of local and cross-domain resources.

The goal of policy integration is to allow information and resource sharing without violating the security and autonomy of individual domains or of the multi-domain system as a whole. The security and autonomy requirements of the individual domains can be extracted from their respective access control policies. Additional security constraints can be defined by an administrator with global security responsibility. The administrator in charge of global security policy may specify both *permitted* and *restricted* inter-domain accesses. The global security policy constructed from the domains' policies and administrator specified access constraints may be inconsistent and may violate the security requirements of constituent domains as well as of the multi-domain system.

We mainly focus on three types of security policy violations. Although these security violations are independent of the underlying access control model, we describe them using the RBAC formalism to be consistent with our earlier discussion. The security policy violations include: i) violation of role assignment ii) violation of role-specific SoD constraint, and iii) violation of user-specific SoD constraint. A *role assignment violation* of domain k occurs when a user u of domain k acquires permission over role r of domain k in the multi-domain environment even though the user u is not directly assigned to role r or any of the senior roles of r that belong to domain k. A multi-domain policy violates the *role-specific SoD* constraint of domain k if the policy allows any user to simultaneously access two conflicting roles of domain k. Similarly, a *user-specific SoD* constraint violation occurs when a multi-domain policy permits conflicting users of role r to acquire permissions over r in concurrent sessions. The following examples illustrate these three types of security violations.

Example 1

Figure 2.4 shows a multi-domain policy that allows collaboration between *County Treasurer Office* (CTO) and *County Clerk Office* (CCO). The *County Treasure Office* has following roles: Tax Collection Manager (TCM), Tax Assessment Clerk (TAC), Tax Billing Clerk (TBC), Tax Collection Clerk (TCC), and Junior Tax Collection Clerk (JTCC). TCM inherits all permissions of TCC which further inherits the permissions of JTCC. The roles TAC and TBC are junior to TCM in the activation hierarchy semantics, implying that a user assigned to TCM can assume the roles TAC and TBC without actually activating TCM. However, an SoD constraint is defined between TAC and TBC meaning that these roles cannot be assumed by same user simultaneously. There is a user-specific SoD constraint between user u_1 assigned to TCM, and u_2 assigned to TAC. This SoD constraint prohibits u_1 and u_2 to assume the role TAC concurrently. The *County Clerk Office* has only two roles, namely: Property Tax Manager (PTM) and Property Tax Clerk (PTC) with PTM inheriting the permissions of PTC.

The multi-domain policy shown in Figure 2.4 defines the following interoperation between CTO and CCO:

1. TCM in the *County Treasure Office* inherits all the permissions available to PTM in the *County Clerk Office*.
2. JTCC in the *County Treasure Office* inherits all the permissions available to PTC in the *County Clerk Office*.
3. PTM in the *County Clerk Office* inherits all the permissions of TAC in the *County Treasurer Office*.
4. PTC in the *County Clerk Office* inherits all the permissions of TCC in the *County Treasurer Office*.

The above multi-domain policy leads to all three types of security violations. It allows JTCC to access the permissions of its senior role TCC through PTC, which is a violation of *role assignment constraint*. Moreover, this policy permits u_1 to activate roles TCM and TBC simultaneously. This leads to a violation of *role-specific SoD*, as by activating the role TCM, u_1 acquires the permissions of the role TAC through PTM. Moreover, the multi-domain policy allows u_1 to activate the role TCM and u_2 to assume

the role TAC. u_1 by activating TCM can acquire permission over TAC through the role PTM. This is a violation of *user-specific SoD constraint* which prohibits u_1 and u_2 from accessing the role TAC simultaneously.

Example 1 considers security constraints that are specific to a particular domain. The security constraints can also be defined between cross-domain entities (roles and users). Following example presents a case where cross-domain security constraints are needed.

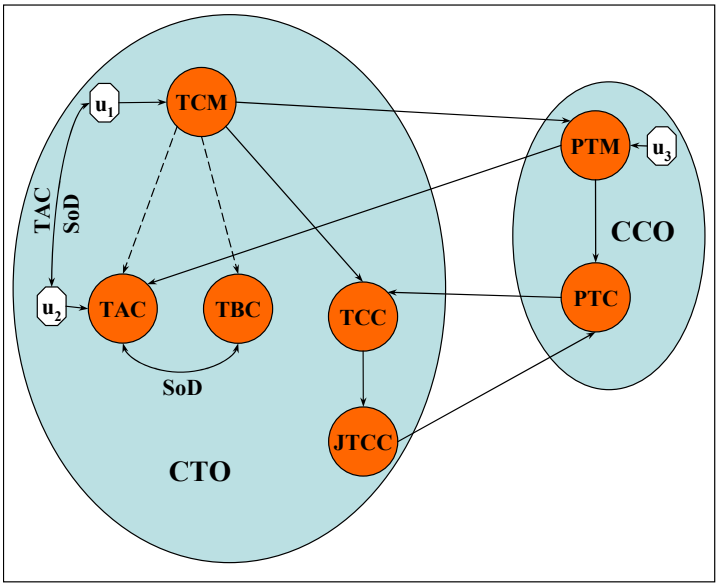

Fig. 2.4 A multi-domain access control policy defining interoperation between CTO and CCO

Example 2

Consider Corporate Audit Department that performs tax auditing of public companies for IRS. For each such company there is a separate auditor role which is authorized to check the books and audit records maintained by the company. IRS may also hire private auditing firms to perform tax auditing. Companies are also required to document their financial information every year and they may also contract private audit firms to perform their internal auditing. The internal auditor is allowed to access all the financial records and books of the company being audited. However, the internal auditor cannot acquire any permission that is exclusively assigned to the IRS auditor. If the

interoperation policy is not carefully designed then there may arise a situation in which same audit firm performs IRS auditing and internal auditing of the same company. To avoid this security flaw, an SoD constraint needs to be defined between the IRS auditor role and the internal auditor role. Note that this SoD is defined between two cross-domain roles. This is illustrated in Figure 2.5.

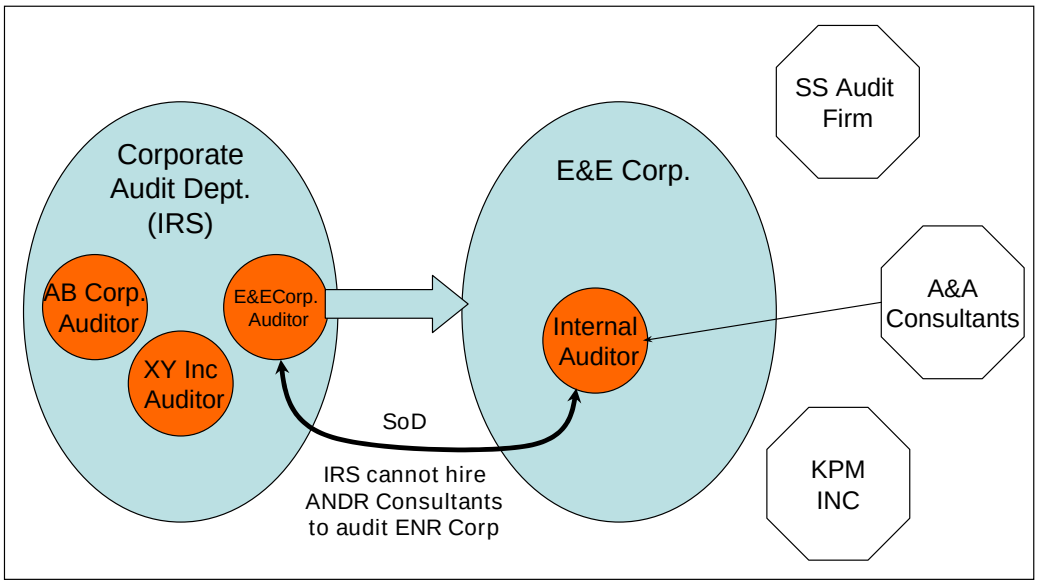

Fig. 2.5 Example of a cross-domain separation of duty (SoD) constraint

3 MULTI-DOMAIN POLICY INTEGRATION

In this chapter, we elaborate on the policy integration phase of Figure 1.1. Before describing the proposed policy integration mechanism, we first introduce the resource sharing policy at the object level and then highlight some of the heterogeneity issues involved in policy integration.

3.1 Information Sharing Policy

In the policy integration step of Figure 1.1, domain policies are composed to form a global interoperation policy. Note that a domain may not allow complete sharing of its data and resource objects. We will use the word object interchangeably for both data and resources. An object can be a file, a database relation/view, or an I/O device etc. For each of the sharable objects the following information needs to be provided by the controller/owner domain of that object.

- Domains which can access the object.
- Sanitization requirements of an object before it is shared with other domains. For instance, an object can be completely shared, or partially shared or the object cannot be shared as is but only certain derived properties of the object are shareable (statistical information).
- Access permissions (read, write, execute etc.) over an object that are available to subjects of foreign domains.
- Any specific condition for sharing. For instance, an object can be shared (completely or partially) with a cross domain subject only if a cross domain subject has local access to certain attributes of the object in its own domain.

Based on the above information, each object can be logically partitioned into multiple objects and only shareable sub-objects of a domain are presented to the policy

integration module. Figure 3.1 describes an abstract view of inter-domain information sharing. This figure depicts partial sharing, which is the most common form of interoperation and is exhibited in almost every collaborative environment. Note that in this figure, access to local information resources is also reduced as a result of cross-domain resource sharing. This reduction in local accesses results in decreasing the autonomy of corresponding domains.

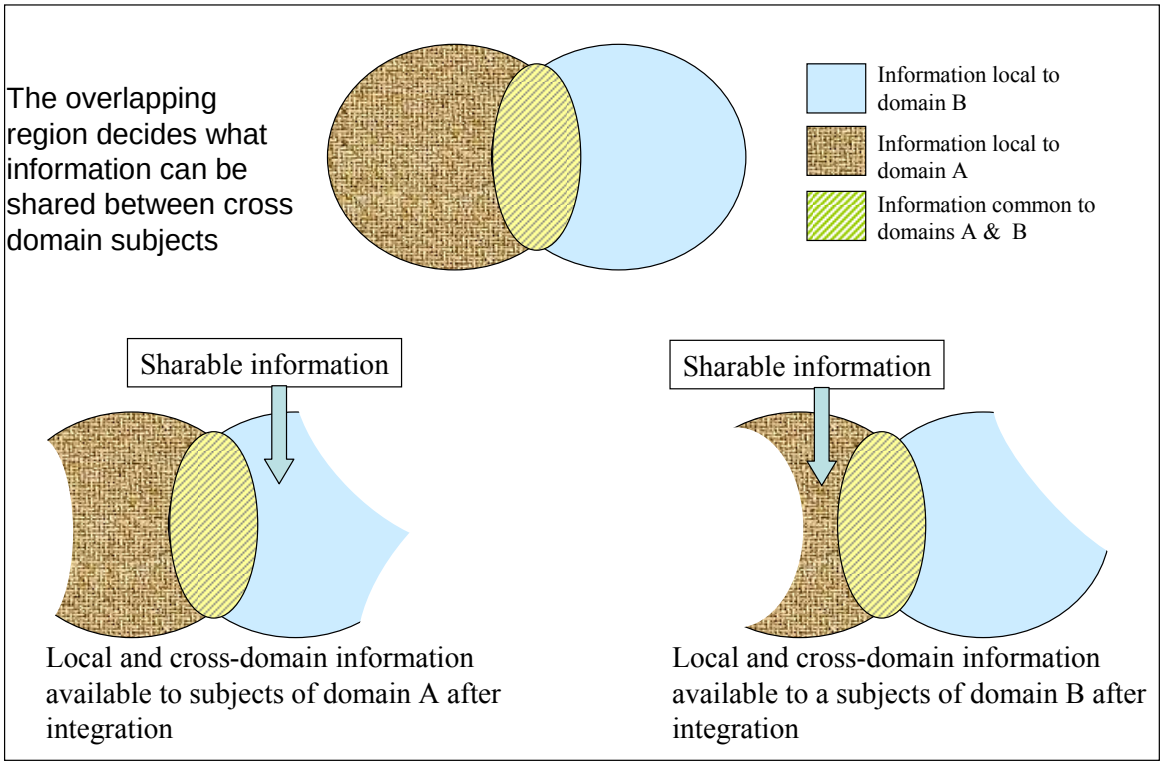

Fig. 3.1 An abstract view of inter-domain information sharing

Figure 3.2 depicts information sharing policy related to *delinquent property tax* between *County Treasurer Office* (CTO) and *District Clerk Office* (DCO). CTO maintains electronic records of tax defaulters containing information such as tax defaulters name and social security number (SSN), delinquent property index and tax amount owed to local govt. redemption cost, tax sale plea filed in district court, and details of other property/properties owned by the tax defaulter. Delinquent taxes can be sold to third parties after obtaining the tax sale order issued by the district court. The District Clerk office (DCO), which keeps record of all court proceedings, is responsible for providing the tax sale orders and other court documents related to delinquent tax

holder to CTO and other concerned agencies/departments. Similarly, DCO is allowed to access the information of delinquent property, maintained by CTO, for record keeping. In order to keep privacy of personal/unrelated information, not all the information about the tax defaulter needs to be shared between the two domains. For instance, the information about other real-estate property owned by the tax defaulter is kept private and is not shared with DCO unless such property is declared delinquent. Similarly, CTO is not allowed to access any information from DCO other than tax indictment record, tax sale order, and local tax lien records. For this purpose, the tax defaulter record in the CTO is partitioned into three objects: O_{com}, O_{sT}, and O_{rT}. O_{rT} is classified information that cannot be shared with the DCO. O_{sT} is a shareable object and can be accessed by DCO. Similarly, the record in the DCO is partitioned into O_{com}, O_{sC} and O_{rC}, where O_{rC} is confidential information, and O_{sC} can be released to CTO. The object O_{com} contains the information about the name and social security number of the defaulted person and is common to both domains. CTO can access only those records from DCO domain for which there is a corresponding O_{com} object in the delinquent tax table. Similarly, DCO can access tax/property information of only those tax holders for which the O_{com} from court records matches with the O_{com} of the delinquent tax record.

3.2 Heterogeneity Issues in Policy Integration

One key challenge in the composition of a multi-domain access control policy is resolving semantic heterogeneity among the local policies of collaborating domains. There are various types of heterogeneity that need to be addressed in the context of policy integration. The heterogeneity may arise because of naming conflicts, schema mismatch, and differences in constraint representation by different domains.

Naming Conflicts arise because of the use of synonyms, or identical names, to represent different conceptual entities, and homonyms, or different names, to represent same conceptual entities. Accordingly, there may be naming conflicts among different inter-domain entities, which may cause security violations if not resolved before establishing interoperation. Resolution of naming conflicts has been addressed in the literature in the context of schema integration in the database area [Gua02, Vet98]. These

techniques require the use of a global lexicon to extract the conceptual meaning of attributes from their names. Additionally, domain-based and value-set-based comparisons can be performed for refinement [Li94].

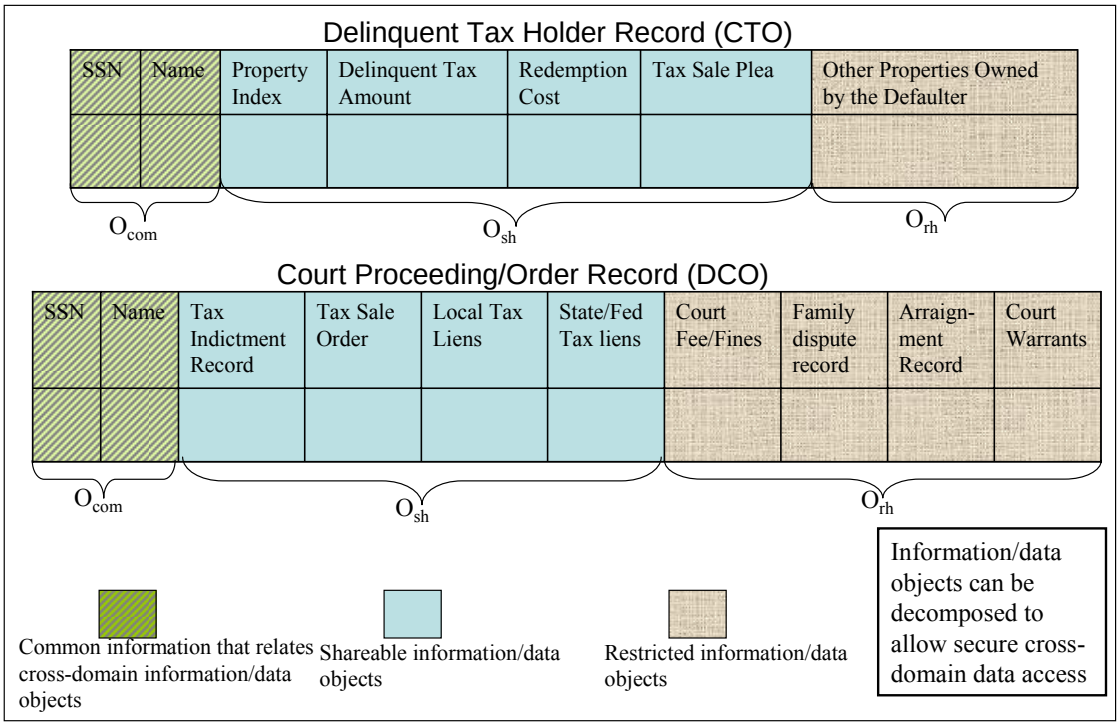

Fig. 3.2. Information exchange between the County Treasurer Office and District Clerk Office

Schema mismatch is another type of semantic heterogeneity that is characterized by representation conflicts, meta-model conflicts and meta-meta-model conflicts [Pot03]. The term model is used to formally describe a complex application, such as a database schema, an application interface, or an access control policy. Representation conflicts are caused by conflicting representations of same real-world concept. For instance, in one domain the attribute *Name* is represented by the element "Person Name," while in another domain, it is represented by two elements: "First Name" and "Last Name". Meta-model conflicts occur due to the use of different models for the same schema. For example, one domain uses the relational model and the other uses the object oriented model to specify the same schema. Conflicts also exist at the meta-meta-model level due to the use of different relationship orderings and cross-relation implication among the

domain's entities. Schema and model merging techniques [Bat86, She90, Pot03] address the issue of reconciliation of semantic differences at the schema level.

In addition to naming and schema conflicts, heterogeneity may appear in the specification of various access control policy constraints, including: hierarchy, SoD, cardinality and other dynamic constraints. Reconciliation of semantic differences becomes more challenging in presence of constraint heterogeneity.

Hierarchical heterogeneity among domains' policies may exist because of two reasons: a) use of different role hierarchies (inheritance *I*, activation A, inheritance-activation *IA*, hybrid [Jos02]) by different collaborating domains; b) domains may use different hierarchical ordering to represent same authorizations for a given role. The following example illustrates the two types of hierarchical heterogeneity that may exist between two or more cross-domain roles.

Example 3

Consider the Senior Clerk (SC) and Junior Clerk (JC) roles of the City Clerk Office shown in Figure 3.3(a). The hierarchical relationship between SC and JC is given by A-hierarchy, $SC \underset{A}{\geq} JC$, i.e., SC cannot directly inherit the permissions associated with the role JC. Suppose permission p_1 is assigned to role SC and p_2 to JC. Figure 3.3(b) shows the RBAC graph of *County Clerk Office* with two roles Clerk (C) and Assistant Clerk (AC). The Clerk role (C) inherits all the permissions of Assistant Clerk, $C \underset{I}{\geq} AC$. Note that the roles C and AC are assigned same permissions as the roles SC and JC. However, roles SC and C are not equivalent because SC is not authorized for permission p_2, whereas, C can directly access p_2 without activating any junior roles. The difference in authorization of the two roles is because of different types of hierarchy used in the two domains.

It can also be noted in Figure 3.3 that the Accountant role in the *City Clerk Office* has the same permission authorization as the Clerk role in the *County Clerk Office*, even though the hierarchical ordering for the two roles is different.

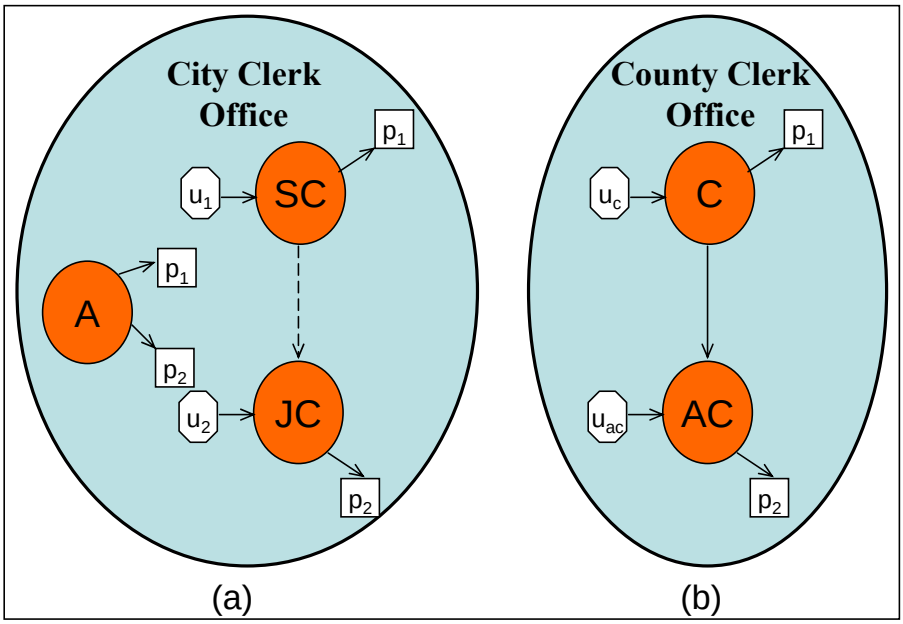

Fig. 3.3 Hierarchical heterogeneity

3.3 RBAC Policy Integration

In this section, we focus on the issue of composing a global access control policy from the access control policies of collaborating domains. The global policy governs both intra-domain and inter-domain information and resource exchange. As mentioned earlier, the access control policies of collaborating domains are specified using RBAC framework. The domains' policies are combined based on the similarity between the permissions associated with the cross-domain roles. Before presenting the proposed policy integration mechanism, we first introduce the general requirements for policy integration.

3.3.1 Policy Integration Requirements (PIR)

The following PIRs define the semantics of a multi-domain RBAC policy in a concrete manner. The RBAC policies are specified using the graph formalism described in Chapter 2.

1. *Element preservation*: Each element (role, user, permission) in the input RBAC graph has a corresponding element in the multi-domain graph G.

2. *Relationship preservation*: Each relationship in the input graph is explicitly in or implied by the multi-domain graph G.
3. *User authorization preservation*: In the multi-domain graph G, for any user *u* of a domain *k*, the permission authorization set of *u* over the objects of domain *k* should not be different from the permission authorization set specified or implied in the input RBAC policy of domain *k*.
4. *Minimum overhead*: In order to satisfy the constraints given above, the multi-domain RBAC graph may include elements and relationships in addition to those given in the input RBAC graphs. However, the number of additional elements should be minimum.
5. *Order independence*: The order in which policies are integrated should not influence the output of policy integration operation.
6. *Constraint satisfaction*: The multi-domain RBAC graph G must satisfy all the constraints of the input RBAC policies.

The above PIRs are similar to the generalized merge requirements defined in the context of model merging by Pottinger and Bernstein [Pot03]. Unlike [Pot03], we do not define the conflict resolution strategy as a part of integration requirement. In general, fundamental conflicts in schema/model merging arise because of type restriction, cyclicity in relationships, and relation cardinality. These conflicts can be resolved using priority-based pre-specified resolution rules. Use of static conflict resolution rules in policy integration may severely reduce the amount of interoperation among the collaborating domains. Moreover, the relationship semantics and cross-relation implications in RBAC framework are different from schema models discussed in [Pot03]. Therefore, a pre-specified conflict resolution strategy that resolves model merging conflicts on the fly cannot be applied for policy integration. We use a separate conflict resolution module that resolves conflicts in the multi-domain policy obtained in the policy integration phase of Figure 1.1.

In the following section, we describe an algorithm, *RBAC-integrate*, for integrating policies of multiple domains. *RBAC-integrate* combines the RBAC policies of

component domains by comparing cross-domain roles. However, *RBAC-integrate* does not resolve any conflicts occurring in the resulting multi-domain policy. For conflict resolution, we propose a conflict resolution mechanism discussed in Chapter 4. It can be proved that the multi-domain policy produced by *RBAC-integrate*, after conflict resolution satisfies all the policy integration requirements given above.

Table 3.1 Functions/predicates used in this report

Function/predicate	**Description**
Pset(*r*)	Returns the set of all permissions either directly assigned to role *r* or are inherited by *r*.
$Pset_{assign}(r)$	Returns the set of permissions directly assigned to role *r*.
Class(*O*)	Returns the conceptual class of object *O*.
Conf-rset(*r*)	Returns the set of all roles conflicting with role *r* i.e., roles that cannot be acquired along with role *r* by any user.
Conf-user(*r*)	Returns the set of the sets of user that cannot acquire role *r* simultaneously.
Shareable(*O ,a, X*)	Returns True if permission (O, a) can be shared with domain X
Seniormost-role(G)	Returns the senior-most role of the RBAC graph G
Children(r)	Returns all roles *r'* such that $r \underset{I}{\geq} r' \ \vee \ r \underset{A}{\geq} r'$
Common-permissions(r_1,r_2)	Returns the set of all directly assigned permissions that are common to the cross-domain roles r_1 and r_2.
Common-juniors-I(r_1,r_2)	Returns the set of roles R_j $R_j = \left\{ r : r_1 \underset{I}{\geq} r \text{ and } \exists r' \ \left(eq\_role(r, r') \wedge r_2 \underset{I}{\geq} r' \right) \right\}$, r_1 and r_2 are cross-domain roles.
New-role(r)	Returns True if *r* is a newly created role as a result of role splitting.
Redundant(r)	Returns True if *r* is a redundant role.
Not-compared-previously(r_1,r_2)	Returns True if the cross-domain roles r_1 and r_2 are not compared by the algorithm Role-integrate
Already-linked(r_1,r_2)	Returns True if r1 and r2 are cross-domain roles and $r_1 \underset{I}{\geq} r_2$ and $r_2 \underset{I}{\geq} r_1$
Eq_role(r_1,r_2)	Returns True if the following hold $pset_{assign}(r_1) = pset_{assign}(r_2) \wedge$ $\left[\text{for all } r_{1j} \text{ such that } r_1 \underset{I}{\geq} r_{1j} \text{ there exists } r_{2j} \text{ for which } r_2 \underset{I}{\geq} r_{2j} \text{ and } eq\_role(r_{1j}, r_{2j}) \right] \wedge$ $\left[\text{for all } r_{1j} \text{ such that } r_1 \underset{A}{\geq} r_{1j} \text{ there exists } r_{2j} \text{ for which } r_2 \underset{A}{\geq} r_{2j} \text{ and } eq\_role(r_{1j}, r_{2j}) \right]$ *i.e.*, the roles r_1 and r_2 set of directly assigned permissions and are also equivalent in their hierarchical structure.
contained(r_1,r_2)	Returns True if the following hold $\left(p \in Pset_{assign}(r_1) \Rightarrow p \in Pset_{assign}(r_2) \right) \wedge \left((r_1 \underset{I}{\geq} r_k \wedge r_k \neq r_2) \Rightarrow r_2 \underset{I}{\geq}^{*} r_k \right)$ *i.e.*, the set of directly assigned permissions of r_1 must be contained in the set of directly assigned permissions of r_2 and all the roles junior to role r_1 must also be junior to r_2 in the same hierarchy semantics.
Overlap(r_1,r_2)	Returns True if the following hold $\left(\exists p \mid p \in Pset_{assign}(r_1) \Rightarrow p \in Pset_{assign}(r_2) \right) \vee \left(\exists r_k, r_m \mid r_1 \underset{I}{\geq} r_k \Rightarrow (r_2 \underset{I}{\geq} r_m \wedge eq\_role(r_k, r_m) \right)$
u-assign(u,r)	Returns True if user *u* is assigned role *r*.
Conf-role(r_1,r_2)	Returns True if r_1 and r_2 are conflicting roles

3.3.2 RBAC Policy Integration Algorithm

The proposed policy integration algorithm establishes correspondences between cross-domain roles by considering the permissions associated with the corresponding roles. Inter-domain roles are compared based on their permission assignments over objects. This permission set includes both directly assigned permissions as well as inherited permissions. We also assume that objects in the RBAC model are organized into conceptual classes, e.g., account tables, insurance claims, and audit reports etc. Two cross-domain permissions p_A:(O_A, a_A) and p_B:(O_B, a_B) of domains A and B respectively, are termed equivalent if the cross domain objects O_A and O_B belong to the same conceptual class and the permissions p_A and p_B are declared shareable in their respective domain policies.

Using the above assumptions and the permission assignments of roles over the objects, four types of relations can be defined between two cross-domain roles r_A and r_B belonging to domain A and domain B respectively. The functions and predicates used in defining these relations are explained in Table 3.1.

1. *Equivalent*: r_A is equivalent to r_B ($r_A \approx r_B$), if the following conditions hold.
    a. The permission sets *Pset*(r_A) and *Pset*(r_B) of roles r_A and r_B are equivalent. Formally:

    $$\forall i, j : class(O_{A_i}) = class(O_{B_j}) \wedge [(O_{A_i}, a) \in Pset(r_A) \Leftrightarrow (O_{B_j}, a) \in Pset(r_B)]$$

    b. All the permissions in the sets *Pset*(r_A) and *Pset*(r_B) are shareable with the domain of r_A and r_B respectively. Formally:

    $$\forall i, j \; shareable(O_{A_i}, a, B) \wedge shareable(O_{B_j}, a, A)$$

2. *Contain*: r_A contains r_B ($r_A \supset r_B$) if the following hold:
    a. The permission set *Pset*(r_B) of role r_B is included in the permission set *Pset*(r_A) of role r_A.

    $$\forall j \exists i : (O_{B_j}, a) \in Pset(r_B) \Rightarrow \left[(O_{A_i}, a) \in Pset(r_A) \wedge \left(class(O_{A_i}) = class(O_{B_j}) \right) \right]$$

    b. All the permissions in the set *Pset*(r_B) are shareable with domain A.

RBAC-integrate(G_1,G_2,…,G_n)
1. G = {V[G_1], E[G_1]}
2. **for** i ← 2 to n
3. r_1 ← seniormost-role(G)
4. r_2 ← seniormost-role(G_i)
5. G ← Role-integrate(r_1, r_2)
6. **for** each r ∈ G
7. **if** (new-role(r) **and** redundant(r))
8. **then** Remove-Role(G, r)
9. **return**

Role-integrate(r_1, r_2)
1. **for** each r_c ∈ children(r_1)
2. **do if** ((Pset(r_c) ∩ Pset(r_2) ≠ ϕ) **and** not-compared-previously(r_c,r_2))
3. **then** Role-integrate(r_c,r_2)
4. **for** each r_c ∈ children(r_2)
5. **do if** ((Pset(r_1) ∩ Pset(r_c) ≠ ϕ) **and** not-compared-previously(r_1,r_c))
6. **then** Role-integrate(r_1,r_c)
7. ► return without doing anything if r_1 and r_2 are already linked
8. **if** already-linked(r_1,r_2)
9. **then return**
10. ► $contained(r_i,r_j)$=True, if $\left(p \in Pset_{assign}(r_i) \Rightarrow p \in Pset_{assign}(r_j)\right) \wedge \left((r_i \geq_I r_k \wedge r_k \neq r_j) \Rightarrow r_j \geq_I^* r_k\right)$
11. **if** contained(r_2, r_1) and contained(r_1, r_2)
12. **then if** linking r_1 and r_2 do not violate RBAC consistency properties
13. **then** link(r_1, r_2)
14. **return**
15. **else if** contained(r_2, r_1)
16. **then** r_{1j}=split(r_1, common-permissions(r_1,r_2), common-juniors-I(r_1,r_2))
17. **if** linking r_{1j} and r_2 do not violate RBAC consistency properties
18. **then** link(r_{1j}, r_2)
19. **return**
20. **else if** contained(r_1, r_2)
21. **then** r_{2j}=split(r_2, common-permissions(r_1,r_2), common-juniors-I(r_1,r_2))
22. **if** linking r_1 and r_{2j} do not violate RBAC consistency properties
23. **then** link(r_{2j},r_1)
24. **return**
25. ► $overlap(r_i,r_j)$=True, if $\left(\exists p \mid p \in Pset_{assign}(r_i) \Rightarrow p \in Pset_{assign}(r_j)\right) \vee \left(\exists r_k, r_m \mid r_i \geq_I r_k \Rightarrow (r_j \geq_I r_m \wedge already-linked(r_k, r_m)\right)$
26. **else if** overlap(r_1,r_2)
27. **then** r_{1j}=split(r_1, common-permissions(r_1,r_2), common-juniors-I(r_1,r_2))
28. r_{2j}=split(r_2, common-permissions(r_1,r_2), common-juniors-I(r_1,r_2))
29. **if** linking r_{1j} and r_{2j} do not violate RBAC consistency properties
30. **then** link(r_{1j}, r_{2j})
31. **return**
32. **return**

Fig. 3.4 Policy integration algorithm

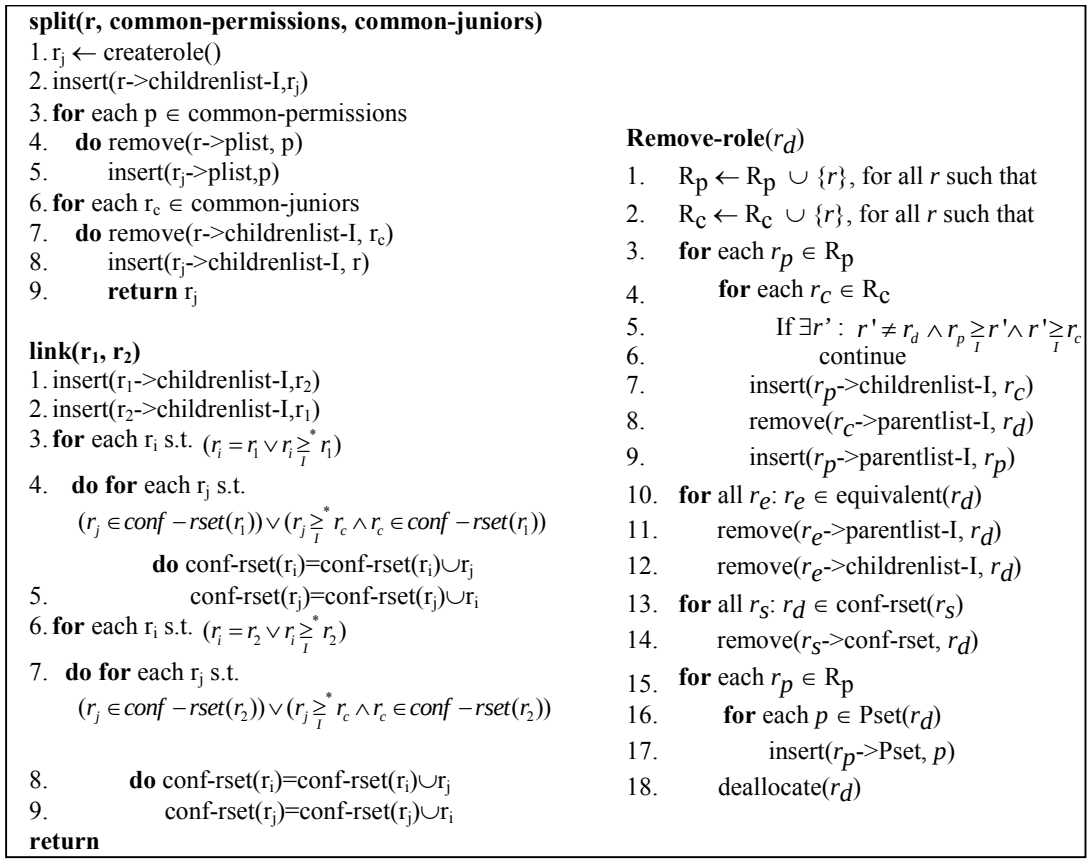

Fig. 3.5 Procedures used by Role-Integrate during Policy Integration

3. *Overlap*: r_A overlaps r_B (r_A *O* r_B) if *Pset*(r_A) and *Pset*(r_B) have some common shareable permissions and neither r_A *contains* r_B nor r_B contains r_A. Formally:

$$\left(\begin{array}{l}\exists i, j : class(O_{A_i}) = class(O_{B_j}) \wedge [(O_{A_i}, a) \in Pset(r_A) \wedge \\ \qquad (O_{B_j}, a) \in Pset(r_B) \wedge shareable(O_{A_i}, a, B) \wedge shareable(O_{B_j}, a, A)]\end{array}\right) \wedge$$
$$\left(\neg(r_A \ contain \ r_B) \wedge \neg(r_B \ contain \ r_A)\right)$$

4. *Not related*: r_A is not related to r_B ($r_A \neq r_B$) roles r_A and r_B do not share any common permissions. Formally:

$$\neg\exists i, j : class(O_{A_i}) = class(O_{B_j}) \wedge [(O_{A_i}, a) \in Pset(r_A) \wedge (O_{B_j}, a) \in Pset(r_B)]$$

Figure 3.4 shows the proposed policy integration algorithm, *RBAC-integrate*, that integrates RBAC policies of *n* domains to produce a global multi-domain policy. The input parameter G_i represents the RBAC policy of domain *i* specified in graphical form. This algorithm iteratively combines the RBAC policies of component domains in a pair-

wise manner. In the first iteration, an integrated RBAC policy is composed from domains 1 and 2 by calling the procedure, *role-integrate*, with the senior-most roles of domains 1 and 2 respectively. In the subsequent iterations, RBAC policy of a new domain is combined with the integrated RBAC policy obtained in previous iteration. After *n*-1 iterations, the RBAC policies of all *n* domains are integrated to produce a global multi-domain policy. In each iteration, after calling *role-integrate*, all the newly created *redundant roles* are removed from the integrated RBAC graph. Redundant roles, formally defined in Section 3.3.3, are roles that do not have any permissions assigned to them nor can any user activate them. Removal of redundant roles, created in the process of integration, is essential to ensure that RBAC-integrate preserve PIR 4 and 5 listed above.

The procedure, *role-integrate,* integrates inter-domain roles based on their permission assignment and hierarchical ordering. *role-integrate* is a recursive algorithm that uses bottom-up strategy to establish role equivalence across two domains. The algorithm basically checks all inter-domain roles for one of the above four relations. If the roles do not share any permission, then it returns without doing anything. If the inter-domain roles say, r_1 and r_2, are equivalent in their permission assignment and hierarchical ordering then they are linked together. An inter-domain link in the graph model is represented by a dashed double-headed arrow between two roles. Linking two inter-domain roles r_1 and r_2 implies that a user say u_i, authorized for role r_1 inherits all the permissions of role r_2. Similarly, a user u_j authorized for role r_2 inherits all permissions in the authorization set of r_1. *Role-integrate* calls *link* function (shown in Figure 3.5) for linking cross-domain roles r_1 and r_2. *link* makes r_1 and r_2 junior to each other in the *I*-hierarchy sense. In addition, conflicting role sets of r_1 and r_2 and all their senior roles that have an I-path to r_1 and r_2, and all the roles that conflict with r_1 and r_2 and their senior roles are updated. This update in the conflicting role sets is essential to preserve the hierarchical consistency property of RBAC model which requires that the conflicting role set of a junior role must be contained in the conflicting role set of the senior role [Gav98]. As a result of this update in conflicting role sets, new *SoD* constraints are added between two or more roles which do not conflict with each other in their original domain RBAC policy. We will use the term *induced SoD constraint* to denote such *SoD* constraints that

are not present in the domains' original RBAC policies. A formal definition of induced SoD constraint is given in Section 4.2 of Chapter 4.

In presence of multiple hierarchy types, addition of roles in the conflicting role sets may lead to a situation in which two conflicting roles, say r_1 and r_2, have a common ancestor, say r_a, which inherits both roles r_1 and r_2, (*i.e.*, $r_a \geq_I^* r_1$, $r_a \geq_I^* r_2$). This situation can be avoided by making r_1 and r_2 conflicting roles only if they do not have a common ancestor role that inherits them. This is illustrated in Figure 3.6 which shows how linking inter-domain roles change the conflicting set of linked roles. Figure 3.6(a) shows roles r_1, r_2, r_3, r_4 and r_5, with r_1, r_2 and r_3 belonging to domain A, and r_3 and r_4 belonging to domain B. The role r_1 inherits all the permission of r_2 and r_3. As shown in Figure 3.6(a) roles r_4 and r_5 are conflicting roles. Roles r_2 and r_4, and r_3 and r_4 are equivalent in terms of their permission assignment and can be linked. Figure 3.6(b) shows the integration of RBAC graph of Figure 3.6(a). Note that after linking, no role specific SoD constraint is defined between r_2 and r_3 because they both have a common ancestor r_1 in the inheritance hierarchy semantics. In contrast, a SoD constraint is defined between r_2 and r_3 in Figure 3.6(d) which have a common ancestor role r_1 in the activation hierarchy semantics. The integrated policy shown in Figure 3.6(b) is conflicting and can be made consistent by removing one of the links $r_2 - r_4$ or $r_3 - r_5$.

Two cross-domain roles may also have a subset-superset (containment) or overlapping relationship. Role r_1 is contained in r_2 if the set of all permissions directly assigned to r_1 is contained in the set of permissions directly assigned to r_2, and all the roles that are junior to r_1 in the *I*-hierarchy semantics are also junior to r_2 in the *I*-hierarchy semantics. Note that containment relation mentioned here is slightly different from the containment relation defined earlier. In this case, hierarchical ordering is also considered in addition to permission assignment in defining the containment relationship between two roles. If r_2 contains r_1, then a junior role r_{2j} is created by calling *split* function shown in Figure 3.5. In the *split* function, all the permissions and junior roles (*I*-hierarchy semantics) common to both r_1 and r_2 are removed from r_2 and are assigned to r_{2j}. Splitting a role does not change the permission authorization set of user and is

formally proved in lemma 3.1. After permission reassignment r_{2j} and r_1 are linked together. If r_1 and r_2 overlap but none of the roles contain each other, then two new roles r_{1j} and r_{2j} are created and made junior to r_1 and r_2 respectively. Permissions and junior roles common to both r_1 and r_2 are removed from the senior roles r_1 and r_2 and assigned to the roles r_{1j} and r_{2j}. After this permission and role assignments, r_{1j} and r_{2j} are linked.

In Chapter 4, we provide an example of the proposed policy integration mechanism for different offices of a county collaborating with each other for collection and sale of real-estate taxes on property parcels located within the jurisdiction of the concerned county. The county offices involved in this collaboration are *County Clerk Office* (CCO), *County Treasurer Office* (CTO), *County Attorney Office* (CAO), and *District Clerk office* (DCO). Figure 4.5 shows the graphical representation of RBAC policies of CCO, CTO, and CAO and Figure 4.6 depicts the RBAC graph after applying the role-integrate algorithm over the role graphs of Figure 4.5. The dotted double-headed arrow in Figure 4.6 between two cross-domain roles defines the access path between the respective roles. If two cross-domain roles are linked together by a cross-domain link then a user who is authorized for one of the roles can also inherit the permissions of the other and vice versa. For instance, user u_5 assigned to the DTC role in CTO is also authorized to access the roles $DTLO_{01}$ and $DTLO_{00}$ in the CCO because of the presence of an inter-domain link between the roles R_5 in CTO and $DTLO_{01}$ in CCO. Similarly, user u_7 assigned to TAC role in CCO is authorized to access the roles $R2_{07}$ and $R2_{05}$ in the CTO by virtue of the inter-domain link between $R9_{08}$ and $R2_{05}$.

Note that some of the roles in Figure 4.6 are split into two or more roles with their permissions redistributed among the newly created junior roles. For instance, the DTM role in Figure 4.5 gets split into three roles DTM, DTM_{10} and DTM_{12} with DTM as the senior of the remaining two (shown in Figure 4.6). The following lemma maintains that role splitting does not change the authorization set of users provided that no user is assigned to the newly created junior roles. Before stating the lemma, we would like to informally introduce the notion of a *uniquely activable set* (UAS) of a role. Interested readers are referred to [Jos03] for a formal definition of UAS of a role. Uniquely activable set (UAS) of a role *r* is the set of *role sets* that can be concurrently activated by

a user assigned to role *r*. In other words, UAS gives the role combinations that can be activated by a user concurrently.

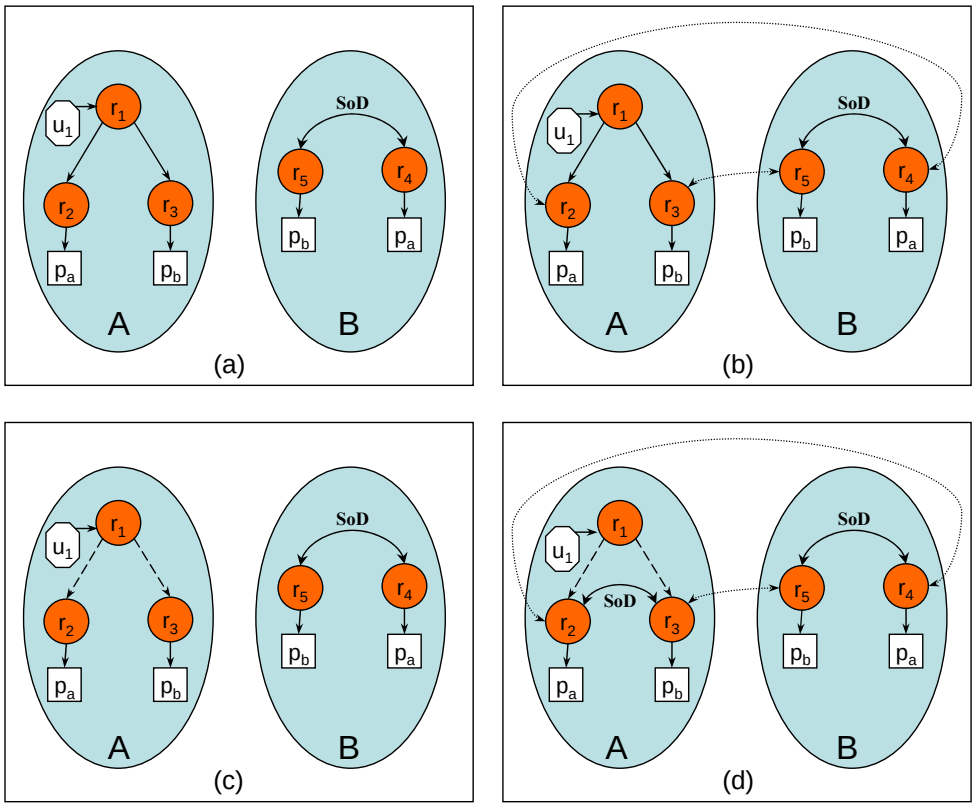

Fig. 3.6 Example of induced SoD

Lemma 3.1: Let a role *r* is split into roles r_s and r_j with $r_s \geq_I r_j$. Then *r* and r_s verify the following conditions:

$pset(r) = pset(r_s)$

$UAS(r) = UAS(r_s)$

The above lemma states that all the permissions that can be acquired through role *r* (before splitting) can also be acquired through role r_s.

Proof of Lemma 3.1 is given in appendix

3.3.3 Properties of RBAC-integrate

In this section, we analyze the properties of the policy integration algorithm *RBAC-integrate* in the context of the policy integration requirements discussed in Section 3.3.1. *RBAC-integrate* satisfies all the policy integration requirements (PIRs) except PIR6. Since conflict resolution is not included in RBAC-integrate, therefore the resulting multi-domain policy may not satisfy all the constraints of input RBAC policies. However, the multi-domain policy obtained after conflict resolution, extensively discussed in Chapter 4, satisfies all the integration requirements. Theorems 3.5, 3.6, and 4.7 provide a formal proof of this claim.

In the following, we first formally define the notion of *redundant role* and then prove that *RBAC-integrate* satisfies the policy integration requirements except PIR 6.

Definition 3.2: Let r_d be a role; r_d is said to be a redundant role if the following conditions hold:

1. r_d is not assigned to any user.
2. r_d is not assigned any permission.
3. r_d has at least one senior role r such that $r \underset{I}{\geq} r_d$
4. No role r' exists such that $r' \underset{A}{\geq} r_d$
5. No role r'' exists such that $r_d \underset{A}{\geq} r''$

Redundant roles may be created during the process of policy integration. However, these roles can be removed from the integrated RBAC graph using the *remove-role* algorithm shown in Figure 3.5. Following lemma states that removal of a redundant role r_d from a multi-domain RBAC graph G does not affect the security, autonomy, and interoperability allowed in G.

Lemma 3.3: Let G be a multi-domain RBAC graph and r_d be a redundant role in G. Let G' be the RBAC multi-domain graph obtained by removing r_d from G using the remove-role algorithm given in Figure 3.5. The following properties hold with respect to G and G':

1. For any user u such that $u \in domain(r_d)$, the authorization set of u over all the permissions associated with all the inter-domain roles $r \notin domain(r_d)$ remains unchanged.
2. For any two roles $r_x \in domain\ (r_d)$ and $r_x \neq r_d$ and $r_y \in domain\ (r_d)$ and $r_y \neq r_d$, if $r_y \in conf\text{-}rset\ (r_x)$ before the removal of r_d, then $r_y \in conf\text{-}rset\ (r_x)$ after the removal of r_d.
3. For any user u such that $u \in domain(r_d)$, the authorization set of u over all the permissions associated with all the intra-domain roles $r \in domain(r_d)$ remains unchanged.
4. For any user u such that $u \notin domain(r_d)$, the authorization set of u over all the permissions associated with all the roles $r \in domain(r_d)$ remains unchanged.
5. For any two roles $r_x \neq r_d$ and $r_y \neq r_d$, if $r_y \in conf\text{-}rset\ (r_x)$ before the removal of r_d, then $r_y \in conf\text{-}rset\ (r_x)$ after the removal of r_d.

1 and 2 imply that removal of a redundant role does not affect the security and autonomy of the domain containing the redundant role. 3, 4, and 5 imply that removing a redundant role does not affect the interoperation among the component domains

Proof of Lemma 3.3 is given in Appendix.

Lemma 3.4: The multi-domain policy produced by *RBAC-integrate* satisfies PIRs 1 – 4.

Proof of Lemma 3.4 is given in Appendix.

One key requirement in composing a multi-domain policy is that the final outcome of the policy integration step should not be influenced by the order in which policies are integrated. If the integration mechanism depends on the order in which policies are combined, then one must find an integration order that gives maximum interoperation with minimum overhead. However, restricting the integration order may not be an attractive option as in most collaborative environments, domains join or leave collaboration any time. Nevertheless, the proposed policy integration mechanism is

independent of the order in which policies are integrated. We prove this by showing that the policy integration algorithm *RBAC-integrate* is both commutative and associative.

Theorem 3.5 (Commutativity of RBAC-integrate): The policy integration operation performed by *RBAC-integrate* is commutative.

Proof: RBAC-integrate is commutative if for any two domains A and B, *RBAC-integrate*(G_A,G_B) = *RBAC-integrate*(G_B,G_A), where G_A and G_B are the RBAC graphs of domain A and B respectively.

The commutativity of *RBAC-integrate* depends on the commutativity of *role-integrate*. Therefore, we first analyze the algorithm *role-integrate*. *Role-integrate* performs role comparison and linking in a recursive manner. Roles are linked by calling *link* function which is symmetric. Linking of equivalent roles (lines 11 -14 of *role-interate*) and overlapping roles (lines 27 – 31) is symmetric and hence commutative. For the containment case, assume that *contained*(r_B, r_A) is true. When *role-integrate*(r_A, r_B) is called then the code in lines 15 – 18 is executed, and when *role-integrate*(r_B, r_A) is called, the code in lines 21 – 24 is executed. In both cases, role r_A is split and a junior role r_{Aj} is created with $r_A \underset{I}{\geq} r_{Aj}$, and r_{Aj} is linked to r_B with same permission assignment and junior roles. This implies that the containment case is also symmetric and commutative.

It can be proved using induction that *role-integrate*(r_A, r_B) and *role-integrate*(r_B, r_A) produces same number of roles during the process of integration and they have same permission assignment and role-hierarchy. Hence, *role-integrate* is commutative, implying that *RBAC*-integrate is commutative. ■

Theorem 3.6 (Associativity of *RBAC-integrate*): The policy integration operation performed by RBAC-integrate is associative.

Proof:

Let G_A, G_B, and G_C be the RBAC graph of domain A, B, and C respectively.

P = RBAC-integrate(G_A, G_B)

Q = RBAC-integrate(G_B, G_c)

X = RBAC-integrate(P, G_c)

Y = RBAC-integrate(G_A, Q)

To prove that policy integration operation is associative, we need to prove that the graph X is *isomorphic* to Y. Two policy models are said to be *isomorphic* if there is 1:1 onto correspondence between their elements and they have the same relationships [Pot03]. To show that two final integrated policy models X and Y are isomorphic, we define a morphism $\varphi(X \rightarrow Y)$ as follows:

- For a user $u_i \in X$, $\varphi(u_i) = u_i$
- For a permission $p_j \in X$, $\varphi(p_j) = p_j$
- For a role $r' \in X$, $\varphi(r') = r$ such that $pset_{assign}(r') = pset_{assign}(r)$

In order to prove that φ is an isomorphism we need to show the following:

(i) φ is 1:1 and onto

(ii) $R(U) \in R_X$ if and only if $R(\varphi(U)) \in R_Y$ (U is a vector).

The Appendix Section contains a detailed proof of (i) and (ii).

Theorems 3.5 and 3.6 imply that the multi-domain policy composed by *RBAC-integrate* is independent of the order in which domain policies are integrated.

3.3.4 Time Complexity of RBAC-integrate

The algorithm *RBAC-integrate* runs in polynomial time, as evident from the following Lemmas and Theorem:

Lemma 3.7: If role graphs representing domains' RBAC policies are acyclic, then the algorithm *role-integrate* terminates.

Proof: Given two acyclic role graphs to be integrated, suppose that the algorithm does not terminate, i.e., *role-integrate* is called recursively for an infinite number of times. This implies that there is a cycle in one or both of role graphs. Creation of new roles does not create any cycle as a newly created role is never made a parent of an existing role. Therefore, the cycle must be present in the input role graph(s) which is a contradiction of our initial assumption. Hence the algorithm *role-integrate* terminates. ■

Lemma 3.8: The worst case complexity of *role-integrate* is $O(|P|^3)$, where |P| is the cardinality of the permission set.

Proof: According to the above lemma, the recursive algorithm *role-integrate* terminates. Therefore, we can build a recursive tree in which each node corresponds to the pair of cross-domain roles to be compared. The predicate *not-compared-previously* in lines 4 and 7 ensures that inter-domain roles are compared only once. If |R1| and |R2| denotes the total number of roles in their respective domains, then the total number of role comparisons made by *role- integrate* while merging the two domains are |R1|x|R2|. Note that |R1| and |R2| also include newly created roles. However, no more that |P| number of roles can be created. Therefore at most $O(|P|^2)$ comparison are made in the integration step. Suppose that all the comparisons result in linking the roles under consideration. In the process of linking roles, the conflicting role sets are updated. In the worst case the conflicting set is updated for all roles. This implies that the time complexity of *link* is $O(|P|)$. In the worst case, *link* is called after each comparison. Therefore, the complexity of role-integrate is $O(|P|^3)$. ■

Corollary 3.9: The worst case complexity of RBAC-integrate is $\boldsymbol{O(n|P|^3)}$, where *n* is the number of input domains. ■

4 OPTIMAL CONFLICT RESOLUTION

The policy integration algorithm described above takes as input the RBAC policies of the domains and creates an integrated multi-domain policy which allows inter-domain role accesses and is homogeneous in terms of role hierarchies and permission assignment. However, the multi-domain policy created in this phase may be inconsistent and may not completely satisfy the component domains' security requirements. Moreover, security administrator(s), in charge of the global security policy, can define additional security constraints and specify both permitted and restricted inter-domain role accesses. These additional constraints may also conflict with the access control policies of individual domains. For instance, in Figure 4.6, allowing role *LSO* from *CCO* to inherit the permissions of role *DTA* from *CTO* (shown as dashed-dot arrow from *LSO* to *DTA* in Figure 4.6) will violate the role specific SoD constraints between roles *DTA* and DTM_{10}. This inter-domain access constraint will enable user u_6 to access role *DTA* through the role *LSO*. Also the presence of link between roles $R10_{11}$ and DTM_{10} allows user u_6 to access role r_{10}. This is a violation of SoD constraint defined between roles *DTA* and *DTM* in the original domain policy.

The solution to this problem is to remove either the unidirectional link (*LSO* – *DTA*) or the link ($R10_{11} - DTM_{10}$). This raises an important question: which accesses from the set of conflicting accesses should be removed such that the security and autonomy requirements of constituent domains are not violated? Although, removing link(s) resolves conflicts in the given policy, it also changes the set of allowable accesses and a poor choice of removable inter-domain links may significantly reduce interoperation among the collaborating domains. A conflict resolution mechanism is needed that resolves the conflicts among the collaborating domains in an optimal manner. The problem of conflict resolution in a given multi-domain RBAC policy can be formulated as an optimization problem with the objective of maximizing permitted accesses according to some pre-specified optimality criterion. Various optimality measures such as

maximizing direct or indirect accesses or minimizing the set of relaxed inter-domain access constraints can be used.

4.1 IP Formulation of a Multi-Domain RBAC Policy

In the following, we describe an approach for formulating the multi-domain policy integration problem into an integer program (IP). The proposed IP formulation is generic in the sense that it can work for any of the above mentioned optimality criteria. Changing the optimality measure in our formulation only requires changing the weights in the objective function.

In the IP formulation of RBAC policy, all the constraints such as hierarchical, SoD, permitted and restricted access constraints are defined using linear equations. The variables used in these equations convey both user and role information. For instance, the variables are of the form u_{ir_j} where the first subscript *i* identifies the user and the second subscript r_j specifies the role. The variable u_{ir_j} is a binary variable, *i.e.*, it can take a value of '0' or '1' only. If the variable $u_{ir_j} = 1$ then user u_i is authorized for role r_j, otherwise u_i is not authorized for r_j and cannot access role r_j by any means. If user u_i and role r_j are from different domains and $u_{ir_j} = 0$ then in the role graph, there should not be any path from the user node u_i to the role node r_j. Note that the given multi-domain RBAC policy may be inconsistent and a path may exist between user u_i from one domain and role r_j from another domain, and in the solution to the IP problem $u_{ir_j} = 0$. This inconsistency is resolved by dropping an inter-domain edge that lies in the path between the user node u_i and role node r_j.

4.1.1 Constraint Transformation Rules

In the following, we list the transformation rules to generate IP constraint equations for an RBAC policy. In specifying the rules we denote by U_k and R_k the set of users and roles of domain k respectively; we also denote by U the union of all U_ks and by R the union of all R_ks.

1. For each domain *k*, if a user $u_i \in U_k$ is not authorized for a role $r_j \in R_k$ by the access control policy of domain *k* then $u_{ir_j} = 0$.
2. For a user $u_i \in U$ and role $r_j \in R$, if *domain*(u_i) ≠ *domain*(r_j) and u_i cannot inherit the permissions of role r_j then $u_{ir_j} = 0$.
3. Let A_u be the set of users assigned to a role r_j. There should be at least one user from the set A_u that is able to access role r_j. Formally, $\sum_{u_i \in A_u} u_{ir_j} > 0$.
4. Suppose $u_{ir_j} = 1$ and there exists a role r_k such that *domain*(r_j) = *domain*(r_k) and $r_j \geq_I r_k$, then u_i is also authorized to access role r_k, i.e., $u_{ir_k} = 1$.
5. Consider a user u_i and a role r_k such that *domain*(u_i) ≠ *domain*(r_k). Let R_m be a set of roles such that for all $r_m \in R_m$, *domain*(r_m) = *domain*(r_k). Also, in the RBAC graph, there is a path from u_i to r_m and $r_m \geq_I r_k$. We define two roles sets R_c and R_{pc} as follows:

   $R_c = \{r \mid r \geq_I r_k \wedge domain(r_k) \neq domain(r)\}$

   $R_{pc} = \{r_p \mid \exists r \in R_c \text{ such that}(r_p = r \wedge u\_assign(u,r)) \vee (r_p \geq_I r \wedge domain(r) = domain(r_p))\}$

   The following constraint equations define the conditions for a user u_i to access role r_k.

   a. $\forall r_m \in R_m,\ u_{ir_m} - u_{ir_k} \leq 0$

   b. $\sum_{r_m \in R_m} u_{ir_m} + \sum_{r_n \in R_c} u_{ir_n} - u_{ir_k} \geq 0$

   c. $\sum_{r_m \in Rm} u_{ir_m} + \sum_{r_p \in R_{pc}} u_{ir_p} - u_{ir_k} \geq 0$

   The above set of constraint implies that a user u_i may access a cross domain role r_k only if one of the following two conditions holds:

   i. u_i is authorized for a cross domain role r_m such that *domain*(r_m) = *domain*(r_k) and $r_m \geq_I r_k$.

   ii. u_i is authorized for role r_n and there is an inter-domain edge from r_n to r_k.

   Condition 5c is necessary to avoid any localized assignment of 1 to variables u_{ir_k} and u_{ir_n}, where $u_{ir_n} \in R_c$

6. Consider any two users u_i and u_j and a role r_k. Suppose u_i is authorized to access role r_k, *i.e.* $u_{ir_k} = 1$. Suppose that a cross-domain link exists from role r_k to role r_l. If user u_i is able to access r_l through the cross domain link (r_k, r_l), then user u_j, if authorized for role r_k, can also access r_l through the link (r_k, r_l). Formally:

   $$if\ \ dom(u_i) = dom(u_j) = dom(r_k)\ \text{ then }\ \left(u_{ir_k} - u_{ir_l}\right) - \left(u_{jr_k} - u_{jr_l}\right) \ = 0$$

   $$else\ \ \left(u_{ir_k} - u_{ir_l}\right) - \left(u_{jr_k} - u_{jr_l}\right) \ \geq \ 0$$

7. A role specific *SoD* constraint may exist between two intra-domain or inter-domain roles. In the graph model, *SoD* constraint between two conflicting roles r_j and r_k is represented by a double-headed arrow between roles r_j and r_k. In the IP formulation, this *SoD* constraint can be written as:

   $u_{ir_j} + u_{ir_k} \leq 1$, for all users u_i such that u_i can access either r_j or r_k

8. Suppose that a *SoD* constraint exists between two intra-domain roles r_m and r_n induced by a cross-domain roles r_k and r_l. This *induced SoD* constraint can be written in equation form as:

   $u_{ir_m} + u_{ir_n} + u_{ir_k} + u_{ir_l} \leq 3$, for all users u_i such that u_i can access either r_m or r_n

9. Let U_{kc} be the set of conflicting users for role r_k. At most one user in the set U_{kc} is allowed to access/activate role r_k at any given time. Formally:

   $$\sum_{u_i \in U_{kc}} u_{ir_k} \leq 1$$

4.1.2 Optimality Criteria

The IP constraints described in the above section are used to define security requirements of collaborating domains' RBAC policies. Once the RBAC constraints are transformed into linear IP constraints by using the above transformation rules, the multi-domain RBAC policy can be formulated as the following integer programming problem.

$$\text{maximize} \quad c^T u_r$$

$$\text{Subject to} \quad Au_r \leq b$$

$$\forall u_{ir_j} \in u_r, u_{ir_j} = 0 \text{ or } 1$$

Where, *c* is the cost vector and *A* is the constraint matrix. The cost vector *c* defines the optimality criteria. The main purpose of formulating the multi-domain RBAC policy into an IP problem is to find a feasible solution (a set of users having permission to role accesses that do not violate the given security requirements of individual domains) that maximizes the objective function according to given optimality criterion. One of the optimality criteria might be to maximize the number of cross domain role accesses. In this case the objective function is the sum of all variables defining inter-domain user to role accesses.

Maximizing inter-domain accesses may lead to relaxing or dropping some of the administrator-specified constraints which may not be desirable in certain situations. When administrative constraints are to be preserved, the element of cost vector corresponding to the administrator specified constraint is assigned a higher value.

4.2 Autonomy Consideration

One key requirement of policy integration is to maintain the autonomy of all collaborating domains. However, preserving the autonomy of individual domains may significantly reduce interoperation and in some cases may not allow interoperation at all. In other words, there is a trade-off between seeking interoperability and preserving autonomy. In the RBAC policy integration framework, violation of a domain's autonomy occurs because of the following two reasons:

Induced SoD constraint: An *induced SoD* constraint as mentioned in Chapter 3 is a *SoD* constraint between two intra-domain roles r_a and r_b which do not conflict with each other in their original domain's RBAC policy. Such a *SoD* constraint is caused by a cross-domain roles r_c and r_d for which the following hold:

$\text{domain}(r_c) \neq \text{domain}(r_a) = \text{domain}(r_b)$

$\text{domain}(r_d) \neq \text{domain}(r_a) = \text{domain}(r_b)$

$$conf - rset(r_c, r_d) \wedge \left[(r_a \geq_I r_c \wedge r_b \geq_I r_d) \quad \vee \quad (r_b \geq_I r_c \wedge r_a \geq_I r_d) \right]$$

Figure 4.2(a) illustrates an *induced SoD* constraint between roles r_2 and r_3 of domain A caused by roles r_4 and r_5 of domain B. Note that in the original RBAC policy of

domain As shown in Figure 4.2(b), r_2 and r_3 are non-conflicting. As a result of this *induced SoD* constraint, user u_1 who in the domain A's original policy is authorized to access role r_2 and r_3 simultaneously, cannot access these roles in concurrent sessions in the multi-domain environment.

Asymmetric cardinality of equivalent roles: There are various types of cardinalities associated with a given role, for instance, role-assignment cardinality, role-activation cardinality, per-user role-assignment cardinality, and per user role activation cardinality [Jos03]. For simplicity of discussion, we only consider role-activation cardinality which is defined as the maximum number of concurrent accesses of a role allowed by a given RBAC policy. Two cross-domain equivalent roles r_a and r_b are said to be asymmetric in their cardinality if they differ in their activation cardinalities. In order to establish interoperability between two cross-domain equivalent roles that are asymmetric in their activation cardinalities, the most restrictive cardinality constraint from the two roles is taken and is applied to both of them. For instance, if r_a has a cardinality constraint of one and r_b has a cardinality constraint of three, then the most restrictive cardinality constraint is one and should be applied to both r_a and r_b if they are to be made interoperable. Adding the most restrictive cardinality constraint may violate the autonomy of one or more of the collaborating domains. On the other hand, retaining the original cardinalities of interoperable roles may lead to security violation which is unacceptable. Obviously, the third option is to disallow any cross-domain accesses via roles with asymmetric cardinalities. This option reduces interoperation between two otherwise similar cross-domain roles. Figure 4.6 depicts the trade-off between interoperability and autonomy in a graphical manner. A discussion on this graph is presented in Section 4.4.

In general, composition of a global multi-domain policy that allows interoperation among multiple domains without any violation of collaborating domains' security and autonomy is not a feasible task. In almost any collaborative environment, violation of any domain's security policy is not permissible at all. However, domains may be willing to compromise their autonomy for the sake of establishing more interoperability. In particular, the autonomy violations described in the context of RBAC policy integration are less critical and can be tolerated. Nevertheless, if a domain's RBAC policy does not

allow any autonomy violation of one or more of its roles, then such roles are not made interoperable with other similar cross-domain roles that either induce *SoD* constraints or have different role cardinalities.

In a multi-domain environment in which certain autonomy violations can be tolerated, the objective of the conflict resolution phase is to maximize interoperation according to the given optimality criterion with a minimum loss of autonomies of collaborating domains. This goal of minimizing autonomy loss is reflected in the objective function of the IP problem by assigning a lower weight to user-role variables that result in autonomy violation and a relatively higher weight to user-role variables that do not cause violation of any sort. The following example illustrates this point in more detail.

Example 4

Consider two collaborating domains A and B with their respective RBAC policies shown in Figure 4.2(a). The multi-domain RBAC policy that allows inter-domain accesses between A and B is shown in Figure 4.2(b). The link from r_3 to r_5 and an administrator-specified access constraint that allows role r_5 to inherit permission of role r_1 make this multi-domain policy inconsistent. Note that the *SoD* constraint between r_2 and r_4 is an *induced SoD* constraint and limits the autonomy of user u_1. The conflict in this multi-domain policy can be resolved by either removing the edge (r_3, r_5) or (r_5, r_1). In both cases the number of cross-domain accesses will remain the same. However, removing (r_3, r_5) is preferred over (r_5, r_1) as it retains the autonomy of u_1 over roles r_2 and r_3. The IP formulation of the multi-domain policy of Figure 4.2(b) is shown in Figure 4.1. Note that in the objective function, the variables u_{1r_4}, u_{1r_5}, u_{2r_4}, and u_{3r_5} are assigned a lower weight than the remaining variables in the objective functions. These variables tend to retain the link from r_2 to r_4 and from r_3 to r_5, which prohibits user u_1 to access r_2 and r_3 simultaneously - a violation of domain A's autonomy. An optimal solution to the IP problem shown in Figure 4.1 has following values of cross-domain variables. $u_{1r_4} = 0,\ u_{1r_5} = 0,\ u_{2r_4} = 1,\ u_{3r_5} = 0,\ u_{4r_2} = 1,\ u_{5r_1} = 1,\ u_{5r_3} = 1,\ u_{5r_6} = 1$.

Since $u_{3r_3} = 1$ (constraint c9 in Figure 4.1), and $u_{3r_5} = 0$, the cross-domain edge (r_3,r_5) needs to be dropped from the multi-domain RBAC graph of Figure 4.2(b).

4.3 Conflict Resolution Algorithm

Figure 4.3 shows an algorithm *ConfRes* for resolving conflicts from the RBAC graph G representing the multi-domain policy. This algorithm first transforms the RBAC policy constraints into IP constraints using the rules given in Section 4.1.1. Before transforming RBAC policy constraints into IP constraints, dummy users are assigned to two classes of roles which do not have any user assigned to them. Class one includes those roles which do not have any senior role in the inheritance hierarchy semantics. Assignment of dummy users to class one roles ensures that all the roles appear in the IP constraint equations, which is essential for conflict resolution. Class two includes roles which have a non-empty set of conflicting users. The dummy user u_{dj} assigned to a class two role r_j is also included in all the conflicting sets of users for role r_j. Since u_{dj} is the only user assigned to r_j therefore $u_{djr_j} = 1$ (by transformation rule 2).This prohibits any user u_k that conflicts with u_{dj} for role r_j to inherit the permissions of r_j through a senior role r_s without activating r_j. Once all the IP constraints are defined, the IP problem is solved using the optimality criterion embedded in the objective function. Based on the solution of the IP problem, the graph G is modified by removing the conflicting cross-domain edges and induced SoD constraints. The resulting graph defines the multi-domain policy that satisfies the security requirements of all collaborating domains. This is formally proved in Chapter 5.

Maximize $u_{1r_4} + u_{1r_5} + u_{2r_4} + u_{3r_5} + 2u_{4r_2} + 2u_{5r_1} + 2u_{5r_3} + 2u_{5r_6}$

Subject to

Constraints derived from rules 1, 2, 3, and 4

c1: $u_{1r_1} = 1$, c2: $u_{1r_6} = 1$, c3: $u_{2r_1} = 0$, c4: $u_{2r_2} = 1$, c5: $u_{2r_3} = 0$, c6: $u_{2r_6} = 0$,
c7: $u_{3r_1} = 0$, c8: $u_{3r_2} = 0$, c9: $u_{3r_3} = 1$, c10: $u_{3r_6} = 0$, c11: $u_{4r_4} = 1$, c12: $u_{4r_5} = 0$,
c13: $u_{5r_4} = 0$, c14: $u_{5r_5} = 1$, c15: $u_{2r_5} = 0$, c16: $u_{3r_4} = 0$, c17: $u_{4r_1} = 0$, c18: $u_{4r_3} = 0$,
c19: $u_{4r_6} = 0$, c20: $u_{5r_2} = 0$

Constraints derived from rule 5

c21: $u_{1r_2} - u_{1r_4} \geq 0$, c22: $u_{1r_3} - u_{1r_5} \geq 0$, c23: $u_{2r_2} - u_{2r_4} \geq 0$, c24: $u_{3r_3} - u_{3r_5} \geq 0$,
c25: $u_{5r_5} - u_{5r_1} \geq 0$, c26: $u_{5r_5} - u_{5r_3} \geq 0$, c27: $u_{5r_1} - u_{5r_6} = 0$, c27: $u_{4r_4} - u_{4r_2} \geq 0$

Constraints derived from rule 6

c28: $u_{3r_3} - u_{3r_5} - u_{1r_3} + u_{1r_5} = 0$, c29: $u_{2r_2} - u_{2r_4} - u_{1r_2} + u_{1r_4} = 0$

c30: $u_{5r_5} - u_{5r_1} - u_{3r_5} + u_{3r_1} \geq 0$

Constraints derived from rule 7

c31: $u_{4r_4} + u_{4r_5} \leq 1$, c32: $u_{5r_4} + u_{5r_5} \leq 1$, c33: $u_{1r_4} + u_{1r_5} \leq 1$, c34: $u_{2r_4} + u_{2r_5} \leq 1$,
c35: $u_{3r_4} + u_{3r_5} \leq 1$, c36: $u_{1r_2} + u_{1r_5} \leq 1$, c37: $u_{2r_2} + u_{2r_5} \leq 1$, c38: $u_{4r_2} + u_{4r_5} \leq 1$,
c39: $u_{3r_3} + u_{3r_4} \leq 1$, c40: $u_{1r_3} + u_{1r_4} \leq 1$: c41: $u_{5r_3} + u_{5r_4} \leq 1$

Induced SoD Constraint derived from rule 8

c42: $u_{1r_2} + u_{1r_3} + u_{1r_4} + u_{1r_5} \leq 3$

Fig. 4.1 IP formulation of multi-domain RBAC policy shown in Fig. 4.2

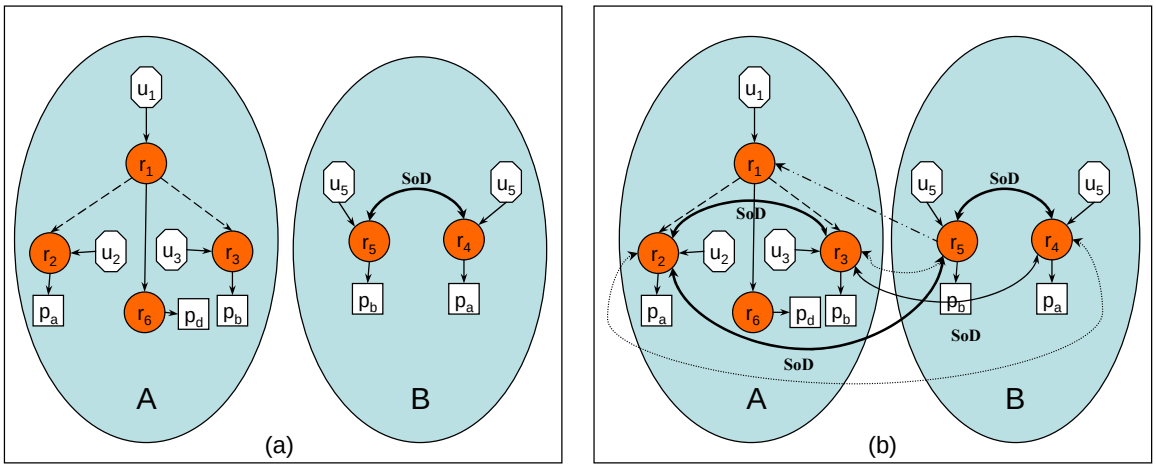

Fig. 4.2 (a) RBAC policy graph of domain A and B in example 4, (b) Integrated RBAC policy defining interoperation between domains A and B.

ConfRes(G)

1. Assign a dummy user u_{di} to all roles r_i for which the following hold:
   a. No user is assigned to r_i.
   b. There does not exist any role r_k for which $r_k \geq_I r_i$.
2. Assign a dummy user u_{dj} to all roles r_j which have a non-empty set of conflicting users.
3. For each role r_j that is being assigned a user u_{dj} in step 2, set $u_{djr_j} = 1$ and update the conflicting set of users by doing the following:
   a. Define user-specific SoD constraint between u_{dj} and all the conflicting users for role r_j that are not assigned to r_j.
   b. add new conflicting set(s) of user for role r_j containing the dummy user u_{dj} and a user u_k for which the following holds:

   $$\exists u_i \in U, r_i \in R \text{ such that } [(r_i \geq_A^t r_j) \wedge conf-user(u_i, u_k, r_i) \wedge (u_i \in conf-user(r_j))]$$
4. Using the constraint transformation rules, write the RBAC policy constraints in algebraic form.
5. Define the objective function.
6. Find an optimal feasible solution for the integer programming (IP) problem.
7. From the multi-domain RBAC policy graph G, remove the inter-domain edge (r_i, r_j) for which there exists a user u_k such that $u_{kr_i} = 1$ and $u_{kr_j} = 0$ in the optimal feasible solution.

   $$G = G - \{(r_i, r_j) \mid \exists u_k \in U \text{ such that } u_{kr_i} = 1 \text{ and } u_{kr_j} = 0 \text{ and } domain(r_i) \neq domain(r_j)\}$$
8. For an edge (r_i, r_j) removed from G, if r_j induces an SoD constraint between r_i and any role r_k, then remove that induced SoD constraint from RBAC policy graph G
9. From the graph G, remove the conflicting set of users added in step 3b.

Fig. 4.3 Conflict resolution algorithm

4.4 An illustrative example

In this section, we illustrate the proposed policy integration framework by considering interoperation/collaboration among various offices of a county for collection and sale of real-state tax on property parcels located within the jurisdiction of concerned county. The concerned county offices include: *County Clerk Office* (CCO), *County Treasure Office* (CTO), *County Attorney Office* (CAO), *District Clerk Office* (DCO), and *District Courts* (DC). These offices/departments share information among each other for budget planning, tax billing and collection, sale of delinquent taxes, auditing and other legal purposes. Each county office keeps the information owned by it in its local databases. Integration of these local databases is needed to provide inter-domain

information access capability. Such an integration not only expedite the process of tax collection and sale by providing immediate access to timely, accurate, and complete information, but also improves the productivity of existing staff by reducing redundant data collection efforts among the county departments.

In order to establish interoperation among various county offices, the access control policies of the collaborating county offices need to be integrated. Due to space limitation, we only focus on interoperation among three county offices: CCO, CTO, and CAO. Table 4.1 lists the roles, job description and permissions associated with each role of all three county offices. The permission authorization in Table 4.1 defines the access rights or permissions available to the corresponding roles on local as well as cross-domain information objects. As mentioned in Chapter 3, an information sharing policy is needed that explicitly specifies the access rights available to cross-domain-roles over a local object and the conditions under which such access is granted. Table 4.2 shows the information sharing policy of information/data objects that can be shared among the collaborating county offices. The letters W, R, and A in the access mode columns indicate *write*, *read*, and *approve* respectively. Note that in the information sharing policy listed in Table 4.2, domains that own information objects do not indicate the actual foreign domain roles that can inherit the permissions of their local objects. Rather the owner domains only specify the conditions that must be fulfilled by cross-domain roles in order to access foreign objects. Listing the prospective cross-domain roles that can access a given object is too cumbersome and requires the knowledge of the organization hierarchy and access control policies of other collaborating domains. Acquisition of this knowledge may not be feasible as domains may not be willing to reveal their access control policies to others. It is therefore the responsibility of the policy integration mechanism to determine the roles that satisfy the condition for accessing each others information objects and link them accordingly.

Figure 4.6 shows the integrated RBAC policy after applying the policy integration operation, *RBAC-Integrate*, on the RBAC graphs shown in Figure 4.5. This integrated policy corresponds to the output of the policy integration step of Figure 1.1. In Figure 4.6, the dotted double headed arrow between two cross domain roles indicates that these

roles are equivalent and can inherit the permissions of each other. For instance in Figure 4.6, role DTM_{10} of CTO is equivalent to $R10_{11}$. This implies that a user of CTO domain can access the permissions of role $R10_{11}$ through DTM_{10}; similarly, a user who is authorized for role $R10_{11}$ can also inherit the permissions of role DTM_{10} through $R10_{11}$. Note that cross-domain roles are related by the *I-hierarchy* semantics only, which implies that user u_1 of CTO cannot access the permissions of role $R10_{11}$ without gaining access of role DTM_{10}. Also, user u_6 can access role DTM_{10} only if it has access over role $R10_{11}$.

In addition to the cross-domain links produced by *RBAC-integrate* between equivalent roles, Figure 4.6 also defines access constraints specified by the global security administrator(s). These administrator specified access constraints are depicted as dash-dotted arrows. In Figure 4.6, administrator-specified access constraints include the following edges: (TA, TAO), (PIO, TRA), (LSO, DTA), (DTLO, DTC), (TAC, DTA), (DTA, ACAT), (ACAT, TAC), and (DTM, ACAT). An administrator specified access constraint edge (r_a, r_b) implies that role r_a inherits the permissions of the cross-domain role r_b. Similar to the cross-domain link between equivalent roles, roles r_a and r_b in the administrator-specified constraint edge (r_a, r_b) are related according to the *I-hierarchy* semantics.

The multi-domain policy shown in Figure 4.6 is conflicting and does not satisfy the security requirement of the collaborating county offices. For instance, the administrator specified access constraint edge (TA, TAO) conflicts with (PIO, TRA). If both of them are retained then a violation of *SoD* constraint between TRE and TRA occurs, enabling user u_2 to access role TRE and TRA simultaneously. Similarly, the cross-domain edge (LSO, DTA) conflicts with (DTLO, DTC) and ($R10_{11}$, DTM_{10}). These cross-domain access constraints allow user u_6 to access roles DTA and DTM_{10} in concurrent sessions, which is again a violation of *SoD* constraint defined between roles DTA and DTM_{10}. Note that in the original RBAC policy of CTO, a *SoD* constraint is defined between DTA and DTM (see Figure 4.5). Since DTM splits into roles DTM_{10} and DTM_{12}, therefore these roles also conflict with DTA. The administrator specified access constraint edge (DTA, ACAT) and the link ($PLAT_9$, DTM) allows u_4 to access the role DTM which is a violation of *role-assignment constraint* as user u_4 is assigned role DTA which is junior to

role DTM. The cross-domain edges (DTM, ACAT), (ACAT, TAC), and (TAC, DTA) results in a violation of the *SoD* constraint defined between roles DTM and DTA. These cross-domain edges enable u_1 to access DTM and DTA simultaneously.

Conflicts in the multi-domain policy shown in Figure 4.6 are resolved by applying the conflict resolution algorithm *ConfRes*. *ConfRes* first transforms the RBAC policy constraints into IP constraints. This IP constraint transformation process produces almost 1500 constraints for the multi-domain RBAC policy of Figure 4.6. The resulting IP problem is solved with the objective of maximizing all cross-domain accesses. The solution thus obtained removes the following edges from the multi-domain policy graph of Figure 4.6: (DTM, ACAT), (TAC, DTA), (DTA, ACAT), (PIO, TRA), and (LSO, DTA). A maximum of 102 cross-domain accesses are obtained if the above edges are removed. Note that in this case, all the cross-domain accesses are assigned equal weight in the objective function. If some cross-domain accesses are more important than others then such accesses can be prioritized by assigning them a higher weight in the objective function. This will increase the likelihood of retaining high priority accesses in the multi-domain policy. However the total number of accesses cannot exceed the maximum value obtained by assigning uniform weights to all cross-domain accesses.

Figure 4.7(a-b) shows the trade-off between interoperability and autonomy for the domains CTO and CCO respectively. The role cardinalities and user assignment used in the measurement of interoperability and autonomy parameters are given in Table 4.3. A role with a cardinality of *n* implies that no more than *n* users can access that role concurrently. For this study/analysis, interoperability of a domain is defined as a measure of the number of cross-domain accesses to that domain. Autonomy of a domain at any given interoperability level is determined by the *autonomy loss* (*AL*) metric defined as:

$$AL(N) = \frac{\begin{pmatrix}\text{Total number of local accesses}\\ \text{with no cross-domain accesses}\end{pmatrix} - \begin{pmatrix}\text{Total number of local accesses}\\ \text{with } N \text{ cross-domain accesses}\end{pmatrix}}{\begin{pmatrix}\text{Total number of local accesses}\\ \text{with no cross-domain accesses}\end{pmatrix}}$$

In the interoperability versus autonomy loss graph, depicted in Figure 4.7, interoperability of a domain X is varied by selecting different sets of cross-domain edges from the set of edges E_X. The set E_X is given by:

$$E_X = \{(r_a, r_b) \mid (r_a \notin X, r_b \in X\ ) \wedge\ ((r_a, r_b) \in G) \wedge$$
(G is a secure multi-domain graph obtained after applying ConfRes algorithm)}

The graph shown in Figures 4.7 contains two curves defining the upper bound and lower bound for the autonomy losses at various interoperability levels. At any given interoperability level, there can be multiple values of autonomy losses corresponding to different selection of cross-domain edges from the set E_X. However, all the autonomy loss values are confined to the region bounded by the upper bound and lower bound curves shown in Figure 4.7. The output of the conflict resolution algorithm gives a set of secure edges that maximizes interoperability. However, the maximal interoperability point may result in a greater loss of autonomy. For instance in Figure 4.7(b), a maximum interoperability level of 43 results in an autonomy loss of 53%. This can be reduced to 29% by selecting a different operating point with an interoperability value of 36. In order to minimize autonomy losses, one should always select an operating point that lies on or close to the lower bound curve. However, this may result in compromising some of the prioritized cross-domain accesses due to the removal of the prioritized cross-domain links.

The drastic variations in autonomy losses with a very small or no change in interoperability level is due to the different selection of cross-domain edges/links. For instance, on the lower bound curve of Figure 4.7(a) when the interoperability level increases from 30 to 31, the autonomy loss increases from 17% to 33%. To explain this drastic variation, we refer to the points (30, 17%) as A and (31, 33%) as B, and the corresponding set of cross domain edges at these operating points as A_{CTO} and B_{CTO} respectively. The difference in this autonomy loss is due to the inclusion of the cross-domain edge ($R10_{11}$, DTM_{10}) in B_{CTO}. As a result, user u_{30}, u_{32}, u_{36}, and u_{37} from domain CCO can now access the cross-domain role DTM_{10} which has a cardinality of four. Please refer to Table 4.3 for user assignments and role cardinalities. As a result of these cross-domain accesses, none of the users from domain CTO can access the role DTM_{10}

and all the roles senior to DTM_{10} in the *I-hierarchy* semantics. This increases the autonomy loss of CTO domain from 17% to 33% at point B.

In the above case, autonomy loss is due to the addition of a cross-domain edge. Addition of a cross-domain edge may also reduce autonomy loss. For instance, consider the points C(39,38%) and D(39,33%) in Figure 4.7(a). The set of cross-domain edges corresponding to points C and D are denoted by C_{CTO} and D_{CTO} respectively. The set D_{CTO} contains the cross-domain edge (DTLO, DTC) which is not present in the set C_{CTO}. The role DTLO has a cardinality of five and DTC has a cardinality of four. Linking (DTLO, DTC) reduces the effective cardinality of DTLO by four, implying that only four users can access the role DTLO and the roles that can be reached through DTLO only. The set of users from domain CCO, that can access DTLO includes users u_{30}, u_{32}, u_{36}, u_{37}, and u_{39}. For maximum interoperability at point D, user u_{39} cannot access DTLO implying u_{39} cannot activate the cross-domain role R_1 which has a cardinality of eleven. There are seven users from the domain CTO that are allowed to access R_1. Since only four cross-domain users are allowed to access R_1, therefore none of the local accesses to R_1 is blocked. However, when cross-domain link (DTLO, DTC) is removed, the cardinality of DTLO is restored to five, which allows u_{39} to access the role DTLO. User u_{39} can access role R_1 through the role $DTLO_{00}$ which is junior to DTLO in the *I-hierarchy* semantics. This increases the number of cross-domain accesses to role R_1 by 5, implying that one local access to role R_1 needs to be blocked. This blocked local access propagates upward in the role hierarchy of domain CTO thus increasing the autonomy loss from 33% to 38%.

Table 4.1 Description of roles involved in collaboration among county offices

Role	Domain	Job Description	Permission Authorization
Treasurer	CTO	Supervises all operations of treasurer office	Inherits all permissions of TCM, TRM, and DTM
Tax Assessor (TA)	CTO	Assess/prepare tax bills	P_6, P_9, P_{10}, P_{11}
Tax Bill Approver (TBA)	CTO	Reassess & approve of tax bils	P_6, P_9, P_{10}, P_{11}, P_{12}
Tax Collector (TC)	CTO	Tax collection & tax sale, record keeping of tax bidders	P_{11}, P_{13}, P_{14}, P_{31}, P_{32}
Tax Collection Manager (TCM)	CTO	supervises TA, TBA, and TC	Inherits all authorized permissions of TA, TB, and TC
Tax Refund Assessor (TRA)	CTO	Assess tax refunds, prepare tax refund orders	P_6, P_9, P_{11}, P_{17}, P_{18}
Tax Refund Examiner (TRE)	CTO	Reassess/approve refund orders	P_6, P_9, P_{11}, P_{18}, P_{19}
Tax Refund Clerk (TRC)	CTO	Prepare refund vouchers	P_{42}, P_{43}
Tax Refund Manager (TRM)	CTO	Approve refund vouchers	P_{42}, P_{43}, P_{44}
Delinquent Tax Clerk (DTC)	CTO	Keep record of delinquent taxes	P_{11}, P_{14}, P_{20}, P_{21}
Delinquent Tax Assessor (DTA)	CTO	Assess delinquent tax records	P_{11}, P_{14}, P_{20}, P_{21},P_{22}
Delinquent Tax Manager (DTM)	CTO	Approve delinquent taxes for sale/resale (supervises DTC & DTA)	Inherit permissions of DTC & DTA, $P_{24}P_{26}P_{27}$,P_{29}, P_{31}, P_{32}, P_{34}, P_{36}
County Clerk	CCO	Supervises all operations of clerk office	Inherits all permissions of PTAM & PDTM
Property Value Assessment Officer (PVAO)	CCO	Property value assessment	P_1, P_2, P_4
Tax Assessment Clerk (TAC)	CCO	Determine property tax rates	P_2, P_4, P_5, P_6, P_9
Tax Assessment Officer (TAO)	CCO	Reassess/approve tax rates	P_2, P_4, P_6, P_7, P_9
Property Tax Assessment Manager (PTAM)	CCO	Supervise TAC & TAO	Inherits permissions of TAC & TAO
Property Indexing Officer (PIO)	CCO	Property indexing	P_2, P_3, P_4
Delinquent Taxes & Lien Officer (DTLO)	CCO	Record keeping of delinquent taxes and other tax liens	P_2, P_4, P_{11}, P_{14}, P_{21}, P_{24}, P_{27}
Lien Sale Officer (LSO)	CCO	Sale of delinquent taxes, keep record of tax buyers	Inherit permissions of DTLO, P_{28}, P_{29}, P_{30}, P_{31}, P_{32}, P_{34}, P_{36}
Redemption Cost Assessor (RCA)	CCO	Prepare redemption cost estimates for delinquent taxes	Inherit permissions of DTLO, P_{29}, P_{31}, P_{34}, P_{35}, P_{36}
Property Delinquent Tax Manager (PDTM)	CCO	Reassess/approve tax redemption cost estimates (supervises LSO & RCA)	Inherit permissions of RCA & LSO, P_{33}, P_{37}
County Attorney	CAO	Heads county attorney department	Permissions of all junior roles
Deputy County Attorney Tax Section (DCAT)	CAO	Assess/approve tax sale plea	Inherits permissions of ACAT, P_{45}
Asst. County Attorney Tax Section (ACAT)	CAO	Prepare tax sale pleas for delinquent taxes and other liens/ Supervise tax sales	Inherits permissions of PLAT, P_{25}
Para Legal tax Section (PLAT)	CAO	Keep records of information obtained from CCO & CTO for tax related affairs, assists attorneys in preparing tax sale pleas	P_2, P_4, P_6, P_9, P_{11}, P_{14}, P_{16}, P_{21}, P_{24}, P_{26}, P_{27}, P_{29}, P_{31}, P_{32}, P_{34},P_{36}

Table 4.2 Information sharing policy of collaborating domains

Information/data Object	Owner domain	Foreign domain	Access Mode available to owner domain	Access mode available to foreign domain	Purpose of access of foreign domain	Condition for cross-domain access
Property value record (O_1)	CCO	CTO, CAO	W:P_1, R:P_2	R:P_2	Property value & tax rate assessment	Access available to subjects dealing with property tax assessment and billing
Property ownership and location record (O_2)	CCO	CTO, CAO	W:P_3, R:P_4	R:P_4	Tax billing, notification	Access available to subjects dealing with tax billing and tax auditing
Tax rate record (O_3)	CCO	CTO, CAO	W:P_5, R:P_6, A:P_7	R:P_6	Tax billing	Access available to subjects dealing with tax billing and tax auditing
Tax exemption record (O_4)	CCO	CTO, CAO	W:P_8, R:P_9	R:P_9	Tax adjustment, billing	Access available to subjects dealing with tax billing, adjustments, refunds and tax auditing
Tax Bill (O_5)	CTO	CCO, CAO	W:P_{10}, R:P_{11}, A:P_{12}	W:P_{10}, R:P_{11}	Auditing, tax readjustment, imposing penalties and fines for non payment or late payment of taxes, checking payment record of tax payers for other purposes	Access available to subjects dealing with tax billing, adjustments, refunds, tax auditing and delinquent taxes and redemption
Tax Payment record (O_6)	CTO	CCO, CAO	W:P_{13}, R:P_{14}	W:P_{13}, R:P_{14}	Auditing, receive payment in certain cases (delinquent taxes, tax/lien sale)	Access available to subjects dealing with tax billing, adjustments, refunds, tax auditing and delinquent taxes and redemption
Refund order (O_8)	CTO	CCO	W:P_{17}, R:P_{18}, A:P_{19}	W:P_{17}, R:P_{18}	Refunds for unsuccessful tax bidders	Access available to subjects dealing with tax refunds and tax sale refunds
Delinquent tax record (O_9)	CTO	CCO, CAO	W:P_{20}, R:P_{21}, A:P_{22}	W:P_{20}, R:P_{21}	Preparing tax sale plea, redemption cost estimates, tax sale, auditing	Access available to subjects dealing with delinquent taxes, tax sale, tax redemption, and tax auditing
Tax Liens (O_{10})	DCO	CCO, CTO, CAO		R:P_{24}	Preparing tax sale plea, redemption cost estimates, tax sale, auditing	Access available to subjects dealing with delinquent taxes, tax sale, tax redemption (write), and tax auditing
Tax Sale Plea (O_{11})	CAO	CCO, CTO	W:P_{25}, R:P_{26}, A:P_{45}	R:P_{26}	Record keeping, identifying pending tax sales awaiting court orders, auditing	Access available to subjects dealing with delinquent taxes, tax sale, tax redemption, and tax auditing
Tax Sale Judgement Order (O_{12})	DCO	CCO, CTO, CAO		R:P_{27}	Record Keeping, tax sale and redemption, auditing	Access available to subjects dealing with delinquent taxes, tax sale, tax redemption, and tax auditing
Tax Sale Record (O_{13})	CTO	CCO, CAO	W:P_{28}, R:P_{29}	W:P_{28}, R:P_{29}	Record Keeping, tax sale and redemption, tax refunds	Access available to subjects dealing with delinquent taxes, tax sale, tax redemption (write), and tax auditing
Tax Buyer Record (O_{14})	CTO	CCO, CAO	W:P_{30}, R:P_{31}	R:P_{30}	Record Keeping, tax redemption, tax refunds, auditing	Access available to subjects dealing with delinquent taxes, tax sale, tax redemption and refunds, and auditing
Tax Redemption Record (O_{15})	CTO	CCO, CAO	W:P_{33}, R:P_{34}	W:P_{33}, R:P_{34}	Record Keeping, tax redemption and refunds, auditing	Access available to subjects dealing with delinquent taxes, tax sale, redemption (Write) and refunds, and tax auditing
Redemption Cost record (O_{17})	CCO	CTO, CAO	W:P_{35}, R:P_{36}, A:P_{37}	W:P_{35}, R:P_{36}	Redemption of delinquent taxes, refunds, auditing	Access available to subjects dealing with delinquent tax redemption (Write) and refunds, and tax auditing

Table 4.3 Cardinality and user assignment of roles used in autonomy loss measurement of Fig. 4.5

Role	Cardinality	User assigned	Role	Cardinality	User assigned	Role	Cardinality	User assigned
Treasurer	1	u_1	R2	11		R7	12	
TCM	2	u_2	R1	11		TAC	4	U_{33}
TRM	2	u_3	$R4_{02}$	7		TAO	4	U_{34}
DTM	2	u_4	$R2_{05}$	11		PVAO	4	U_{35}
TA	3	u_5	$R2_{07}$	11		PIO	8	U_{38}
TBA	3	u_6	DTM_{10}	4		R9	11	
TC	3	u_7	DTM_{12}	5		$DTLO_{00}$	11	
TRE	4	u_8	CC	1	u_{30}	$DTLO_{01}$	11	
TRA	4	u_9	PTAM	2	u_{31}	$R10_{03}$	7	
DTA	4	u_{10}	PDTM	2	u_{32}	LSO_{04}	7	
DTC	4	u_{11}	LSO	5	u_{36}	$R8_{06}$	11	
TRC	8	u_{12}	RCA	5	u_{37}	$R9_{08}$	11	
R6	11		DTLO	6	u_{39}	$DTLO_{09}$	11	
R5	11		R10	9		$DTLO_{13}$	5	
R4	7		R8	9		$R10_{11}$	4	

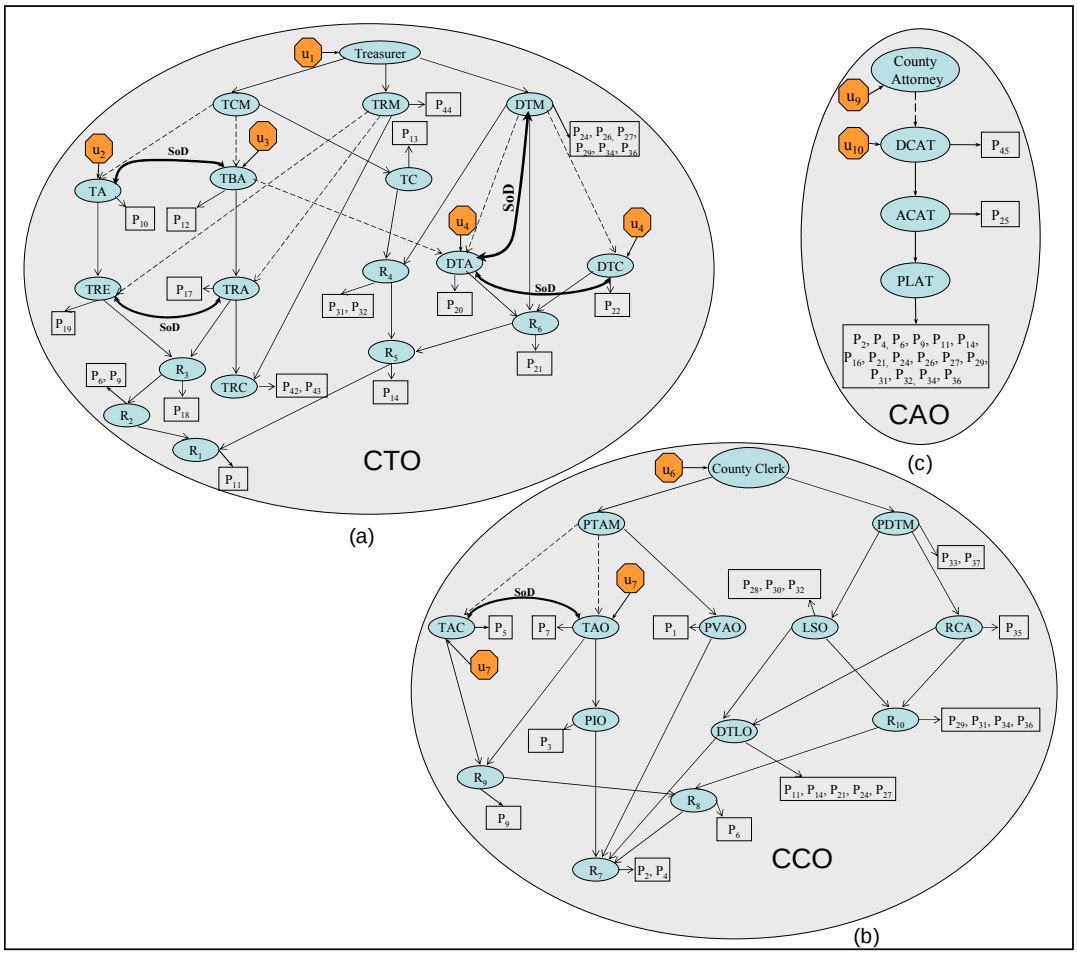

Fig. 4.4 RBAC policy graphs of collaborating county offices

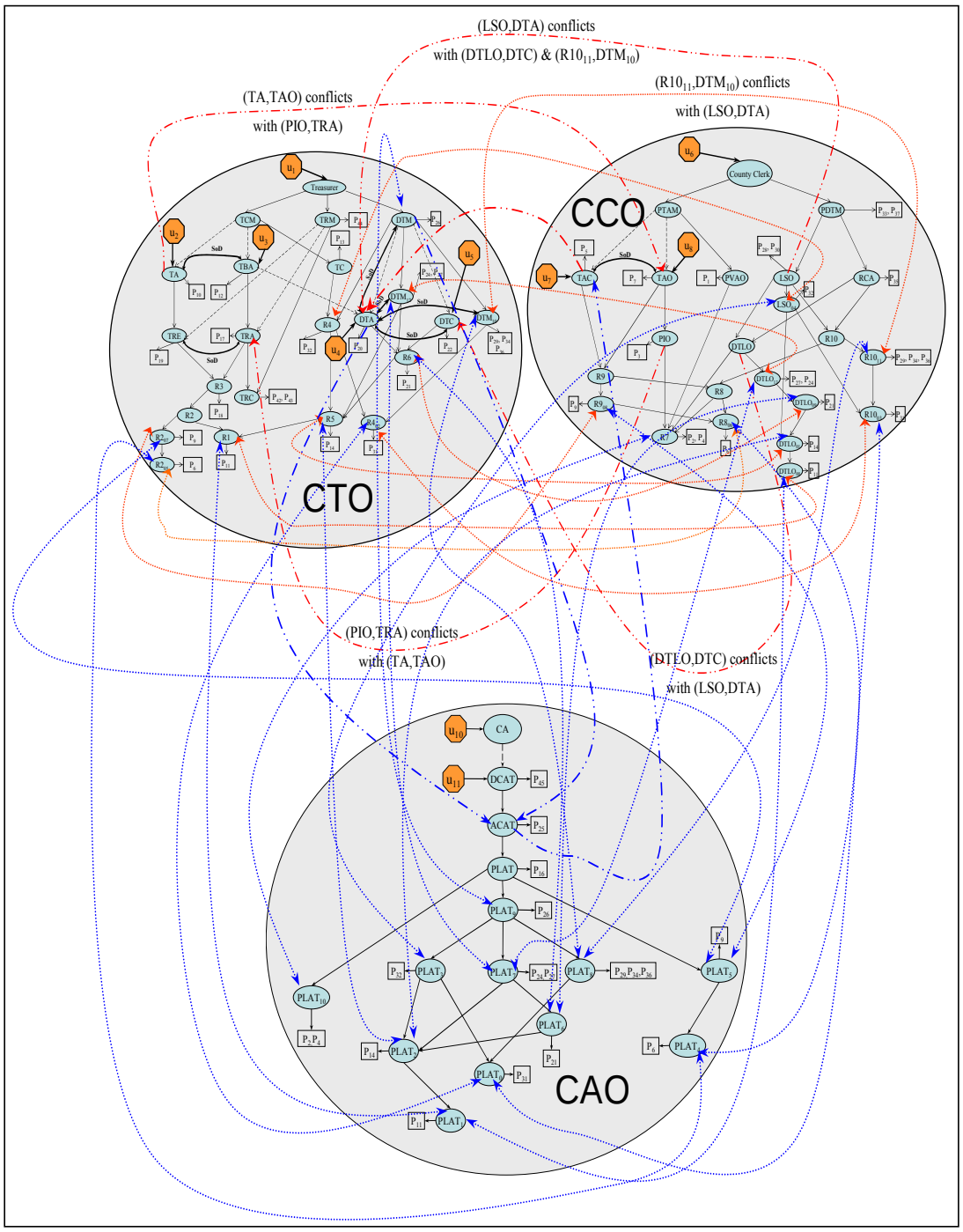

Fig. 4.5 Integrated RBAC policy governing collaboration among the county offices

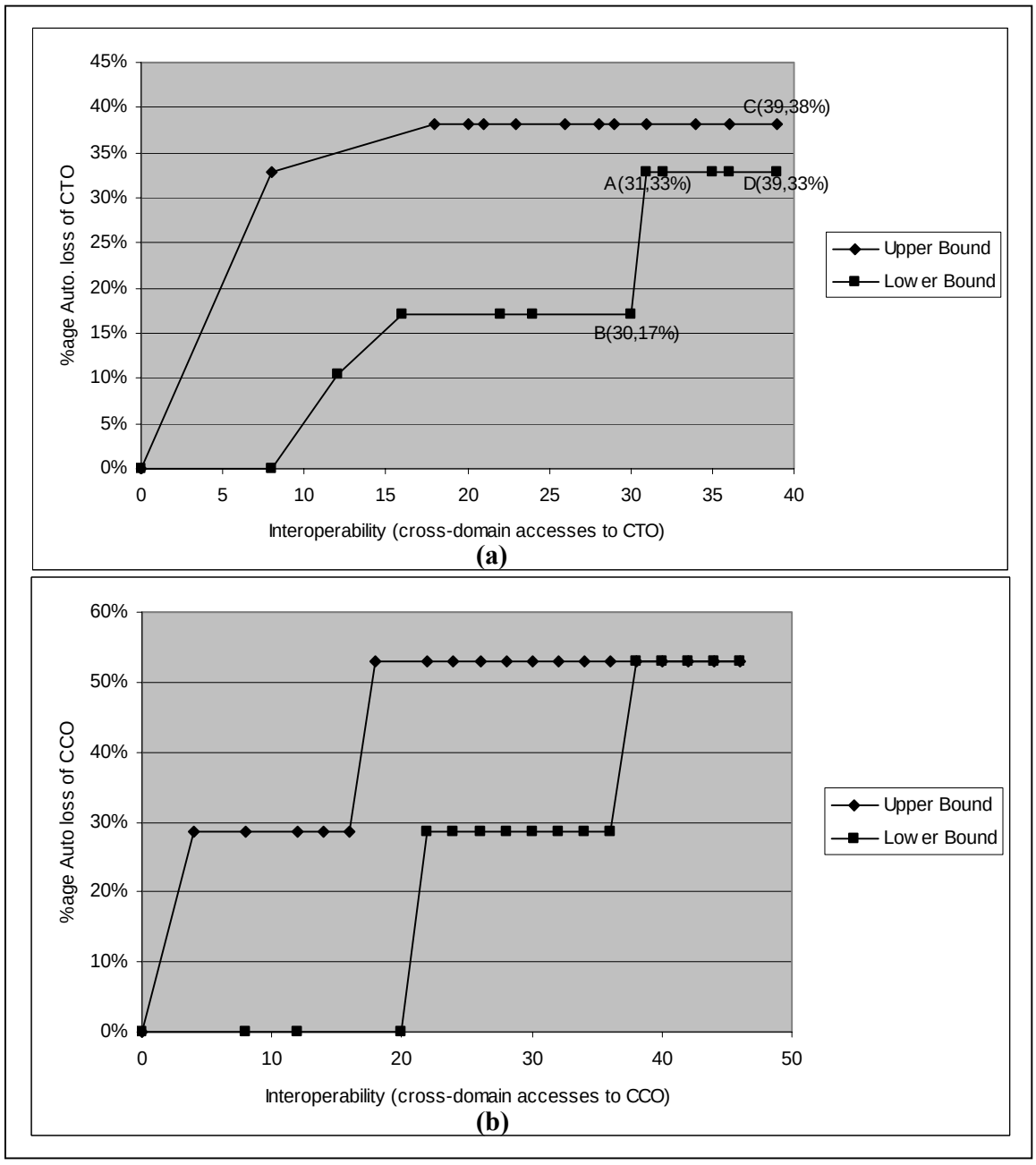

Fig. 4.6 Interoperability versus autonomy loss

4.5 Verification of Multi-domain policy

In this section, we formally analyze the proposed policy integration mechanism in the context of security constraints of collaborating domains. As mentioned in chapter 24, interoperation may results in three types of security constraint violations, including: *role-assignment constraint*, *role-specific SoD constraint,* and *user-specific SoD constraint*. However, the multi-domain policy produced by the proposed policy integration mechanism satisfies all the security constraints of all collaborating domains. Before proving the afore-mentioned claim, we first introduce some notations and definitions that help in defining security constraints in a formal manner.

Adjacency matrix: An adjacency matrix A_k represents the user-role graph of domain *k*. A_k defines role hierarchy and user to role assignment for domain *k*. $\dim(A_k) = \| U_k \| + | R_k \| \times \| U_k \| + | R_k \|$, where U_k is the set of user and R_k is the set of roles of domain *k*.

Closure matrix: A closure matrix A_k^+ is the transitive closure of the adjacency matrix A_k. $\dim(A_k^+) = \dim(A_k)$.

Projection operator: A projection operator π_{ur} takes an adjacency or closure matrix as input and returns a matrix with users along the rows and roles along the column. $\pi_{ur} : \{A_k, A_k^+\} \rightarrow U_k x R_k$, Projection of a closure matrix A_k^+ defines all possible user to role accesses in domain *k*.

$$\text{for any } a_{ij} \in \pi_{ur}(A_k^+), \qquad a_{ij} = \begin{cases} 0, & \text{if } u_i \text{ is not authorized to access } r_j \\ 1, & \text{otherwise} \end{cases}$$

Note that a_{ij}=1 does not imply that u_i is allowed to access role r_j. An *SoD* or cardinality constraint may prevent u_i from accessing r_j even though a_{ij}=1 in the projected closure matrix.

State matrix: A state matrix S is a matrix of dimension |U| x |R| ($U=\bigcup_k U_k$, $R=\bigcup_k R_k$) and it describes the user to roles accesses in the multi-domain environment. Note that the state matrix captures both intra-domain and inter-domain role accesses.

$$\text{for any } s_{ij} \in S, \qquad s_{ij} = \begin{cases} 1, & \text{role } r_j \text{ is being accessed by user } u_i \\ 0, & \text{otherwise} \end{cases}$$

State projection operator: There are two types of projection operator defined on state matrix, namely: per domain user-role projector (π_{ur_k}) and per domain role projector (π_{r_k}).

The operator π_{ur_k} takes any state matrix as input and projects the elements corresponding to the users and roles of domain *k*. It is defined as: $\pi_{ur\_k} : \mathrm{S} \rightarrow \mathrm{U}_k \times \mathrm{R}_k$, where $\mathrm{U_k} \subseteq \mathrm{U}$ and $\mathrm{R_k} \subseteq \mathrm{R}$.

The operator π_{r_k} takes any state matrix as input and projects the elements corresponding to the roles of domain *k*. It is defined as: $\pi_{ur\_k} : \mathrm{S} \rightarrow \mathrm{U} \times \mathrm{R}_k$.

R-SoD matrix: A R-SoD matrix, R^k_{SoD}, is a M × N matrix that defines role-specific *SoD* constraints of domain *k*. M is the number of roles in domain *k* (M = $|\mathrm{R}_k|$) and N is the number of role-specific *SoD* constraints defined in domain *k*. Note that a given domain can have multiple conflicting role sets R_{con}. Each column in the R-SoD matrix corresponds to one of the conflicting role sets R_{con}. Let λ^p be the p^{th} column of R^k_{SoD}, and R_{con_p} be the corresponding conflicting role set, then for each $r_j \in \mathrm{R}_k$.

$$\lambda^p(j) = \begin{cases} 1, \text{ if } r_j \in R_{con\_p} \\ 0, \text{ otherwise} \end{cases}$$

U-SoD matrix: A U-SoD matrix, $\mathrm{U}^k_{\mathrm{SoD}\_r}$, is a M × N matrix that defines user-specific *SoD* constraints for a role $r \in \mathrm{R_k}$. M is the number user-specific SoD constraints defined for the role *r*. N is the total number of users (N = |U|). Note that any role, *r*, in a given domain can have multiple sets of conflicting users (μ_r). The set U_{r_con} denotes the union of all the conflicting user sets of role *r*, i.e., $U_{r\_con} = \bigcup_{i=1}^{M} \mu_r^i$, where μ_r^i is the i^{th} conflicting user set of role *r*. Each row in the U-SoD matrix corresponds to one of the conflicting user sets μ_r. Let ρ^q be the q^{th} row of $\mathrm{U}^k_{\mathrm{SoD}\_r}$ and $\mu_r{}^q$ be the corresponding conflicting user set for role $r \in \mathrm{R_k}$, then for each $u_i \in \mathrm{U}$,

$$\rho_r^q(j) = \begin{cases} 1, \text{ if } u \in \mu_r^q \\ 0, \text{ otherwise} \end{cases}.$$

Definition 4.1: A state S is secure with respect to the role-assignment constraints of domain *k*, if:

$(s_{ij} = 1) \Rightarrow (a_{ij} = 1)$, where $s_{ij} \in \pi_{ur\_k}(S)$ and $a_{ij} \in A_k^+$. Alternatively, state S is secure if there does not exist any user $u_i \in U_k$ who accesses a role $r_j \in R_k$ in state S (s_{ij} = 1) and no intra-domain access path exists from u_i to r_j. Formally:

$$\text{for any } s_{ij} \in \pi_{ur\_k}(S),\ s_{ij} = 1 \Rightarrow \exists r_{j1},...,r_{jn} \in R_k \text{ such that}$$
$$u - assign(u_i, r_{j1}) \wedge \left[\begin{array}{l} (r_{j1} = r_j) \vee (r_{j1} \underset{I}{\geq} r_{j2} ... \underset{I}{\geq} r_{jn} \underset{I}{\geq} r_j) \vee (r_{j1} \underset{A}{\geq} r_{j2} ... \underset{A}{\geq} r_{jn} \underset{A}{\geq} r_j) \vee \\ (r_{j1} \underset{A}{\geq} r_{j2} .. \underset{A}{\geq} r_{jk} \underset{I}{\geq} r_{j(k+1)} ... \underset{I}{\geq} r_{jn} \underset{I}{\geq} r_j) \end{array} \right] \quad \text{(A)}$$

Proposition 4.2: A state S is secure with respect to the role-assignment constraints of domain *k* if and only if:

$\pi_{ur\_k}(S) \leq \pi_{ur}(A_k^+)$

i.e., $\forall s_{ij} \in \pi_{ur\_k}(S),\ s_{ij} \leq a_{ij},$ where $a_{ij} \in \pi_{ur}(A_k^+)$

Proof: $\Rightarrow$ In the transitive closure matrix A_k^+ of domain *k*, for a user u_i and role r_j, a_{ij} = 1 if and only if there is an intra-domain access path for u_i to r_j. Formally:

$$\text{for any } a_{ij} \in \pi_{ur}(A_k^+),\ a_{ij} = 1 \Leftrightarrow \exists r_{j1},...,r_{jn} \in R_k \text{ such that}$$
$$u - assign(u_i, r_{j1}) \wedge \left[\begin{array}{l} (r_{j1} = r_j) \vee (r_{j1} \underset{I}{\geq} r_{j2} ... \underset{I}{\geq} r_{jn} \underset{I}{\geq} r_j) \vee (r_{j1} \underset{A}{\geq} r_{j2} ... \underset{A}{\geq} r_{jn} \underset{A}{\geq} r_j) \vee \\ (r_{j1} \underset{A}{\geq} r_{j2} .. \underset{A}{\geq} r_{jk} \underset{I}{\geq} r_{j(k+1)} ... \underset{I}{\geq} r_{jn} \underset{I}{\geq} r_j) \end{array} \right] \quad \text{(B)}$$

(A) and (B) imply that $\pi_{ur\_k}(S) \leq \pi_{ur}(A_k^+)$.

$\Leftarrow$If for all $s_{ij} \in \pi_{ur\_k}(S),\ s_{ij} \leq a_{ij},$ where $a_{ij} \in \pi_{ur}(A_k^+)$, then condition (A) is satisfied and by Definition 4.1, S is secure with respect to the role-assignment constraints of domain *k*. ■

Definition 4.3: A state S is secure with respect to the role-specific SoD constraints of domain k, if no user $u_i \in U$ exists who accesses two or more roles in the conflicting role set $R_{con} = \{r_1,...,r_n \mid conf - role(r_i, r_j),$ where $1 \leq i, j \leq n$ and $i \neq j\}$. Formally:

$$\text{for all } u_i \in U,\ \sum_{r_j \in R_{con}} s_{ij} \leq 1 \quad \text{(C)}$$

R-SoD matrix: A R-SoD matrix, R^k_{SoD}, is a M × N matrix that defines role-specific *SoD* constraints of domain *k*. M is the number of roles in domain *k* (M = $|R_k|$) and N is the number of role-specific *SoD* constraints defined in domain *k*. Note that a given domain

can have multiple conflicting role sets R_{con}. Each column in the R-SoD matrix corresponds to one of the conflicting role sets R_{con}. Let λ^p be the p^{th} column of R^k_{SoD}, and R_{con_p} be the corresponding conflicting role set, then for each $r_j \in R_k$.

$$\lambda^p(j) = \begin{cases} 1, \text{ if } r_j \in R_{con\_p} \\ 0, \text{ otherwise} \end{cases}$$

Proposition 4.4: A state S is secure with respect to the role-specific *SoD* constraints of domain *k* if and only if:

$$\pi_{r\_k}(S) \times R_{SoD\_k} \leq \Theta_k \quad (D)$$

Where, Θ_k is a matrix of dimension $|U| \times |R_k|$ with all elements equal to one.

Proof: $\Rightarrow$ immediate from Definition 4.3 when applied to all conflicting role sets R_{con} of domain *k*.

$\Leftarrow$ Any user $u \in U$ in state S accesses at most one role from all the conflicting role sets of domain *k*. Hence, by Definition 4.3, S is secure. ■

Definition 4.5: A state S is secure with respect to the user-specific SoD constraints of domain *k*, if for each role $r \in R_k$ which have a non-empty set (U_{r_con}) of conflicting user sets, at most one user from each of the conflicting user sets ($\mu_r \in U_{r\_con}$) accesses role *r* in sate S. Formally:

$$\forall r_j \in R_k \left(\forall \mu \in U_{r_j\_con}, \sum_{u_i \in \mu} s_{ij} \leq 1 \right) \quad (E)$$

U-SoD matrix: A U-SoD matrix, $U^k_{SoD\_r}$, is a M × N matrix that defines user-specific *SoD* constraints for a role $r \in R_k$. M is the number user-specific SoD constraints defined for the role *r*. N is the total number of users (N = |U|). Note that any role, *r*, in a given domain can have multiple sets of conflicting users (μ_r). The set U_{r_con} denotes the union of all the conflicting user sets of role *r*, i.e., $U_{r\_con} = \bigcup_{i=1}^{M} \mu_r^i$, where μ_r^i is the i^{th} conflicting user set of role *r*. Each row in the U-SoD matrix corresponds to one of the conflicting user sets μ_r. Let ρ^q be the q^{th} row of $U^k_{SoD\_r}$ and $\mu_r{}^q$ be the corresponding conflicting user set for role $r \in R_k$, then for each $u_i \in U$,

$$\rho_r^q(j) = \begin{cases} 1, \text{ if } u \in \mu_r^q \\ 0, \text{ otherwise} \end{cases}.$$

Proposition 4.6: A state S is secure with respect to the *user-specific SoD* constraints of domain k, if and only if for all roles $r_j \in R_k$ which have a non-empty set of conflicting user sets $(U_{r_j\_con})$, the following condition holds:

$$U^k_{SoD\_r_j} \times s_j \leq \theta_{r_j} \quad \text{(F)}$$

Where, s_j is the j^{th} column of the state matrix S and θ_{r_j} is a vector with all elements equal to one. The length of θ_{r_j} is equal to the number of *user-specific SoD* constraints defined for role r_j.

Proof: $\Rightarrow$ immediate from Definition 4.5.

$\Leftarrow$ For all roles r_j with at least one conflicting set of users, $U^k_{SoD\_r_j} \times s_j \leq \theta_{r_j}$ implies that at most one user from each of the conflicting set of users for role r_j accesses role r_j in state S. Hence, by *definition 6.5*, S is secure with respect to the *user-specific SoD* constraints of domain *k*. ■

Having described the formal specification and conditions for the satisfaction of security constraints, we now provide a formal proof that the multi-domain policy generated by the policy integration mechanism, satisfies the security requirements of all collaborating domains.

Theorem 4.7: Let K_1, …..,K_n, n ≥ 2, be collaborating domains such that the security policy of each K_i be consistent. Let **G** be the multi-domain RBAC graph obtained from K_1, …..,K_n by applying the conflict resolution algorithm *ConfRes*. Any state S reachable from **G** is secure with respect to the *role-assignment, role-specific SoD,* and *user-specific SoD* constraints of all collaborating domains.

Proof of Theorem 4.7 is given in Appendix

5 CONCLUSION

5.1 Summary of Current Work

Our current research focuses on the problem of integrating the access control policies of heterogeneous and autonomous domains to allow inter-domain information and resource sharing in a secure manner. The policy integration mechanism, discussed in this proposal, is a two step process including composition of a global multi-domain policy from the access control policies of collaborating domains and removing conflicts from the global policy in an optimal manner without compromising the security of constituent domains. Another key requirement of policy integration is to maintain the autonomy of all collaborating domains. There is a trade-off between seeking interoperability and preserving autonomy. Violation of a collaborating domain's security policy in general is not permissible. However, some domains may tolerate a compromise in their autonomy for establishing more interoperability. We have formulated the problem of secure interoperation as an optimization problem with an objective of maximizing interoperability with minimum autonomy losses and without causing any security violation of collaborating domains. The multi-domain policy obtained from the proposed policy integration framework is conflict-free and satisfies the security requirements of the collaborating domains. However, the resulting policy may not yield the desired autonomy level. Various heuristics can be used to obtain a sub-optimal solution from the given optimal solution to attain the desired autonomy level. One such heuristic is to try all possible combinations of cross-domain links obtained from the optimal solution and then selecting a solution which meets the desired autonomy loss with maximum interoperability. The multi-domain policy produced by this heuristic is also secure and conflict free.

5.2 Future Work

Following are four research problems related to policy management that we intend to address during the course of this research.

1. Verification of RBAC policy specification using state-space analysis techniques.
2. Reconfiguration of interoperation policy because of changes in domains' access control policies.
3. Evaluation of domains' autonomy in the collaborative environment.
4. Semantic partitioning of a single access control policy into multiple independent and autonomous policies. Partitioning of a policy is required when an organization or a business alliance breaks up into multiple organizations or alliances

A description of these problems and a possible strategy is given below.

5.2.1 Verification of RBAC policy specification

The consistency of the multi-domain policy generated by the proposed policy integration framework depends on the consistency of the access control policies of the collaborating domains. If the access control policy of any one of the collaborating domain is conflicting then the resulting multi-domain policy will be inconsistent. Therefore, the access control policies of domains need to be verified before interoperation is established. The verification of security policies of individual domains must precede the policy integration step.

Security policy verification in general is an undecidable problem [Har76]. However, much work has been done to determine reasonable models and limitations under which safety is decidable and tractable [Amm91, Amm92, Amm94, Sny97, Jae01 Koc02]. Verification of a domain's access control policy entails various challenges, including: i) specifying policy using a formal model, ii) identifying the safety requirements, and iii) determining if a given policy yields unauthorized accesses. The policy specification model should be generic and flexible enough to express a wide range of security and access control policies. Generally the safety requirements are specified in the form of security constraints. The security constraints can be a part of the policy specification

model or can be expressed separately. In both cases the positive authorizations implied by the model and the negative authorizations defined by the constraints may conflict, making the policy inconsistent.

In some application domains, it may not be possible to transform all the safety requirements into formal constraints which can derive the underlying access control policy specification. For instance, in state-event based applications with huge state space, it may not be possible to define security constraints on all the states that may lead to insecure or unsafe states. In this case state space partitioning [Jae03] can be used to check the consistency of an access control policy. Accordingly, the entire state space can be partitioned into the following two state spaces: *prohibited space* consisting of authorizations precluded in the safety policy specification; and *specified space* consists of authorizations/permissions implied by the access control policy under current configuration. However, finding the prohibited space and specified space is not an easy task. In order to explore state spaces a state generating machine is needed that can be driven by the underlying policies under any arbitrary configuration. Petri-nets and its variants have been widely used as a specification and modeling tool for most event driven applications, including: multi-media documents, workflow applications, and business transaction procession systems [Lit93, Atl97, Atl98]. The structural properties of Petri-nets together with the use of predicates can be used to model a wide range of constraints in access control, including: hierarchy, SoD, and cardinality constraints. Moreover, the state-event semantics of the Petri-nets can be exploited to capture the event-based constraints of access control policies that cannot be modeled by simple graph-based models. In our earlier work on interoperation in a multi-domain environment, we concentrated on the role assignment, SoD and cardinality constraints and did not extensively address the event-based constraints. To some extent the activation hierarchy captures the event based semantics of RBAC, however, it does not fully characterize the wide range of dynamic constraints needed in trigger based systems such as active databases and workflow systems. We plan to combine the event-based approach taken in GTRBAC [Jos03] with the Petri-net based model to develop a framework for modeling and analysis of non-temporal RBAC policies. This will allow us to perform state-based

analysis for policy verification and also facilitate in developing an event-based execution model of an RBAC system in order to ensure safety. Furthermore, several formal tools and techniques are available for Petri-nets that can be used to carry out relevant analysis for correctness verification.

Deciding the correctness of an access control policy is one aspect of the verification problem. In some cases, it may not be possible completely eliminate conflicts from a given policy. Therefore, some conflict resolution strategy is needed to resolve conflicting authorizations from a given policy. Policy conflicts can be broadly classified into two classes: i) conflicts that are independent of the system state or configuration and can be captured in the policy specification, ii) conflicts that depend on the state or configuration of the system. State independent conflicts can be identified using offline analysis techniques and can be removed by modifying the policy specification. There may be several policy readjustment options available to resolve a given conflict, and each option may yield a different set of constraints and accesses. However, one would desire an option that resolves the conflicts in an optimal manner. There can be several optimality measures such as maximizing accessibility, minimizing new constraint additions, and maximizing active constraint set. We believe that the Integer Programming based approach discussed in the context of policy integration can be used to resolve state-independent conflicts present in a domain's security policy in an optimal manner. However, the underlying treatment of constraints would be different. In the multi-domain policy integration problem, the security of collaborating domains is given utmost importance in establishing interoperation. However, in a single domain, policy conflicts are caused by the security constraints. Conflict resolution in this case involves dropping some security constraints. Question is how much compromise in security is acceptable? We believe that the policy designing framework must have the capability to infer relative importance of the security constraints specified in the given policy. In the context of RBAC, such inference can be made by analyzing the role attributes, permissions assignment, and credentials of users assigned to the roles. The policy designers can also provide their input by prioritizing different security constraints.

For the state dependent conflicts, several dynamic conflict resolution policies can be defined depending on the domain [Ber03, Fer00]. Examples of dynamic conflict resolution policies are denials take precedence, most specific authorization take precedence, and permission takes precedence. Note that dynamic conflict resolution does not make the underlying security policy consistent. It only provides an exceptional handling mechanism, which may also lead to an inconsistent state. This again motivates for the state-space based policy verification approach discussed above.

5.2.2 Policy Evolution

In a multi-domain collaborative environment, the local access control policy of the collaborating domains may evolve with time. The security policy of the multi-domain system itself may change. System administer(s) responsible for the global interoperation policy may define new rules or constraints for cross-domain accesses. Consequently, the interoperation policy needs to be redefined to incorporate the new security and access control requirements. Defining a new interoperation policy by reintegrating the access control policies is a time consuming process and may not be viable in environments where domain policies change frequently. Therefore, a policy adjustment mechanism is needed that upon sensing any policy changes, reconfigures the existing interoperation policy in a timely manner. The readjusted policy may not yield an optimal level of interoperation, but it must preserve the security and autonomy of collaborating domains.

5.2.3 Autonomy and interoperability trade-off

We plan to investigate the relationship between interoperability and autonomy in a distributed collaborative environment. The autonomy and interoperability trade-off discussed in Chapter 4 of this proposal, is based on the worst-case analysis in which it is assumed that a user by assuming a local roles also acquires the permissions of related cross-domain roles. This is a very conservative assumption and may unnecessarily restrict interoperability because of the possibility of autonomy losses. Moreover, in presence of temporal and event-based constraints, assumption of a local role may not enable acquisition of cross-domain permissions all the time. The event-based constraints may

even restrict the time during which the local roles can be acquired. This motivates for reassessing the affect of interoperability on domains' autonomy and vice versa. We believe that stochastic estimates will help in determining the access patterns of users for both cross-domain and local accesses. This probabilistic analysis can then be used in determining the autonomy losses at a given interoperability level.

5.2.4 Policy partitioning for enterprise splitting

Our major focus until now has been on the integration of access control policies for facilitating interoperation and business collaboration. However, in a ever-changing business world, collaborations and business alliances keep evolving, big companies get split, merge and sometimes displaced by entirely new companies. Splitting of companies is not a new phenomenon. Giant companies sometimes split into multiple independent units for various reasons. In the event of an organization split-up, the information infrastructure owned by the parent organization is also divided among the newly formed organizations. Consequently, policies governing access to the inherited information resources need to be defined for the new setup. The organizational hierarchy of the newly formed organizations may not differ drastically from the organizational hierarchy of the parent organization. This implies that the access control policy of the parent organization can be used to derive the policies of new organizations. Therefore, a policy generation framework is needed that can compose access control policies for organizational units formed as a result of a company split-up. Input provided to this framework may consist of the access control policy of the parent organization, scope and business requirement, potential organizational hierarchy, and a list of information resources and assets inherited by the new organizational unit. In an abstract sense, this problem can be considered as a partitioning of a policy based on the scope and business requirement of new organization. Since we have considered a graph based formalism for access control policy specification, we plan to explore different graph transformation and partitioning techniques to solve the policy partitioning problem.

6 REFERENCES

[Ahn00] G. Ahn, R. Sandhu, "Role-Based Authorization Constraints Specification," *ACM Transactions on Information and System Security*, 3(4), November 2000.

[Amm91] P. Amman and R. Sandhu, "Safety Analysis for the Extended Schematic Protection Model," *in proc. of the IEEE Symposium on Research in Security and Privacy*, 1991

[Amm92] P. Amman and R. Sandhu, "The Extended schematic Protection Model," *J. Computer Security,* 1992

[Amm94] P. Amman and R. Sandhu, "One-Representative safety Analysis in the Non-Monotonic Transform Model," *in proc. of the 7th IEEE Computer security Foundations Workshop*, pp. 138 – 149, 1994.

[Atl97] V. Atluri and W-K. Huang, "An Extended Petri Net Model for Supporting Workflow in a Multilevel Secure Environment," In *Proceedings of the Tenth Annual IFIP TC11/WG11.3 International Conference on Database* pp.240-258, January 1997, Como, Italy.

[Atl98] V. Atluri, W-K. Huang and E. Bertino, ``A Semantic Based Redesigning of Distributed Workflows," *9th International Conference on Management of Data,* December 1998.

[Bat86] C. Batini, M. Lenzerini, and S. Navathe, "A Comparative Analysis of Methodologies for Database Schema Integration," *ACM Computing Survey*, Vol. 18, No. 4, pp. 323 – 364, December 1986.

[Ber99] E. Bertino, E. Ferrari, and V. Atluri, "The Specification and Enforcement of Authorization Constraints in Workflow Management Systems," *ACM Transactions on Information and System Security*, 2(1):65-104, 1999.

[Ber99b] E. Bertino, E. Ferrari, V. Atluri, "The Specification and Enforcement of Authorization Constraints in Workflow Management Systems," *ACM Transactions on Information and System Security*, 2(1), February 1999, pages 65-104.

[Ber03] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca, "A Logical Framwork for Reasoning About Access Control Models," *ACM TISSEC*, Vol. 6, No. 1, pp. 71 – 127, February 2003.

[Bel73] D. Bell and L. Lapadula, "Secure Computer Systems: Mathematical Foundations," Technical Report MTR-2547, Vol. 1, MITRE Corporation, March 1973.

[Bew89] D. F.C. Bewer, M. J. Nash, "The Chinese Wall Security Policy," *In Proceedings of the Symposium on Security and Privacy*, IEEE Computer Society, May 1989, pages 206-214.

[Bib77] K. Biba, "Integrity Considerations for Secure Computer Systems," Technical Report MTR-3153, Vol. 1, MITRE Corporation, April 1977.

[Bon96] P.A. Bonatti, M. L. Sapino, V.S. Subrahmanian, "Merging Heterogeneous Security Orderings," *ESORICS 1996*, pp. 183-197

[Coh02] E. Cohen, R. K. Thomas, W. Winsborough, and D. Shands "Models for Coalition-based Access Control," *Seventh ACM Symposium on Access Control Models and Technologies*, pp. 97 – 106, June 2002.

[Elm01] A. K. Elmagarmid and W. J. Mciver Jr., "The Ongoing March Toward Digital Government," *IEEE Computer*, Vol. 34, No. 2, pp. 32 – 38, February 2001.

[Fer93] D. F. Ferraiolo, D. M. Gilbert, N. Lynch, "An Examination of Federal and Commercial Access Control Policy Needs," In *Proceedings of NISTNCSC National Computer Security Conference*, Baltimore, MD, September 20-23, 1993, pages 107-116.

[Fer00] E. Ferrari and B. Thuraisingham, "Secure Database System," In *Advanced Databases: Technology and Design*, O. Diaz and M. Piattini, Eds, Artech House, London.

[Fer01] D. Ferraiolo, R. Sandhu, S. Gavrila, R. Kuhn, R. Chandramouli, "The NIST Model for Role-Based Access Control: Towards a Unified Standard," *ACM Transactions on Information and System Security*, Vol. 4, Issue 3, August 2001, pp. 224-274.

[Gav98] S. I. Gavrila , J. F. Barkley, "Formal Specification for Role Based Access Control User/role and Role/role Relationship Management," *Proceedings of the*

third ACM workshop on Role-based access control, Fairfax, Virginia, United States, pp. 81-90, October 1998.

[Giu95] L. Giuri, "A New Model for Role-based Access Control," In *Proceedings of 11th Annual Computer Security Application Conference*, New Orleans, LA, December 11-15 1995, pages 249-255.

[Giu97] L. Giuri. Role-based Access Control: A natural approach. In *Proceedings of the 1st ACM Workshop on Role-Based Access Control*. ACM, 1997.

[Gon96] L. Gong and X. Qian, "Computational Issues in Secure Interoperation", *IEEE Transaction on Software and Engineering*, Vol. 22, No. 1, January 1996.

[Gua02] G. Yan, W. K. Ng, E. Lim, "Product Schema Integration for Electronic Commerce - A Synonym Comparison Approach," *IEEE TKDE* Vol. 14, No. 3 pp. 583-598, June 2002.

[Har76] M. Harrisson, W. Ruzzo, and J. Ullman, "Protection in Operating Systems," *Communications of the ACM*, Vol. 19, No. 2, August 1976, pp. 461-471.

[Hos91] H. Hosmer, "Metapolicies I," *ACM SIGSAC Review*, 1992, pp. 18-43.

[IJIS] "Integrated Justice Information System," The Department of Justice Initiative, available at http://www.ojp.usdoj.gov.

[Jae01] T. Jaegar and J. Tidswell, "Practical Safety in Flexible Access Control Models," *ACM TISSEC*, Vol. 4 No. 2, pp. 158 – 190, May 2001.

[Jae03] T. Jaegar and X. Zhang, "Policy Management Using Access Control Spaces," *ACM TISSEC*, Vol. 6 No. 3, pp. 327 – 364, August 2003.

[Jos01a] J. B. D. Joshi, W. G. Aref, A. Ghafoor and E. H. Spafford, "Security Models for Web-based Applications," *Communications of the ACM*, Vol. 44, No. 2, Feb. 2001, pages 38-72.

[Jos01b] J. B. D. Joshi, A. Ghafoor, W. Aref, E. H. Spafford, "Digital Government Security Infrastructure Design Challenges", *IEEE Computer*, Vol. 34, No. 2, February 2001, pages 66-72.

[Jos02] J. B. D. Joshi, E. Bertino, A. Ghafoor, "Temporal Hierarchies and Inheritance Semantics for GTRBAC," *Seventh ACM Symposium on Access Control Models and Technologies*, pp. 74-83, June 2002.

[Jos03] J. B. D. Joshi, "A Generalized Temporal Role Based Access Control Model for Developing Secure Systems," Ph.D. Thesis, School of Electrical and Computer Engineering, Purdue University, 2003.

[Ker02] A. Kern, "Advanced Features for Enterprise-Wide Role-Based Access Control," *Annual Computer Security Applications Conference*, 2002

[Koc02] M. Koch, L.V. Mancini and F. P. Presicce, "A Graph-Based Formalism for RBAC," *ACM Transactions on Information and System Security*, Vol. 5, No. 3, pp. 332-365, August 2002.

[Kun99] D. R. Kuhn, "Mutual Exclusion of Roles as a Means of Implementing Separation of Duties in a Role-based Access Control System," *ACM Transactions on Information and System Security*, 2(2), 1999, pages 177-228.

[Li94] W. S. Li and C. Clifton, "Semantic Integration in Heterogeneous Databases Using Neural Networks," *VLDB* 1994.

[Lit93] T. D. C. Little and A. Ghafoor, "Interval-Based Conceptual Models for Time-Dependent Multimedia Data," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 5, No. 4 , pp. 551 -563, August 1993.

[Osb00] S. L. Osborn, R. Sandhu, Q. Munawer, "Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies," *ACM Transactions on Information and System Security*, Vol. 3, No. 2, February 2000, pp. 85-106.

[Osb02] S. L. Osborn, "Integrating Role Graphs: A Tool for Security Integration," Data and Knowledge Engineering, Vol. 43 No. 3, pp. 317-333, 2002.

[Pot03] R. Pottinger and P. A. Bernstein, "Merging Models Based on Given Correspondences," VLDB 2003, pp. 826-873.

[Pow00] R. Power, ""Tangled Web": Tales of Digital Crime from the Shadows of Cyberspace," *Que/Macmillan Publishing*, Aug. 31, 2000.

[Nya93] M. Nyanchama, S. L. Osborn, "Role-Based Security, Object-Oriented Databases and Separation of Duty", *SIGMOD Rec*. 22, 4, December 1993, pages 45-51.

[Nya99] M. Nyanchama and S. Osborn. The Role Graph Model and Conflict of Interest. *ACM Transactions on Information and System Security*, 2(1), 1999, pages 3-33.

[San91] R. Sandhu, "Separation of Duties in Computerized Information Systems", In *Database Security IV: Status and Prospects*. Elsevier North-Holland, Inc., New York, 1991, pages 179-189.

[San95] R. Sandhu, editor. *Proc. of the First ACM Workshop on Role-Based Access Control*, Fairfax (VA), 1995.

[San96] R. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman, "Role-Based Access Control Models," *IEEE Computer* 29(2), IEEE Press, 1996, pages 38-47.

[San97] R. Sandhu, editor. *Proc. of the 2nd ACM Workshop on Role-Based Access Control*, Fairfax (VA), 1997.

[San98a] R. Sandhu editor. *Proc. of the 3rd ACM Workshop on Role-Based Access Control*, Fairfax (VA), 1998.

[San98b] R. Sandhu, "Role-based Access Control," *Advances in Computers*, vol. 46, Academic Press, 1998.

[San98c] R. Sandhu, "Role Activation Hierarchies," In *Proceedings of the third ACM workshop on Role-based access control*, pp.33-40, October 22-23, 1998.

[She90] A. P. Sheth and J. A. Larson, "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases," *ACM Computing Survey*, Vol. 22, No. 3, pp. 183 – 236, September1990.

[Sim97] R. Simon, M.E. Zurko, "Separation of Duty in Role-based Environments," In *Proc. 10th IEEE Computer Security Foundations Workshop*, June 1997.

[Sny97] L. Snyder, "On the Synthesis and Analysis of Protection Systems," In *Proceedings of the 6th ACM Symposium on Operating System Principles*, pp. 141 – 150, 1997

[Tar97] Z. Tari, S. Chan, "A Role-Based Access Control for Intranet Security," *IEEE, Internet Computing*, Sept-Oct, 1997, pages 24-34.

[Tid98] J. Tidswell, J. Potter. A Dynamically Typed Access Control Model. I*n Proceedings of the Third Australasian Conference on Information Security and Privacy*, July 1998.

[Vet98] V. Vet and N. Mars "Bottom-Up Construction of Ontologies," *IEEE TKDE*, Vol. 10, No. 4, pp. 513-526, August 1998.

[Wol98] L. A. Wolsey, *Integer Programming*, John Wiley, New York, 1998.

7 APPENDIX

Proof of Lemma 3.1: The split function, given in Figure 3.5, first creates a new role r_j and makes it junior to r_s. Note that until line number 2 of the split function, role r before splitting and r_s have same directly assigned permissions and all the roles that are related to r are also related to r_s in the same manner.

Lines 3 - 4 in the split function algorithm make sure that all the permissions that are removed from r_s are assigned to r_j. Since $r_s \underset{I}{\geq} r_j$, therefore these permissions are still included in the permission set of r_s, i.e., $pset(r_s) \supseteq pset_{assign}(r_j)$.

Lines 6 -8 ensures that the inheritance relationship is maintained between r_s and all the roles that were junior to the unsplit role in the I-hierarchy semantics. Since $pset_{assign}(r) = pset_{assign}(r_s) \cup pset_{assign}(r_j)$ and all the roles that can be reached from the unsplit role r through an I-path can also be reached from r_s through an I-path; therefore, $pset(r) = pset(r_s)$

It can be noted that splitting a role does not change the *activation hierarchy* and the *user to role assignment*. That is, all the users that were assigned to unsplit role r remain assigned to role r_s and all the roles that are related to r by an A-edge are also related to r_s by an A-edge. This implies that the uniquely activable set of role r_s is same as that of the unsplit role r. ■

Proof of Lemma 3.3: The algorithm remove-role ensures that the inheritance relationship between all the roles r_p such that $r_p \underset{I}{\geq} r_d$ and all roles r_c such that $r_d \underset{I}{\geq} r_c$ is maintained, that is, $r_p \underset{I}{\geq} r_c$ holds after role r_d is removed. Since r_d is a redundant role, no user is assigned to r_d nor is any permission assigned to it. Hence, the user set and the

permission set is unaffected by the removal of the redundant role r_d. Since all the user-to-role assignment relations, role-to-permission-assignment relations and hierarchy relations among roles other than r_d are preserved, properties 1, 3, and 4 hold. Moreover, the algorithm remove-role does not remove any role other than r_d from the conflicting role set of any role, implying that 3 and 5 hold. ■

Proof of Lemma 3.4

PIR 1 *Element preservation*: RBAC-integrate does not remove any element except the newly created redundant roles. Since these roles are not a part of any of the input RBAC graphs, RBAC-integrate satisfies element preservation requirement.

PIR 2 *Relationship preservation*: In RBAC-Integrate, relationship between the elements of input RBAC graph is altered when a newly created redundant role is removed or when a role is split. Lemma 3.3 states that removing a newly created redundant role does not change the relationship that exists between the elements of input RBAC graphs. When a role is split some of the relations involving the split roles are removed and some new relations are added. This modification may alter some of the explicit relationships specified in the input RBAC graphs, however, the original relations are implied in the final graph G as stated in Lemma 3.1.

PIR 3 *User authorization preservation*: In RBAC-integrate no user to role assignment is removed and all the hierarchical relationship between roles is maintained (PIR 2). Furthermore, equivalent roles have same permission assignment and inheritance. Therefore, the permission authorization set of users is preserved by RBAC-integrate.

PIR 4 *Minimum overhead*: The algorithm RBAC-integrate may create new roles during the process of policy integration. These roles are not present in the original RBAC policies of component domains and are created as a result of role splitting. These additional roles are considered as an overhead associated with the integration process. It is therefore important to minimize any additional number of roles created during the integration step. However, some of these newly created roles are essential for allowing cross-domain accesses in a secure manner. Some of newly created roles are redundant and do not have any permissions associated with them. Lemma 3.3 states that the removal

of redundant roles does not affect any intra and inter-domain accesses in the multi-domain environment, hence all the newly created redundant roles are removed by RBAC-integrate.

Splitting roles unnecessarily may also introduce considerable overhead. However, RBAC-integrate maintains the minimal splitting property defined below. This property ensures that cross-domain equivalent roles are not split unless there is a third role from some other collaborating domain, which has some permission(s) in common with the permissions associated with the cross-domain equivalent roles.

Minimal splitting: When integrating RBAC policies of two domains A and B, no roles $r_i \in$ A and $r_j \in$ B exist for which all the following conditions hold:

(a) $eq\_role(r_i, r_j)$

(b) $\exists r_{is}, r_{js} : (r_{is} \in A \wedge r_{js} \in B \wedge r_{is} \geq_I r_i \wedge r_{js} \geq_I r_j \wedge eq\_role(r_{is}, r_{js}))\ eq\_role(r_i, r_j)$

(c) r_i, r_j, r_{iS}, and r_{jS} are not present in the original RBAC policies of A and B and are created during the process of integration.

We assume that duplicate permission assignment to two or more roles belonging to the same domain is not permitted in the input RBAC graph, *that is*. two or more roles belonging to same domain cannot have same permissions assigned to them. Also, in RBAC-integrate two cross-domain roles are compared only once. Consequently, when integrating the RBAC graphs of two domains, say A and B, the permission set of any newly created role r_A in A does not include any permissions assigned to any role *r'* in B except for one role r_B in B for which *eq_role*(r_A, r_B) is true. This means that during the process of integrating the RBAC graphs of domains A and B, a newly created role cannot split.

RBAC-integrate maintains non-redundancy (all redundant roles are removed from the integrated policy) and minimal splitting. Removing any role from a multi-domain RBAC graph that maintains minimal splitting property may either violate the element preservation property (PIR 1) or reduces the level of interoperation. This implies that the multi-domain RBAC policy output by RBAC-integrate has minimum overhead. ■

Proof of Theorem 3.5: We first show that the morphism 'φ' is onto. (i.e., for all y ∈ Y, there exists x ∈ X such that φ(x) = y).

φ is onto: The elements in X and Y can be divided into two types: (i) elements which are present in G_A, G_B, and G_C, (ii) elements that are created in the process of integration of local graphs. As stated in the above theorem that RBAC-integrate satisfies the element preservation property, therefore all the elements of type (i) are present in both X and Y.

Type (ii) elements include those roles that are not present in G_A, G_B, and G_C and are created during the process of policy integration. These roles are created by the role split function in the RBAC-integrate algorithm. Note that type (ii) elements do not include any redundant role as the redundant roles that are created in the policy integration step are eliminated from X and Y. To complete the proof that φ is onto, we need to show that for all type (ii) roles $r \in Y$, there exists $r' \in X$ such that $\varphi(r') = r$ and for all p such that $p \in pset_{assign}(r) \Rightarrow p \in pset_{assign}(r')$

In the following we use the terminology $r \in dom(X)$ if $r \in G_X$ or r is created by splitting a role $r_s \in dom(X)$. Without loss of generality, assume that there exists a role $r_A \in G_A$ such that $pset(r_A) \supseteq pset(r)$. Also r is created by splitting role r_A i.e., $r \in dom(A)$. Since r is created in the process of integration, therefore one of the following three conditions holds for r.

(a) $\exists\, r_{BA} \in dom(B)$: $eq\_role(r,r_{BA}) \wedge \neg\exists\, r_{CA} \in dom(C)$: $eq\_role(r,r_{CA})$

(b) $\exists\, r_{CA} \in dom(C)$: $eq\_role(r,r_{CA}) \wedge \neg\exists\, r_{BA} \in dom(B)$: eq_role$(r,r_{BA})$

(c) $\exists\, r_{BA} \in dom(B), r_{CA} \in dom(C)$: $eq\_role(r,r_{BA}) \wedge eq\_role(r,r_{CA})$

Case a: $\exists\, r_{BA} \in dom(B)$: $eq\_role(r,r_{BA}) \wedge \neg\exists\, r_{CA} \in G_C$: $eq\_role(r,r_{CA})$

The above implies that there is no role in G_C whose permission set overlaps with that of r or r_{BA}. Role r does not exist in Q; however, r_{BA} may or may not exist in Q.

If r_{BA} exists in Q then $r_{BA} \in G_B$ and the following is true in Y:

(i) $(r_A \underset{I}{\geq} r) \wedge (r_A \; contains \; r_{BA}) \wedge (\neg r_{BA} \; contains \; r_A)$

If r_{BA} does not exists in Q, then there exists a role $r_B \in G_B$ such that that $pset(r_B) \cap pset(r_A) = pset(r_{BA})$, and the following hold in Y:

(ii) $(r_A \underset{I}{\geq} r) \wedge (r_A \textit{ overlaps } r_B)$

Since $eq\_role(r, r_{BA})$ holds, therefore $pset_{assign}(r_{BA}) = pset_{assign}(r)$ and $pset(r_{BA}) = pset(r)$

For the case $r_{BA} \in G_B$ and $r_A \in G_A$, since r_A *contains* r_{BA}, when integrating G_A and G_B, a role *r'* junior to r_A is created and is assigned the permission in the set $pset_{assign}(r_{BA}) \cap pset_{assign}(r_A)$. This means that there exists a role r' in P with $pset_{assign}(r') = pset_{assign}(r_{BA}) \cap pset_{assign}(r_A) = pset_{assign}(r_{BA}) = pset_{assign}(r)$. Also, when integrating P with G_C role r' is not split nor the permission in the set $pset_{assign}(r')$ gets redistributed as there is no role in G_C whose permission set overlaps with that of *r'*.

For the case $r_{BA} \notin G_B$, $r_A \in G_A$ and $r_B \in G_B$, since r_A *overlaps* r_B, when integrating G_A and G_B, role *r'* junior to r_A, and r_{BA} junior to r_B are created with $pset_{assign}(r') = pset_{assign}(r_B) = pset_{assign}(r_{BA}) \cap pset_{assign}(r_A)$. This means that there exists a role *r'* in P with $pset_{assign}(r') = pset_{assign}(r_{BA}) = pset_{assign}(r)$. Also, when integrating P with G_C role *r'* is not split nor the permission in the set $pset_{assign}(r')$ gets redistributed as there is no role in G_C whose permission set overlaps with that of *r'*.

Therefore for a type (ii) role $r \in Y$, for which case a holds, there exists a role $r' \in X$ such that $pset_{assign}(r') = pset_{assign}(\underline{r})$, i.e., $\varphi(r') = r$. In a similar manner, we can prove the above for case b and c as well. Hence, for all type (ii) roles $r \in Y$, there exists a role $r' \in X$ such that $pset_{assign}(r') = pset_{assign}(r)$, i.e., $\varphi(r') = r$.

Now, we need to show that for all roles $r \in Y$, there exists a role $r' \in X$ such that $pset_{assign}(r') = pset_{assign}(r)$. We have proved this for type (ii) roles, now we need to prove it for type (i) roles. Type (i) role can be further classified into two types: (a) roles which remain unsplit during policy integration; (b) roles which split in the policy integration step. Note that the permissions assigned to a role are removed from that role only if it gets split in the process of integration. Consider an unsplit role *r* in Y and with out loss of generality assume that $r \in G_A$. Since *r* is an unsplit role therefore, there does not exist any role $r'' \in \{G_B, G_C\}$ such that $pset_{assign}(r) \subset pset_{assign}(r'')$. This and the element preservation property implies that there exists a role r' $\in$ X, such that $pset_{assign}(r') = pset_{assign}(r)$.

We need to prove the above for the type (i) roles that get split. Consider a role $r \in Y$ that got split in the process of policy integration to produce a junior role r_j. We already proved that there exists a role $r_j' \in X$ such that $pset_{assign}(r_j') = pset_{assign}(r_j)$.Without loss of generality suppose that $r \in G_A$. Note that $r_j \notin \{G_A, G_B, G_C\}$, which also implies that $r_j' \notin \{G_A, G_B, G_C\}$. Therefore there exists a role r' that produce r_j' after splitting. We maintain that $pset_{assign}(r') = pset_{assign}(r)$. Suppose this is not the case and $pset_{assign}(r') \neq pset_{assign}(r)$. Both r_j and $r_j' \in dom(A)$, which implies that $r' \in dom(A)$. Suppose that $pset_{assign}(r') \supset pset_{assign}(r)$. Note that permissions are removed from a role only if the role gets split and the removed permissions are assigned to the newly created role that is made junior to the role being split. Before splitting, r' and r have same permission assignment. However, after splitting we assume that $pset_{assign}(r') \supseteq pset_{assign}(r)$, implying that either $pset_{assign}(r_j') \subset pset_{assign}(r_j)$ which is not possible, or r has at least one more newly created junior role r_{j2} which acquires some of the permissions that were earlier assigned to r. If this is the case then r_{j2} must be equivalent to some role $r_{j2'} \in X$ with $pset_{assign}(r_{j2}') = pset_{assign}(r_{j2})$. Nevertheless, r_{j2}' resulted from the split of role r'. This implies that all the permissions in the $pset_{assign}(r') \setminus pset_{assign}(r)$ are removed from r' and are assigned to r_{j2}'. Therefore, $pset_{assign}(r) \not\subset pset_{assign}(r')$

If we assume $pset_{assign}(r') \subset pset_{assign}(r)$ then, either $pset_{assign}(r_j') \supset pset_{assign}(r_j)$ which is not possible; or there exists at least one more newly created child role r_{j2}' ($r_{j2}' \neq r_j'$) of role r'. In this case $pset_{assign}(r_{j2}') = pset_{assign}(r) \setminus pset_{assign}(r')$. Note that $r_{j2}' \in dom(A)$ and therefore there exists a role $r'' \in \{G_B, G_C\}$ such that either r' *contains* r" or r' *overlaps* r''. The element preservation property of RBAC-integrate ensures that r'' also exists in Q. When integration between G_A and Q is performed role r is compared with r'' and role r is split to produce a child role r_{j2} with $pset_{assign}(r_{j2}) = pset_{assign}(r) \cap pset_{assign}(r'') = pset_{assign}(r_{j2}')$. This proves that $pset_{assign}(r') \not\subset pset_{assign}(r)$ provided r is split once or twice. Using induction we can prove that $pset_{assign}(r') \not\subset pset_{assign}(r)$ is independent of the number of times role r is split. The above implies that for a type (i) split role $r \in Y$, there exists a role $r' \in X$ such that $pset_{assign}(r') = pset_{assign}(r)$, hence $\varphi(r') = r$.

The final step in proving that φ is onto is to show that all the elements in X map to at least one element in Y. The element preservation property of RBAC-integrate maintains that all the user, permissions and type (i) roles that are present in X are also present in Y. So, all the users, permissions and type (i) roles in X can be mapped to at least one element in Y. Since we disallow non-redundant roles and addition of new permissions and users during the process of integration therefore both X and Y have same number of type (ii) roles. We already proved that for every type (ii) role in Y there exists a type (ii) role in X with the same permission assignment. Since the cardinality of type (ii) roles in both X and Y is same, therefore there exists a 1:1 correspondence between the type two roles in X and Y.

This concludes the proof that φ is onto.

φ is 1:1 (for all $e_1, e_2 \in X$, $\varphi(e_1) = \varphi(e_2) \rightarrow e_1 = e_2$)

The element preservation property of the integration algorithm implies that all the elements in the input graphs G_A, G_B, G_C are present in X and Y. Moreover, RBAC-integrate does not add any new user, permission and type (i) roles, i.e., the cardinality of user set, permission set, and type (i) role set is same in both X and Y. We already proved that φ is onto. Since we disallow non-redundant roles and duplicate permission assignment during the process of integration therefore both X and Y have same number of type (ii) roles. This implies that there is 1:1 correspondence between the user, permission and role elements between X and Y. Hence, φ is 1:1.

Relationship Preservation: To conclude the proof that φ is isomorphic, we need to show that any relation $R(U) \in R_X$ if and only if $R(\varphi(U)) \in R_Y$. The relationship preservation property of RBAC-integrate guarantees that each relation R (except the *P-assign*) in the input RBAC graph has a corresponding relationship R' in the integrated RBAC graph. We already proved that for any role *r'* in X, there exists exactly one role *r* in Y such that that $pset_{assign}(r) = pset_{assign}(\varphi(r))$. Moreover, φ is a 1:1 morphism. This implies that for any permission p *P-assign*$(r,p) \in R_X$ if and only *P-assign*$(\varphi(r),p) \in R_Y$.

This concludes the proof that φ is isomorphic, implying that the operator RBAC-integrate is associative. ■

Proof of Theorem 4.7: We prove this theorem separately for *role assignment*, *role-specific SoD*, and *user-specific SoD* constraints.

*Any state S reachable from G is secure with respect to the role-assignment constraint of all collaborating domains***.** Suppose this is not true. This means that in some state S reachable from G there exists a user $u_i \in U_k$ who accesses a role $r_j \in R_k$ (s_{ij} = 1, $s_{ij} \in \pi_{ur\_k}(S)$), while $a_{ij} = 0$, where, $a_{ij} \in \pi_{ur}(A_k^+)$, i.e., there is no intra-domain access path from u_i to r_j. The above implies that in the multi-domain RBAC graph G, there is a path from u_i to r_j that consists of at least two cross-domain edges. Without loss of generality, assume that these cross-domain edges are (r_l, r_m) and (r_n, r_p), where, r_l, $r_p \in R_k$ and r_m, $r_n \notin R_k$; and $\left(u_{ir_l} = 1\right) \wedge \left(r_m \underset{I}{\overset{*}{\geq}} r_n \vee r_m = r_n\right) \wedge \left(r_p \underset{I}{\overset{*}{\geq}} r_j \vee r_p = r_j\right)$.

Since there is no intra-domain access path from u_i to r_j, $u_{ir_j} = 0$ is specified as one of the constraint to the IP problem (constraint transformation rule 1). Therefore, in any feasible solution $u_{ir_j} = 0$ and $u_{ir_p} = 0$ .There are two possibilities for the variable u_{ir_n} in any feasible (optimal feasible) solution:

$u_{ir_n} = 1$. If this is an optimal feasible solution to the IP problem, then step 7 of the algorithm *ConfRes* removes the edge (r_n, r_p).

$u_{ir_n} = 0$. If this yields an optimal solution then step 7 of the algorithm *ConfRes* removes the edge (r_l, r_m) if $u_{ir_m} = 0$, otherwise it removes the edge (r_n, r_p).

In either case, any cross-domain edge leading u_i to r_j through r_n is dropped. If there are multiple such paths through other cross-domain roles, then in a similar manner those paths will be eliminated by *ConfRes*. Hence in the resulting graph G there is no cross-domain path from u_i to r_j, implying that $s_{ij} = 0$. This contradicts our initial assumption.

*Any state S reachable from G is secure with respect to the role-specific SoD constraint of all collaborating domains***.** We prove this statement by considering all possible role-specific SoD violations that might occur as a result of interoperation. The

following cases capture all the role-specific SoD violations in the multi-domain environment:

Case 1: In this case, a local user u_l accesses two conflicting roles r_i and $r_j \in R_k$. There are following sub-cases corresponding to case 1:

Sub-case 1(a): The security policy of domain *k* does not allow u_l to access any of the roles r_i and r_j. If we assume that in some state S, u_l is able to access r_i and r_j through some cross-domain role (see Figure 7.1(a)), then this will be a violation of role-assignment constraint of domain *k*. However, all the reachable states from the multi-domain RBAC graph obtained after applying conflict resolution algorithm, *ConfRes*, are secure with respect to the *role-assignment* constraints of all collaborating domains (proved above). Hence in this sub-case, u_l cannot access r_i and r_j simultaneously.

Sub-case 1(b): RBAC policy of domain *k* allows u_l to access r_i but not r_j as depicted in Figure 7.1(b). Since the multi-domain policy is secure with respect to the role-assignment constraints of domain *k* (proved above), therefore, u_l cannot access r_j through a cross-domain path, implying that SoD violation between r_i and r_j never occurs in this case.

Sub-case 1(c): Suppose u_l is assigned to r_s and $r_s \underset{A}{\geq}^* r_i$, $r_s \underset{A}{\geq}^* r_j$. Moreover, r_i and r_j are conflicting roles as shown in Figure 7.1(c). A *role-specific SoD violation* occurs if u_l activates one of the conflicting roles, say r_i, and inherits the other one, say r_j, through r_t such that $\left(r_s \underset{A}{\geq}^* r_t \vee r_s = r_t\right) \wedge r_t \underset{I}{\geq}^* r_j$. For a hierarchically consistent RBAC policy, the conflicting role set of a junior role must be contained in the conflicting role set of the senior role. $r_t \underset{I}{\geq}^* r_j \Rightarrow conf-rset(r_t) \supseteq conf-rset(r_j)$. This means that $r_i \in$ *conf-rset*(r_t). If there is no inter-domain path from u_l to r_t then user u_l cannot access r_t and r_i simultaneously implying that u_l cannot access r_i and r_j simultaneously. If there exists an inter-domain path from u_l to r_t, then by using induction we can show that there exist a role $r_u \in R_k$ such that $\left(r_s \underset{A}{\geq}^* r_u \vee r_s = r_u\right) \wedge \left(r_u \underset{I}{\geq}^* r_t\right) \wedge conf-role(r_u, r_i)$ and there does not exists a cross-domain role $r_o \notin R_k$ such that $r_s \underset{I}{\geq}^* r_o \underset{I}{\geq}^* r_u$. If $r_s = r_u$ then this leads to sub-

case 1(d) discussed next. If not then this means that u_l cannot access r_u and r_i simultaneously implying that u_l cannot access r_t and r_i simultaneously, which in turns imply that u_l cannot access r_j and r_i simultaneously.

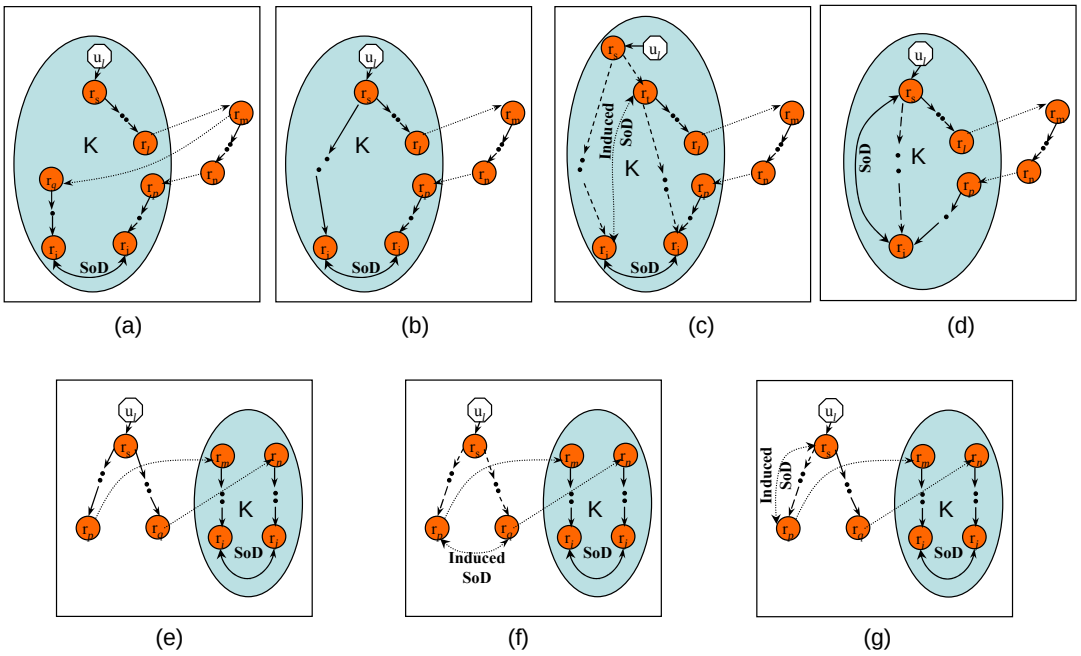

Fig. 7.1. Cases of role-specific *SoD* violations involving cross-domain paths

Sub-case 1(d): Suppose u_l is assigned to r_s and $r_s \underset{A}{\geq}^{*} r_i$. Moreover, r_s and r_i are activation time conflicting roles as shown in Figure 7.1(d). If security policy of domain k is consistent then there is no intra-domain path from r_s to r_i consisting of only *I*-edges. Suppose that there is a cross-domain path from r_s to r_i. Such a path must have at least two cross-domain edges. Without loss of generality, assume that these cross-domain edges are (r_l, r_m) and (r_n, r_p), where, $r_l, r_p \in \mathrm{R}_k$ and $r_m, r_n \notin \mathrm{R}_k$; and $\left(r_s \underset{I}{\geq}^{*} r_l \vee r_s = r_l\right) \wedge \left(r_m \underset{I}{\geq}^{*} r_n \vee r_m = r_n\right) \wedge \left(r_p \underset{I}{\geq}^{*} r_i \vee r_p = r_i\right)$. This cross-domain path enables any user to access permissions of r_i by accessing role r_s, which is a violation of SoD constraint between r_s and r_i. At least one user activates role r_s (Step 1 of the *ConfRes* algorithm and transformation rules 3 and 4 ensures that each role in the multi-domain graph is accessed by at least one user). Let the user be u_l. Since r_s and r_i are conflicting roles, therefore $u_{lr_s} + u_{lr_i} \leq 1$ is one of the constraint of the IP problem formulated in the

step 4 of conflict resolution algorithm *Confres*. Since $u_{lr_s} = 1$, therefore in any feasible solution $u_{lri} = 0$ and $u_{lrp} = 0$. There are two possibilities for the variable u_{lr_n} in any feasible (optimal feasible) solution:

$u_{lr_n} = 1$. If this is an optimal feasible solution to the IP problem, then step 7 of the algorithm *Confres* removes the edge (r_n, r_p).

$u_{lr_n} = 0$. If this yields an optimal solution then step 7 of the algorithm *ConfRes* removes the edge (r_l, r_m) if $u_{lr_m} = 0$, otherwise it removes the edge (r_n, r_p).

In either case, any cross-domain edge leading u_l to r_j through r_n is dropped. If there are multiple such paths through other cross-domain roles, then in a similar manner those paths will be eliminated by *ConfRes*. Hence in the resulting graph G there is no cross-domain path from r_s to r_i, implying that u_l cannot access role r_s and r_i simultaneously.

Case 2: In this case, a foreign user $u_l \notin U_k$ accesses two conflicting roles r_i and $r_j \in R_k$. There are three sub-cases corresponding to case 2. Figures 7.1(e), 7.1(f) and 7.1(g) depicts these sub-cases.

Sub-case 2(a): Suppose u_l is assigned to r_s and there is a cross-domain path from r_s to r_i and from r_s to r_j as shown in Figure 7.1(e). For the cross-domain path from r_s to r_i the following hold:

$$\left(r_s \underset{I}{\geq}^{*} r_p \vee r_p = r_s\right) \wedge \left(r_p \underset{I}{\geq}^{*} r_m\right) \wedge \left(r_m \underset{I}{\geq}^{*} r_i \vee r_m = r_i\right)$$

Similarly, for the cross-domain path from r_s to r_j the following hold:

$$\left(r_s \underset{I}{\geq}^{*} r_q \vee r_s = r_q\right) \wedge \left(r_q \underset{I}{\geq}^{*} r_n\right) \wedge \left(r_n \underset{I}{\geq}^{*} r_j \vee r_n = r_j\right)$$

Since r_i and r_j are conflicting roles and a user u_l assigned to r_s have an access path to both r_i and r_j, therefore $u_{lr_i} + u_{lr_j} \leq 1$ is one of the constraint of the IP problem formulated in the step 4 of conflict resolution algorithm *Confres*. At least one user activates role r_s (Step 1 of the *ConfRes* algorithm and transformation rules 3 and 4 ensures that each role in the multi-domain graph is accessed by at least one user). Let the user be u_l, i.e.,

$u_{lr_s} = 1$, which also implies that $u_{lr_p} = 1$ and $u_{lr_q} = 1$. There are three possibilities for the variables u_{lr_i} and u_{lr_j} in any feasible solution.

$u_{lr_i} = 0$ and $u_{lr_j} = 0$, implying that $u_{lr_m} = 0$ and $u_{lr_n} = 0$. If this is an optimal solution then step 7 of *ConfRes* removes the edges (r_p, r_m) and (r_q, r_n).

$u_{lr_i} = 0$ and $u_{lr_j} = 1$, implying that $u_{lr_m} = 0$. If this is an optimal solution then step 7 of *ConfRes* removes the edge (r_p, r_m).

$u_{lr_i} = 1$ and $u_{lr_j} = 0$, implying that $u_{lr_n} = 0$. If this is an optimal solution then step 7 of *ConfRes* removes the edge (r_q, r_n).

In any of the above cases, at least one of the cross-domain paths from r_s to r_i or r_j is removed in the process of conflict resolution. Hence, u_l cannot access both r_i and r_j simultaneously in the resulting RBAC graph G.

Sub-case 2(b): Suppose u_l is assigned to r_s and $r_s \underset{A}{\geq}^* r_p \wedge r_s \underset{A}{\geq}^* r_q$. Let there be a cross-domain path from r_p to r_i and a cross-domain path from r_q to r_j. This is depicted Figure 7.1(f). These cross-domain relationship $r_p \underset{I}{\geq}^* r_i$ and $r_q \underset{I}{\geq}^* r_j$ induces an *SoD* constraint between r_p and r_q as shown in Figure 7.1(e). This implies that user u_l cannot activate r_p and r_q concurrently, and therefore cannot access the cross-domain roles r_i and r_j simultaneously.

Sub-case 2(c): Suppose u_l is assigned to r_s and $r_s \underset{A}{\geq}^* r_p \wedge \left(r_s \underset{I}{\geq}^* r_q \vee r_s = r_q \right)$. Let there be a cross-domain path from r_p to r_i and a cross-domain path from r_q to r_j. The relation $r_q \underset{I}{\geq}^* r_j$ implies $r_s \underset{I}{\geq}^* r_j$. This is depicted Figure 7.1(g). These cross-domain relationship $r_p \underset{I}{\geq}^t r_i$ and $r_s \underset{I}{\geq}^t r_j$ induces an SoD constraint between r_p and r_s as shown in Figure 7.1(e). This implies that user u_l cannot activate r_s and r_p concurrently, and therefore cannot access the cross-domain roles r_i and r_j simultaneously.

Any of the *role-specific SoD* constraint can be reduced to one of the above cases. In all of the above cases, we have proved that SoD violation between conflicting roles can

never happen. Hence, any state S reachable from the multi-domain RBAC graph G obtained after applying conflict resolution algorithm, *ConfRes*, is secure with respect to the *role-specific SoD* constraints of all collaborating domains.

*Any state S reachable from G is secure with respect to the user-specific SoD constraint of all collaborating domains***.** A user-specific SoD violation of role r_t occurs when a user u_i belonging to the conflicting user set(s) of r_t accesses r_t through multiple paths and at least one of such path includes cross-domain edges. This is shown in Figure 7.2, in which users u_1, u_2,.., u_m conflict with user u_{dt} for role r_t. The following relationship exists among the roles depicted in Figure 7.2.

$$\left(r_s \underset{A}{\geq}^{*} r_t \vee r_s = r_t\right) \wedge \left(r_s \underset{I}{\geq}^{*} r_l \vee r_s \underset{A}{\geq}^{*} r_l \vee r_s = r_l\right) \wedge \left(r_l \underset{I}{\geq}^{*} r_m\right) \wedge \left(r_m \underset{I}{\geq}^{*} r_n\right) \wedge \left(r_n \underset{I}{\geq}^{*} r_p\right) \wedge \left(r_p \underset{I}{\geq}^{*} r_t \vee r_p = r_t\right)$$

Where, r_s, r_l, r_p, and $r_t \in R_k$; and r_m and $r_n \notin R_k$, otherwise, domain k's RBAC policy becomes inconsistent. The case when r_s and r_t are not distinct is trivial and does not involve any cross-domain path for *SoD* violation. The following discussion considers the case when r_s and r_t are distinct roles.

In Figure 7.2, a user specific SoD is violated when u_{dt} activates role r_t and any of the users conflicting with u_{dt} for role r_t accesses role r_l. By accessing role r_l, a user, say u_1, accesses the permissions of r_t through the cross-domain path.

After step 3 of the conflict resolution algorithm, *ConfRes*, all the user specific SoD constraints in the multi-domain RBAC graph G can be reduced to the case shown in Figure 7.2. Since users u_1, u_2,.., u_m conflict with user u_{dt} for role r_t, therefore the following is included as one of the constraints to the IP problem formulated in step 4 of C*onfRes*.

$\sum_{i=1}^{m} u_{ir_t} + u_{dtr_t} \leq 1$, Also u_{dtr_t} is set to one in step 3 of the algorithm *Confres*. This implies that in any feasible solution the the IP problem, $u_{ir_t} = 0$ for all $i \in \{1,2,..,m\}$.

There are two possibilities for the variable u_{ir_n} in any feasible (optimal feasible) solution:

$u_{ir_n} = 1$. If this is an optimal feasible solution to the IP problem, then step 7 of the algorithm *ConfRes* removes the edge (r_n, r_p).

$u_{ir_n} = 0$. If this yields an optimal solution then step 7 of the algorithm *ConfRes* removes the edge (r_l, r_m) if $u_{ir_m} = 0$, otherwise it removes the edge (r_n, r_p).

In either case, any cross-domain edge leading u_i to r_t through r_n, is dropped. If there are multiple such paths through other cross-domain roles, then in a similar manner those paths will be eliminated by *ConfRes*. This implies that no user u_i belonging to the conflicting user set(s) of r_t can access r_t through a cross-domain path.

Hence, any state S reachable from the multi-domain RBAC graph G obtained after applying conflict resolution algorithm, *ConfRes*, is secure with respect to the *user-specific SoD* constraints of all collaborating domains provided their access control policies are consistent.

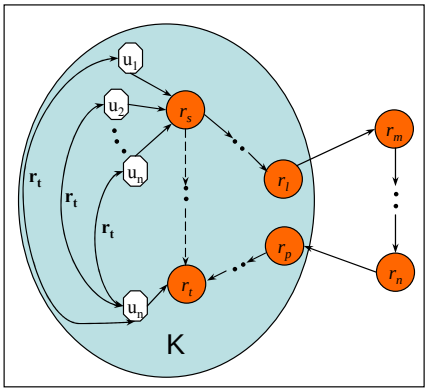

Fig. 7.2. User-specific SoD violation through a cross-domain path

This concludes the proof of Theorem 4.7. ■