ACCESS CONTROL MANAGEMENT AND SECURITY IN MULTI-DOMAIN COLLABORATIVE ENVIRONMENTS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Basit Shafiq

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2006

Purdue University

West Lafayette, Indiana

This thesis is dedicated to my parents.

ACKNOWLEDGMENTS

I would like to sincerely thank my PhD advisor Professor Arif Ghafoor for his invaluable guidance and support at each step of my graduate studies at Purdue University. I am highly indebted to Professor Elisa Bertino for her unequivocal help. I relied heavily on her feedback throughout my research. I would also like to express my gratitude to Professors Mary P. Harper and Yu C. Hu for their participation in my PhD committee. Their guidance and suggestions have been very valuable.

I am also indebted to Center of Education and Research in Information Assurance and Security (CERIAS) at Purdue University for the unfettered support provided throughout my doctoral studies. I would like to thank all my colleagues in the Distributed Multimedia Systems Laboratory at Purdue University for their assistance, encouragement, and discussions. I am grateful to Wasfi Al-Khatib, Husni Fahmi, James Joshi, Rafae Bhatti, Ammar Masood, Arjmand Samuel, Hasib Siddiqui, Mohammed Shehab, and Hasib Amjad for their valuable help and friendship. Finally, I would like to acknowledge the support provided by the National Science Foundation through Grants IIS-0209111, IIS-0242419, and ITR-0428554.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABSTRACT

Basit Shafiq. Ph.D.. Purdue University, August 2006. Access Control Management and Security in Multi-Domain Collaborative Environments. Major Professor: Arif Ghafoor.

With the increase in information and data accessibility, there is a growing concern for security and privacy of data. In large corporate Intranets, the *insider attack* is a major security problem. Numerous studies have shown that unauthorized accesses, in particular by insiders, pose a major security threat for distributed enterprise environments. This problem is highly magnified in a multi-domain environment that spans multiple enterprises collaborating to meet their business requirements. The challenge is in developing new or extending existing security models for efficient security management and administration in multi-domain environments that allow extensive interoperation among individuals or systems belonging to different security domains.

In this dissertation, we have addressed the issue of secure interoperation from policy management perspective. In particular, we have developed a policy-based framework that allows secure information and resource sharing in multi-domain environments supporting both tightly-coupled and loosely-coupled collaborations. The level of coupling in such environments is characterized by the degree of interoperation, the level of trust among domains, and the security, autonomy, and privacy requirements of the collaborating domains. The proposed framework provides efficient solutions and strategies for ensuring secure interoperation in both tightly-coupled and loosely-coupled multi-domain environments. This framework is designed for distributed systems that employ role based access control (RBAC) policies, and therefore addresses the secure interoperability requirements of emerging distributed application systems.

1. INTRODUCTION

1.1. Research Motivation and Problem Statement

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 collaborative environments 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 management, subcontracting relationships, or joint marketing ventures [38]. In the public sector, government has taken various initiatives to increase collaboration among government agencies and non-government organizations (NGOs) in order to provide better public service to citizens, and provide accurate and comprehensive information to relevant government agencies and general public in a timely manner. Two major initiatives in this regard are Digital Government Program and Integrated Justice Information Systems. The aim of the Digital Government Program is to use 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 [43]. 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 [68]. 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. In large corporate Intranets, the *insider attack* is a major security problem. Numerous studies have shown that unauthorized accesses, in particular by insiders, pose a major security threat for distributed enterprise environments [105]. This problem is highly magnified in a multi-domain system that spans multiple enterprises collaborating to meet their business requirements [75, I3P]. The challenge is in developing new or extending existing security models for efficient security management and administration in multi-domain environments that allow extensive interoperation among individuals or systems belonging to different security domains. A security domain, in the context of collaborative environment, is a bounded group of protected objects and users to which applies a single security policy executed by a single security administrator [60].

In a multi-domain environment, domains may use different models, semantics, classification schemes, and constraints for representing their information security policies [26, 40, 106, 101, 61]. The underlying objective of any domain's security policy is to protect the information systems, managed by the domain, against unauthorized accesses and against denial of service to authorized users. Any solution for secure interoperation in a multi-domain environment must be designed based on this objective, which can be elaborated as the following set of security goals [75, 88]:

Confidentiality: assuring that information is not disclosed without proper authorization from the owner domain.

Integrity: assuring that information is not modified without proper authorization from the owner domain.

Availability: assuring that information is accessible to authorized users when required.

Accountability: assuring that every action of a user in the multi-domain environment is uniquely traced back to that user.

Authentication, access control, and auditing have been considered as the key security services employed by domains to meet the above security goals [110b, 108]. Authentication establishes the identity of a user and is a prerequisite for access control. Access control deals with the authorization management and limits the actions or operations that a legitimate user can perform in the system. The auditing mechanism collects data about the system's activity and detects possible security breaches. The focus of this dissertation is to develop solutions for secure interoperation among collaborating domains based on the authorizations specified in the domains' security policies. Therefore, we only consider the security issues related to access control and authorization management in multi-domain environments.

Several access control models have been proposed in literature to address the diverse security requirements of information systems. Traditional access models can be classified into two broad categories: discretionary access control (DAC) [62, 65, 72, 85, 108] and mandatory access control (MAC) [16, 25, 86, 94]. DAC Models allow subject to grant their privileges over their owned objects to other subjects. The subjects in this model can be users, groups, or processes that act on behalf of other subjects. This high degree of flexibility in DAC models can let unauthorized users to find ways to access protected objects. Therefore, DAC models do not provide adequate mechanisms to support secure information and resource sharing in multi-domain systems. MAC models use a classification approach for labeling subjects and objects. These security classifications lead to various clearance levels for access control. To avoid unauthorized flow of sensitive information, the MAC model prohibits users with low clearance level to read information objects at higher security levels (*no read-up*). In addition, MAC can also enforce the *no write-down* rule that prohibits users, at high classification level to write to objects at lower level.

MAC only addresses the multi-level security aspect of secure interoperation [61, 28]. Multi-level security or Bell-Lapadula [15] model is more suitable for environments which have static security constraints and cannot be used to capture the dynamic constraint requirements of emerging applications and information systems [74, 75, 19]. Separation of duty (SoD), event dependencies, and time-dependent authorizations 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 [12, 54, 55, 42, 19]. Role based access control (RBAC) [110, 47] and its time-based extended models [Ber01a, 78, 79] provide a promising approach to satisfy the access control requirements of the afore-mentioned applications [19, 3]. Furthermore, RBAC is capable of modeling a wide range of access control policies including discretionary access control (DAC) and mandatory access control (MAC) [101].

In this dissertation, we have addressed the issue of secure interoperation from policy management perspective. In particular, we have developed a policy-based framework that allows secure information and resource sharing in multi-domain environments supporting both tightly-coupled and loosely-coupled collaborations. The level of coupling in such environments is characterized by the degree of interoperation, the level of trust among domains, and the security, autonomy, and privacy requirements of the collaborating domains. For instance in a digital government setting, a tightly-coupled collaboration is needed to support sharing of sensitive and time-critical information among various government agencies. On the other hand, a company that outsources its non-core business processes to external partners creates a loosely-coupled environment with the external partners. The proposed framework provides efficient solutions and strategies for ensuring secure interoperation in both tightly-coupled and loosely-coupled multi-domain environments. This framework is designed for distributed systems that employ role based access control (RBAC) policies, and therefore addresses the secure interoperability requirements of emerging distributed application systems including distributed service-based systems and workflow management systems.

1.2. Summary of Contributions

In this research, we address the issue of secure interoperation in multi-domain environment with varying degree of coupling and association among collaborating domains. Our main objectives are:

To develop a framework that allows secure information and resource sharing among heterogeneous and autonomous policy domains.

To develop a methodology that guides the development of secure and verifiable distributed applications based on the policies of collaborating domains.

The contributions of the research reported in this dissertation can be summarized as follows:

We have developed a policy composition methodology that generates a secure and conflict-free global meta-policy from the access control policies of individual domains. This methodology is designed for federated systems employing RBAC policies. Two key aspects of this methodology include: composition of a global meta-policy from the RBAC policies of domains with diverse and possibly conflicting security requirements, and optimal resolution of conflicts in the global-meta policy.

We have proposed a policy merging algorithm that resolves the constraint heterogeneities among the RBAC policies of different domains and establish cross-domain role mappings based on the permission assignment and hierarchical ordering of corresponding roles. Such a mapping enables inter-domain information and resource sharing via mapped roles. In addition to the automated generation of role mapping between cross-domain roles, the proposed approach allows security policy administrators to map cross-domain roles based on the requirements of collaborative applications.

The meta-policy generated by merging the RBAC policies of domains may not be consistent and may not satisfy the security constraints of collaborating domains. For resolution of such policy conflicts, we have proposed a novel integer-programming based approach. The proposed approach generates a secure and consistent meta-policy by finding a set of non-conflicting role-mappings that maximizes inter-domain accesses according to the specified optimality criterion. The notion of optimality is further analyzed in terms of trade-offs between autonomy and the level of interoperation.

We have proposed an approach for verifying secure composibility of distributed applications requiring interactions among autonomous domains in a loosely-coupled multi-domain environment. The proposed approach is designed for verifying the specification of distributed services or workflows for conformance with the time-dependent access control policies of collaborating but autonomous domains. The time-dependent policies of domains are represented using generalized temporal role based access control (GTRBAC) model.

We have analyzed the trade-off between the proposed meta-policy composition approach for federated collaboration and the service composibility verification approach for loosely-coupled collaborative systems. For this analysis we have proposed a set of metrics. We have also performed a comparative analysis between the policy composition and verification approaches proposed in this dissertation and the existing approaches for composition and verification of distributed systems, protocols, and services.

1.3. Outline of Dissertation

The dissertation is organized as follows. In Chapter 2, we first present a brief overview of the RBAC and GTRBAC models and then discuss the characterization of multi-domain collaborative environments with respect to the degree of coupling among the domains. In Chapter 3, we describe the proposed policy composition framework that generates a global meta-policy to facilitate secure information and resource sharing in federated multi-domain environments. In Chapter 4, we present a multi-policy based approach for verifying secure composibility of distributed workflow based applications requiring interactions among autonomous domains in a loosely-coupled multi-domain environment. In Chapter 5, we first discuss the trade-off between the global meta-policy based approach and distributed multi-policy based approach with respect to various collaboration metrics. In addition, we present a comparison between the proposed policy composition and verification framework and the existing approaches for verification of distributed systems and services. Finally, in Chapter 6, we provide conclusion and discuss the future work.

2. OVERVIEW

In this chapter, we provide a brief overview of the RBAC model and its temporal extensions. In addition, we discuss the characterization of multi-domain systems with respect to the degree of coupling among the domains.

2.1. Role Based Access Control (RBAC)

Role based access control (RBAC) models are receiving increasing attention as a generalized approach to access control [47, 58, 59, 74, 75, 81, 98, 99, 101, 109, 110, 111, 112, 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/she is a member. Role-based approach naturally fits into organizational contexts as users are assigned organizational roles that have well-defined responsibilities and qualifications [48].

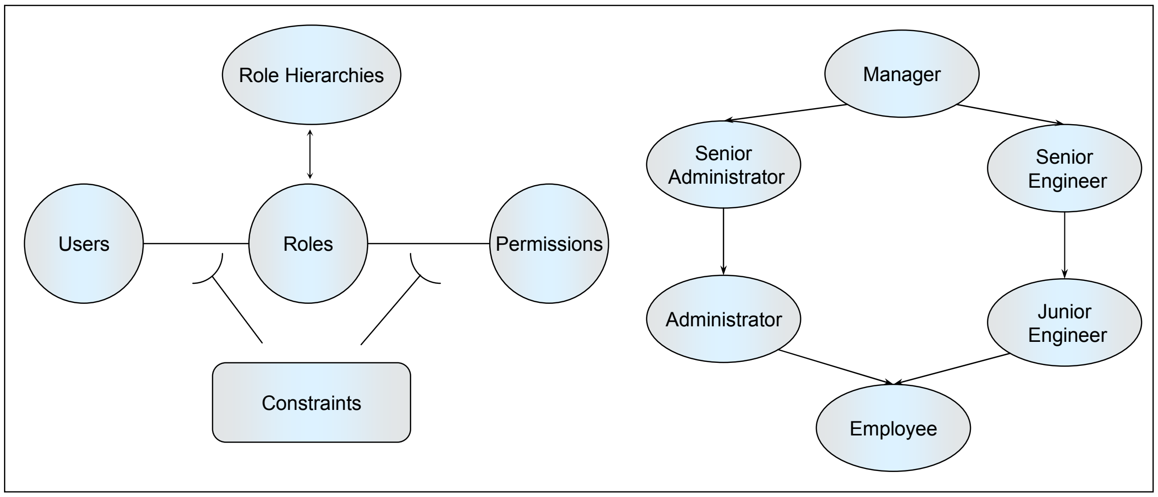


Fig. 2.1 Constraints and hierarchy in RBAC

According to a survey conducted by the U.S. National Institute of Standards and Technology (NIST) [48], 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 [75, 110]:

- *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 [114]. 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 [75, 110].

- *Separation of Duties*: Separation of duties (SoD) has been considered a very desirable organizational security requirement [3, 20, 21, 23, 84, 99, 107, 121]. 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 [110]. This further increases the manageability of authorizations.
- *Policy-neutrality*: Role-based approach is policy-neutral and is a means for articulating policy [75, 110]. Role-based systems can be configured to represent many useful DAC, MAC policies [Nay95, Osb97, 101b] and user-defined and organizational security policies.

2.1.1. The NIST RBAC model

The RBAC model [47], currently being used as the basis for 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, the user establishes a session by activating a set of enabled roles that the user 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 requested roles.

Several functions are defined for the sets `Users, Roles, Permissions`, and `Sessions`. The *user role assignment* (UA) and the *role permission assignment* (PA) functions model the assignment of users to roles and the assignment of permissions to roles, respectively. The *user* function maps each session to a single user, whereas the *role* function establishes a mapping between a session and a set of roles activated by the corresponding user in the session.

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 [114, 113, 78].

2.2. Generalized Temporal Role Based Access Control (GTRBAC) Model

The GTRBAC model [78] incorporates a set of language constructs for specification of various temporal and periodicity constraints on role, including constraints on role enabling, role activation, user-to-role assignment, and permission to role assignment. In particular, GTRBAC makes a clear distinction between role *enabling* and role *activation*. A role is *enabled* if a user can acquire the permissions assigned to the role. An *enabled* role becomes active when a user acquires its permissions in a session. By contrast, a *disabled role* cannot be activated by any user. The GRTBAC model allows specification of the following set of constraints:

1. *Temporal constraints on role enabling/disabling*. These constraints are used to specify time intervals during which a role is enabled. It is also possible to specify enabling duration for a role.

2. *Temporal constraints on user-role and role-permission assignments*. These constraints allow specification of intervals and durations in which a user or permission is assigned to role.
3. *Activation constraints*. These constraints are used to specify restrictions on the activation of a role. These include, for example, specifying the total duration for which a user is allowed to activate a role or the number of users that can be allowed to activate a particular role in concurrent sessions.
4. *Runtime events*. A set of run-time events allows an administrator to dynamically initiate GTRBAC events or a user to issue role-activation requests.
5. *Triggers*. Triggers are used to specify dependency among GTRBAC events.
6. *Separation of duties* (*SoD*). SoD constraints are used to prevent conflicting users from assuming the same role or to prohibit assumption of conflicting roles by same user.

The temporal constraint expressions in GTRBAC are summarized in Table 2.1. The GTRBAC events and status predicates used in the GTRBAC constraint expressions are listed in Table 2.2. The periodicity constraints on various GTRBAC events are specified using the expression (I, PE), where PE is a periodic expression denoting an infinite set of periodic time instants, and I = [begin, end] is a time interval denoting the lower and upper bounds that are imposed on instants in PE [20, 21, 78]. The periodic time uses the notion of calendar defined as a countable set of contiguous intervals [20, 21]. The GTRBAC model consider a set of calendars with granularities in *minutes*, *hours*, *days*, *weeks*, *months*, and *years*. A subcalendar relation can be established among these calendars. Given two calendars Cal_1 and Cal_2, Cal_1 is said to be a subcalendar of Cal_2, written as $Cal_1 \sqsubseteq Cal_2$, if each interval of Cal_2 is covered by a finite number of intervals of Cal_1. A periodic expression PE is formally defined as:

$$PE = \sum_{i=1}^{n} O_i Cal_i \rhd x.Cal_d$$

Where Cal_d, Cal_1, Cal_2,…,Cal_n are calendars and $O_n = all$, $O_i \in 2^{\mathbb{N}} \cup \{\text{all}\}$, $Cal_{i-1} \sqsubseteq Cal_i$, and $x \in \mathbb{N}$. The symbol $\rhd$ separates the first part of the periodic expression that distinguishes the set of starting points of the intervals, from the specification of the duration of each interval in terms of the calendar Cal_d. As an example consider the periodic expression $\{all.days, \{9, 15, 23\}.Hours, \{20, 50\}.Minutes \triangleright 15.Minutes\}$. This periodic expression represents the following set of intervals {[09:20, 09:35], [09:50, 10:05], [15:20, 15:35], [15:50, 16:05], [23:20, 23:35], [23:50, 23:59], [00:00, 00:05], [09:20, 09:35], [09:50, 10:05]…..}.

A set of time instants corresponding to a periodic constraints expression (I, PE) is denoted by *Sol*(I, PE). Similarly, the set of time intervals in (I, PE) is denoted by $\Gamma(I, PE)$. For example, for I = [0, 1440 minutes (1 day)] and $PE = \{all.days, \{9, 15, 23\}.Hours, \{20, 50\}.Minutes \triangleright 15.Minutes\}$, $\Gamma(I, PE)$ = {[09:20, 09:35], [09:50, 10:05], [15:20, 15:35], [15:50, 16:05], [23:20, 23:35], [23:50, 23:59], [00:00, 00:05]}.

Example 2.1: Consider the GTRBAC policy of a *medical information system* (MIS) given in Table 2.3. Row 1 of this table lists all the periodic expressions that are used to constrain the assignment and enabling of roles in the MIS policy. In row 2A, the enabling times of the role `DayDoctor` is restricted to *DayTime* (9:00am-9:00pm) and the enabling time of `NightDoctor` is restricted to *NightTime* (9:00pm-9:00am). In row 2B, *Adams* is assigned the role `DayDoctor` on every Monday, Wednesday, and Friday of the year 2006. In row 2C, *Carol* is assigned the role of `HeadNurse` in the second shift of every day. Trigger 3A in Table 2.3, indicates that the role `NurseinTraining` is enabled ten minutes after the `HeadNurse` role is activated by *Carol*. As a result, a nurse-in-training can have access to the system only if *Carol* is on duty. Trigger 3B captures the dependency relationship between `NightDoctor` and `NightNurse` roles. Accordingly, enabling of `NightDoctor` role enables `NightNurse` role after ten minutes. Similarly, disabling of `NightDoctor` role triggers disabling of `NightNurse` role. The activation constraint 4A limits the activation duration of `NurseinTraining` role to two hours. Row 4B specifies that at most ten users can activate the `DayNurse` role at any time.

Table 2.1
GTRBAC constraints

$r \in$ R, $u \in$ U, $p \in$ P, tg $\in$ TRG R is a set of roles, U is a set of users, P is a set of permissions, and TRG is a set of Triggers			
Constraints		Expression	Semantics
User-role assignment		([(I,PE)|D], assign$_U$ r to u)	Role r is assigned to user u. This user-to-role assignment is valid for all intervals contained in Γ(I, PE) for a duration D
Role-Permission assignment		([(I,PE)|D], assign$_p$ p to r)	Permission p is assigned to role r. This permission-to-role assignment is valid for all intervals contained in Γ(I, PE)
Role enabling		([(I,PE)|D] enable r)	Role r is enabled during the time intervals contained in Γ(I, PE) for a duration D.
Duration constraint on role activation		(D, active r)	The activation duration of role r in any session must be less than or equal to D.
Run-time Request	Users' request	(activate/deactivate r for u)	request for activation/deactivation of role r for u
	Adminis-trator's request	(enable/disable r) (assign$_U$ r to u) (assign$_p$ p to r)	administrator's request for role enabling/disabling or user-to-role assignment, or permission-to role assignment
Trigger		$ev, sp_1, \ldots sp_k \rightarrow ev'$ after Δt	Event ev is followed by the event ev' after a time duration of Δt, provided that all status predicates $sp_1, \ldots sp_k$ hold at the time of the occurrence of ev.
Static SoD	Role-specific	SSoD$_{Role}$(R', u)	$\forall r_1, r_2 \in R' \subseteq R$, ur-assigned($u$, r_1) $\Rightarrow$ $\neg$ur-assigned(u, r_2)
	User-specific	SSoD$_{User}$(r, U')	$\forall u_1, u_2 \in U' \subseteq U$, ur-assigned($u_1$, r) $\Rightarrow$ $\neg$ur-assigned(u_2, r)
Dynamic SoD	Role-specific	DSoD$_{Role}$(R', u)	$\forall r_1, r_2 \in R' \subseteq R$, u-active($u$, r_1) $\Rightarrow$ $\neg$u-active(u, r_2)
	User-specific	DSoD$_{User}$(r, U')	$\forall u_1, u_2 \in U' \subseteq U$, u-active($u_1$, r) $\Rightarrow$ $\neg$ u-active(u_2, r)

Table 2.2
GTRBAC events and status predicates

$r \in$ R, $u \in$ U, $p \in$ P, tg $\in$ TRG R is a set of roles, U is a set of users, P is a set of permissions, and TRG is a set of Triggers			
Simple Event		Status Predicate	Semantics
enable *r*		ur-assigned(*u*, *r*)	*u* is assigned to *r*
disable *r*		pr-assigned(*p*, *r*)	*p* is assigned to *r*
activate *r* for *u*		r-enabled(*r*)	*r* is enabled
De-activate *r* for *u*		r-active(*r*)	r is active in at least one user's session
		u-active(*u*, *r*)	*r* is active in *u*'s session
		trg-enabled(*tg*)	Trigger *tg* is enabled, i.e., the event *ev* defined in the body of the trigger *tg* has occurred and the status predicates hold.

Table 2.3
GTRBAC policy of a medical information system (MIS)

		Policy Specification	Explanation
1	A	Daytime = (all.Days, + 10.Hours $\triangleright$ 12.Hours)	Day time is from 9:00am to 9:00pm
	B	Nighttime = (all.Days, + 22.Hours $\triangleright$ 12.Hours)	Night time is from 9:00pm to 9:00am
	C	(M, W, F) = (all.Weeks, + {1, 3, 5}.Days + 1. Hours $\triangleright$ 24.Hours)	Monday, Wednesday, and Friday of every week.
	D	Second-Shift = (all.Days, + 13.Hours $\triangleright$ 5.Hours)	Second Shift starts from 12:00 noon and ends at 5:00pm
2	A	(*DayTime*, enable DayDoctor), (*NightTime*, enable NightDoctor)	Enable DayDoctor during day time, Enable NightDoctor during night time
	B	(([1/1/2006,12/31/2006] (M, W, F)), assign$_U$ *Adams* to DayDoctor)	Adam is assigned the role DayDoctor on Monday, Wednesday and Friday of every week of the year 2006.
	C	(*Second-Shift*, assign$_U$ *Carol* to HeadNurse)	User Carol is assigned the role of HeadNurse everyday from 12 noon to 5 pm (second-shift)
3	A	(activate HeadNurse for *Carol* $\rightarrow$ enable NurseInTraining after 10 *min*)	When *Carol* activates the HeadNurse role, NurseInTraining role is enabled after 10 minutes
	B	(enable NightDoctor $\rightarrow$ enable NightNurse after 10 *min*); (disable NightDoctor $\rightarrow$ disable NightNurse)	When the NightDoctor role is enabled, the Night nurse is also enabled after 10 minutes. similarly, disabling of NightDoctor role triggers disabling of NightNurse role.
4	A	(2 *hours*, active$_{R\_total}$ NurseInTraining)	Nurse in training role can be activated for a total of 2 hours
	B	(10, active$_{R\_n}$ DayNurse);	At most 10 users can activate the role DayNurse

2.3. Taxonomy of Multi-Domain Collaborative System

The strategies and techniques used for assuring secure information and resource sharing depend on the basic structure of the collaborative environment which is determined by the following metrics: degree of interoperation, degree of autonomy, degree of privacy, and verification complexity (complexity of assuring secure interaction among collaborating domains). Based on these metrics, we can define three types of collaborations: Federated (tightly-coupled) collaboration, loosely-coupled collaboration, and ad-hoc collaboration. These collaboration types are depicted in Fig. 2.2. In the following, we briefly describe the above metrics used for characterizing collaboration types. These metrics are formally defined in Chapter 5 with reference to the formal model for policy analysis proposed in this dissertation. After introducing these metrics, we describe the key issues and challenges related to secure interoperation in all three types of collaboration.

2.3.1. Collaboration metrics

We use the following metrics to characterize any collaboration. These metrics include: i) *degree of interoperation*, ii) *degree of autonomy*, iii) *degree of privacy*, and iv) *verification complexity*. Note that these metrics can also be used to determine the effectiveness of policies and mechanisms employed by the collaborative systems to satisfy the interoperability and security requirements.

The *degree of interoperation* is a measure of information and resource sharing in a multi-domain environment. We evaluate the *degree of interoperation* in terms of the number of cross-domain accesses supported by the collaborative system. These cross-domain accesses may correspond to querying of a collaborating domain's database by external users/agents or use of a domain's computational and storage resources for processing of distributed services.

Autonomy refers to the ability of a domain to carryout its local operations and activities without any interference from cross-domain accesses or services provided to external users. An autonomous domain can deny any cross-domain access if such access violates its local policy constraints or conflicts with its local operations. In other words, the local operations of an autonomous domain are logically unaffected by its participation in the collaborative environment. This notion of autonomy is similar to the notion of *execution autonomy* in federated database systems, which allows component database systems to abort or delay any transaction that does not meet its local constraints [118].

The *degree of privacy* specifies how much information a domain is willing to disclose about its internal (access control) policies and constraints. Generally, the access control policy of any domain is considered as a protective object as it contains information about the domain's organizational structure, business strategies, security mechanisms, and other protective resources [133, 134, 128, 129, 89, 116]. Therefore, disclosing the contents of domain's access control policy may leak sensitive information which can be misused by adversaries.

Verification complexity refers to the overhead associated with verifying the correctness of distributed applications that require interaction among different systems or domains. This overhead can be evaluated in terms of the algorithmic complexity of the verification approach. In addition, we also consider the overhead associated with structuring, organization, and management of data and policies for facilitating secure information and resource sharing.

2.3.2. Collaboration types

In the following, we describe the three types of collaboration depicted in Fig. 2.2**.**

2.3.2.1. Federated collaboration

Federated collaborations are characterized by the high degree of mutual dependence and trust among the collaborating domains and are used to support long-term interactions. In this respect, a digital government can be considered as a federated collaborative system that provides a set of services by integrating several government agencies. Similarly, the integrated information system for sharing criminal records among justice and public safety agencies can be characterized as federated collaboration.

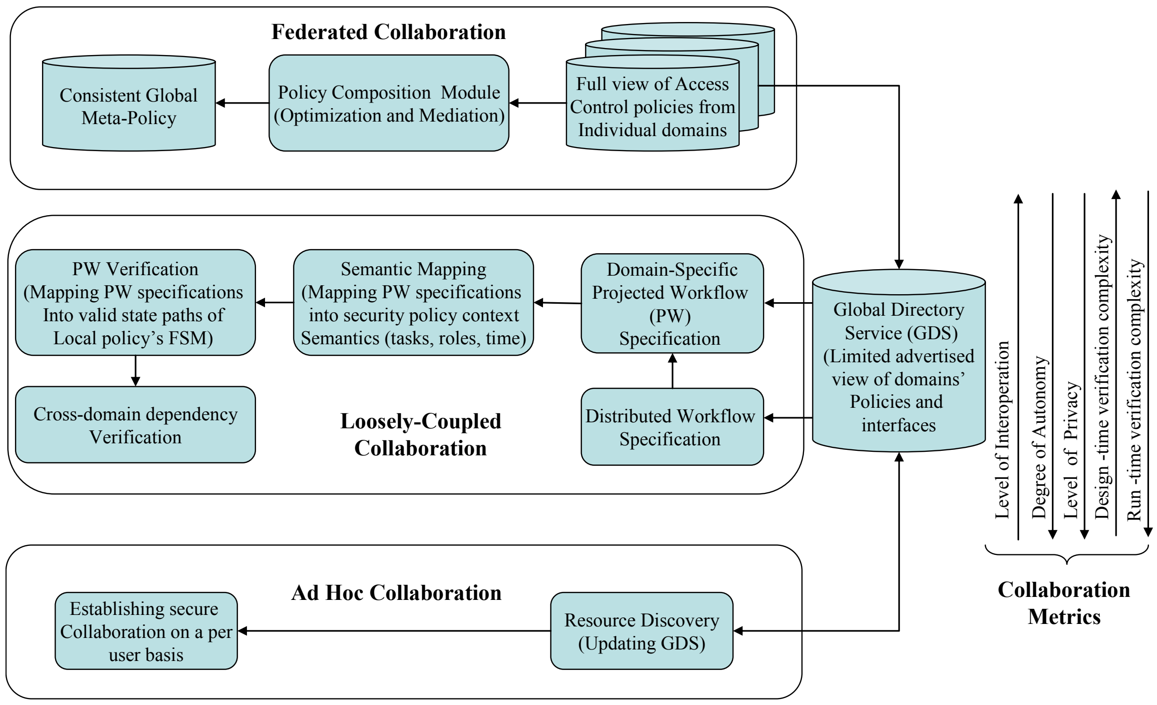

Federated Collaboration
Consistent Global Meta-Policy
Policy Composition Module (Optimization and Mediation)
Full view of Access Control policies from Individual domains
PW Verification (Mapping PW specifications Into valid state paths of Local policy's FSM)
Semantic Mapping (Mapping PW specifications into security policy context Semantics (tasks, roles, time)
Domain-Specific Projected Workflow (PW) Specification
Cross-domain dependency Verification
Loosely-Coupled Collaboration
Distributed Workflow Specification
Global Directory Service (GDS) (Limited advertised view of domains' Policies and interfaces
Level of Interoperation
Degree of Autonomy
Level of Privacy
Design -time verification complexity
Run -time verification complexity
Collaboration Metrics
Ad Hoc Collaboration
Establishing secure Collaboration on a per user basis
Resource Discovery (Updating GDS)

Fig. 2.2 Characterization of collaboration in multi-domain environment

Federated collaborations are designed to support time-critical and safety critical distributed applications requiring high degree of information sharing among collaborating domains. To facilitate secure and timely access to information in a federated multi-domain environment, a global meta-policy is needed that defines the access rights of individuals in one domain over the information resources in other domains. There are two key advantages of using a meta-policy based approach: i) it provides a single interface for accessing information and data resources distributed across multiple domains, thus hiding the heterogeneities and semantic differences among the local policies of domains. ii) The meta-policy can guide the development of secure distributed applications in the federated system. Accordingly, any distributed application that conforms to the meta-policy can be supported by the domains forming the federated system. However, for secure interoperation, the global meta-policy needs to be consistent with the local policies of domains. In particular, the global meta-policy should not allow any inter-domain access that violates the local policy constraints of any domain.

Composition of a global meta-policy is a challenging problem and requires extensive mediation among domains for resolution of policy conflicts. These conflicts may arise because different domains may use different models, semantics, schema format, data labeling schemes, and constraints for representing their local access control policies [26, 40, 106, 101, 61]. Resolution of these conflicts requires full disclosure of domains' policies, thereby reducing their privacy. The meta-policy may also restrict the autonomy of domains by adding new constraints in their local policies.

For federated collaboration, we have developed policy composition framework that generates a global meta-policy from the access control policies of individual domains. This policy composition framework corresponds to the topmost block of Fig. 2.2 and is designed for federated systems employing role-based access control (RBAC) policies. In this framework, the global meta-policy is generated from RBAC policies of the collaborating domains by defining inter-domain role mappings across domains. Such mappings enable inter-domain information and resource sharing via mapped roles. In addition to the automated generation of role mappings between cross-domains roles, the framework also allows security policy administrators to map cross-domain roles based on the interoperation requirements of the federated system. The RBAC policies of collaborating domains may have conflicting security and access control requirements which may cause serious security implications in terms of unauthorized accesses and erroneous system behavior. To resolve such inconsistencies and conflicts in the meta-policy, we have proposed a systematic approach for policy synthesis and conflict resolution with various optimality measures, including, maximizing information and data sharing, maximizing prioritized accesses, and minimizing constraint relaxation.

As discussed above, conflict resolution may require strong mediation among domains' policies, and may trigger policy transformations to support secure collaboration. Such transformation in policies, although increases interoperation among collaborating domains, may result in a loss of their autonomy. A key requirement for developing the global meta-policy is to allow maximum autonomy. Although, violations of domain's security policy are generally not permissible, some domains may concede their autonomy for allowing an increased level of interoperation. In the proposed approach, the problem of secure interoperation is formulated as an optimization problem with an objective of maximizing interoperability with minimum autonomy losses and without causing any security violations of collaborating domains. This optimization problem is solved using 0-1 integer programming based technique with the given optimality measure. The overall process for composition of a secure and conflict-free meta-policy is described in Chapter 3 of this dissertation.

2.3.2.2. Loosely-coupled collaboration

In a loosely-coupled collaborative environment, the interactions among domains are governed by the local policies of domains and do not require a mediated global meta-policy. A loosely-coupled multi-domain environment provides an alternate approach to achieve collaboration that is more flexible in terms of individual domains' participation and is more autonomous as compared to a federated collaborative system. In a loosely-coupled system, the collaborating domains provide interfacing mechanisms and ontology description to facilitate access to their resources without revealing their security and access control policies completely. The information about domains' shareable resources and interfaces through which such resources can be accessed are stored in a global directory service (GDS). (The GDS is similar to UDDI which is used in discovery of web services [39]).

Loosely-coupled system, despite providing a lower degree of interoperation than federated system, can be used to support tightly integrated and time-critical business processes and distributed applications requiring long-term collaboration among domains. Such collaborative applications are widely used in e-commerce, supply-chain management, health-care administration, and web services. The main challenge is that when the resource access requirements of distributed applications and the policies of collaborating domains are not aligned correctly due to lack of full knowledge, security and privacy of information is jeopardized. Accordingly, security assurance must be incorporated in the design of distributed applications from the onset, and such design must be verified for conformance with the information security and privacy policies of individual domains.

Due to the autonomous nature of domains and the absence of a global meta-policy in loosely-coupled environment, the conformance verification of a distributed application needs to be performed independently for each domain. A key aspect of this conformance verification is to identify domain-specific tasks from the application specifications and determine whether or not these tasks can be allowed to access domain resources according to their local policy constraints. In addition, such verification entails determining the satisfiability of inter-domain synchronization constraints for execution of the distributed application. Verification of such synchronization constraints becomes highly challenging when domains employ time-dependent policies.

To address the above-mentioned issues, we have proposed an approach for verifying secure composibility of distributed applications in an autonomous and loosely-coupled multi-domain environment. The approach is designed for distributed-services or workflow based applications that are invoked on a recurrent basis and requires interactions among a pre-selected set of collaborating domains. The overall verification process, depicted in the middle block of Fig. 2.2, consists of four steps: i) specification modeling of distributed workflows, ii) state-based representation of domains' time-dependent policies, iii) domain-specific workflow verification, and iv) inter-domain dependency verification.

Workflow specification modeling involves defining the interactions and information flow among collaborating domains using the interfaces provided by the GDS. The workflow specification model is also used to decompose a distributed workflow into domain-specific projected workflows. A projected workflow specifies the domain-specific tasks and their interdependencies which need to be verified against the respective domain's time-dependent access control policy. We use Generalized Temporal Role based Access Control (GTRBAC) model to specify the time dependent access control policies of domains. These GTRBAC policies are represented using time augmented finite state machines (FSMs) to capture the time-dependent authorizations for the underlying resources. The correctness of a domain-specific projected workflow is verified by finding traces in the FSM of each domain that support execution of domain-specific tasks under the given temporal constraints. After verification of projected workflows, the synchronization and dependency constraints amongst the workflow tasks performed by different collaborating domains are verified. The timing information computed in the projected workflow verification phase is used to determine an interleaving of projected workflow tasks that satisfies the synchronization constraints of the distributed workflow. This timing information is also used to determine a feasible schedule for the overall verified distributed workflow.

2.3.2.3. Ad-hoc collaboration

Ad-hoc collaborations are similar to loosely-coupled collaboration in terms of establishing interoperation based on the local policies of domains without considering any policy mediator. However, unlike loosely-coupled systems, domains involved in an ad-hoc collaboration are more autonomous in disassociating themselves from the collaborative system. Therefore, ad-hoc collaborations cannot be used to support time-critical and/or safety critical distributed applications that requiring extensive interoperation among a pre-selected set of domains. A large number of distributed applications including, web-services, peer-to-peer computing, and peer-to-peer filing sharing can be supported by ad-hoc collaborative system.

In an ad-hoc collaboration, a domain is only aware of a few other domains to which direct information sharing can be carried out. Two key issues need to be addressed to establish secure ad-hoc collaborations: i) discovery of domains for resource sharing and ii) establishing a secure mediator-free collaboration between mutually unknown domains. The discovery process involves finding all the cross-domain resources/services that can be accessed from a given domain. For ensuring secure and authorized access to the discovered cross-domain resources/services in a dynamically changing environment, appropriate authentication and authorization mechanisms need to be developed.

An approach that supports secure ad-hoc collaboration in a RBAC environment has been proposed in [119, 120]. The approach uses a novel role-based routing technique for discovering domains and determining authorization for cross-domain resource accesses. The authorizations for cross-domain resources are determined for individual users on a session basis. The approach relies on the access history of a user’s session to determine the authorizations of users for requested resource accesses. The access history is maintained in form of a sequence of roles accessed during the current session.

3. GLOBAL META-POLICY FOR SECURE INTEROPERATION IN FEDERATED ENVIRONMENT

In this chapter, we describe the proposed policy composition framework that integrates the access control policies of collaborating domains to facilitate secure information and resource sharing in a federated environment. For policy composition, we assume that the local policy of each domain is consistent and is specified using role based access control (RBAC) model. The proposed policy composition framework generates a secure and conflict-free meta-policy policy that governs all the inter-domain accesses. In the following, we provide a brief introduction to the overall process of policy composition and then discuss the details of each step involved in the generation of the global meta-policy.

3.1. Policy Composition

Composition of a global meta-policy governing interoperation among heterogeneous domains is a challenging task leading to various types of conflicts. These conflicts may arise because different domains may use different models, semantics, schema format, data labeling schemes, and constraints for representing their access control policies [26, 40, 106, 22, 61]. In this chapter, we mainly focus on the conflicts related to access control constraints. In particular, we consider constraint conflicts arising as a result of integrating RBAC policies of multiple domains. An example of access control constraint conflict in the context of RBAC policy integration is the introduction of cycles in domain-specific role-hierarchies as depicted in Fig. 3.1. Such cycles in role hierarchy enable junior roles to inherit the permission of senior roles leading to violation of domain specific security constraints [61]. In addition, the interplay of role hierarchy and SoD constraints may lead to other types of constraint conflicts which are described in

Section 3.4 of this chapter. These conflicts, if remain undetected and unresolved, expose the collaborating systems to numerous vulnerabilities and risks pertaining to the security and privacy of their data and resources.

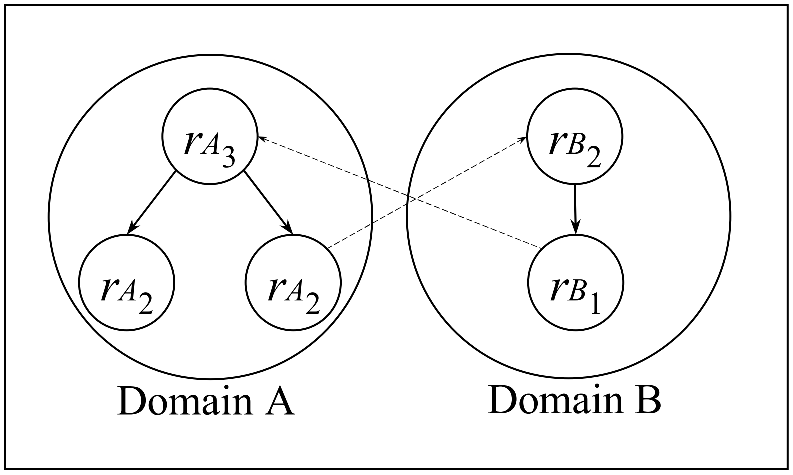


Fig. 3.1 An inconsistent meta-policy because of cycles in domain-specific hierarchies

The proposed policy composition framework generates a secure and conflict-free meta-policy policy in two steps as shown in Fig. 3.2. In the first step of policy composition, the RBAC policies of domains are merged by establishing role mapping across-domains. Such a mapping enables inter-domain information and resource sharing via the mapped roles. In addition to the automated generation of role mapping between cross-domain roles, the framework also allows security policy administrators to map cross-domain roles based on the interoperability requirements of collaborating domains. The resulting meta-policy may not be consistent and may not satisfy the security constraints of collaborating domains. In particular, three types of security violations, discussed in Section 3.4, may occur as a result of an inconsistent role-mapping. These include: *role-assignment violation*, *role-specific-SoD violation*, and *user-specific SoD violation*. These conflicts are resolved in the next step by removing some of the mapping links specified in the role-mapping step. Resolving policy conflicts in an arbitrary manner may significantly reduce interoperation in terms of data sharing and cross-domain accesses. The proposed policy integration framework uses an integer programming (IP) based approach for optimal resolution of meta-policy conflicts. The optimality criterion is to maximize information and data sharing via assumption of cross-domain roles.

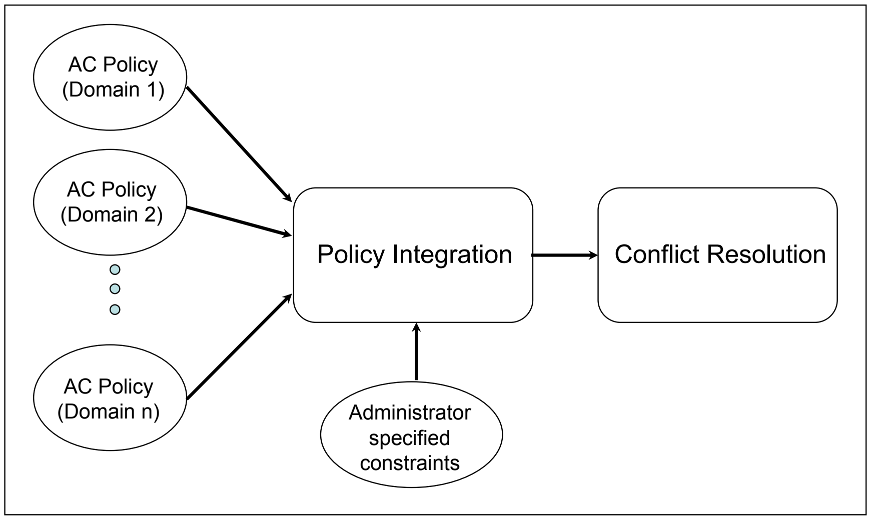


Fig. 3.2 Policy composition framework.

An important consideration in composing an optimal meta-policy policy is the preservation of domains' autonomy. Ideally, both security and autonomy of collaborating domains need to be preserved. However, satisfaction of both security and autonomy requirements may not be feasible. In almost every collaborative environment, violation of any domain's security constraints is not permissible. Domains may compromise their autonomy for establishing more interoperability provided the autonomy losses remain within the acceptable limits. The proposed IP-based approach for conflict resolution provides the flexibility of autonomy relaxation in favor of greater interoperability. Accordingly, in a collaborative environment in which certain autonomy violations can be tolerated, the objective of the conflict resolution phase is to generate a global meta-policy policy that maximizes inter-domain role accesses and keep the autonomy losses within the acceptable limits.

3.2. Role Based Access Control for Secure Interoperation

In the proposed policy composition framework, the access control policies of domains are specified using RBAC model with support for role hierarchies and separation of duties (SoD) constraints. We consider two types of role hierarchies including, *inheritance hierarchy* and *activation hierarchy*. The *inheritance hierarchy*, denoted by *I-hierarchy*, allows a user activating a senior role to inherit all permissions of junior roles without activating them. *Activation hierarchy*, denoted by *A-hierarchy,* does not support the permission inheritance semantics. In *A-hierarchy* semantics, any user assigned to a senior role is entitled to activate all its junior roles, and by activating a role a user is only authorized to acquire the permissions that are directly assigned to the activated role. The *A-hierarchy* semantics is incorporated in the RBAC model to support the *principle of least privilege*, which requires that a user be given no more privilege than necessary to perform a job. We use the symbols $\geq_I^*$ and $\geq_A^*$ to express I and A hierarchy relationship between two roles respectively. Accordingly, $r_i \geq_f^* r_j$, where $f \in \{I, A\}$, implies that role r_i is senior to r_j and the hierarchical relationship between them can be either *inheritance* only or *activation*. If role r_i is immediately senior to role r_j then the superscript * is omitted from the relation symbol $\geq_f^*$.

RBAC can be used to enforce SoD constraints to prevent possible fraud in organizations. We consider two types of SoDs namely: *role-specific SoD* and *user-specific SoD*. A *role-specific SoD* disallows assumption of conflicting roles by the same user. Similarly, a *user specific SoD* constraint prohibits conflicting users from assuming the same role simultaneously.

3.3. 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 Fig. 3.3, 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* and *A-hierarchy* are represented by *solid* and *dashed 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. 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).

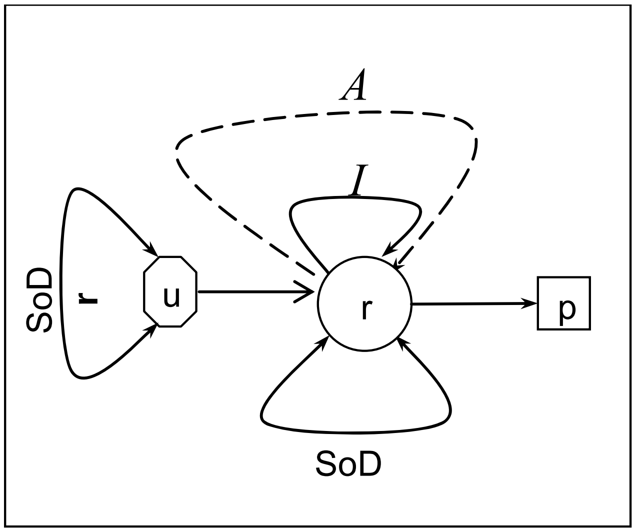


Fig. 3.3 RBAC type graph

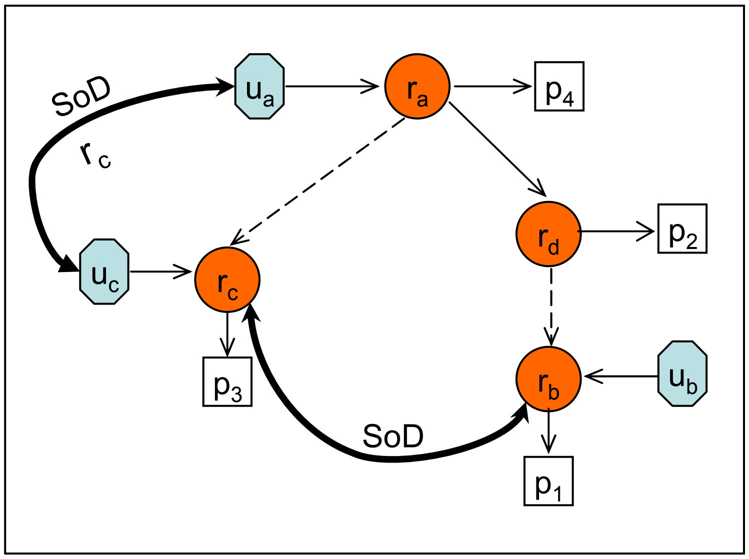


Fig. 3.4 An example of RBAC graph

Fig. 3.4 shows the graphical representation of an RBAC policy instance. The RBAC graph in Fig. 3.4 consists of four roles r_a, r_b, r_c and r_d, with $r_a \geq_A r_c$, $r_a \geq_I r_d$, and $r_d \geq_A r_b$. 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 SoD* constraint between role r_b and r_c, shown as a double headed arrow between these two roles in Fig. 3.4. Also users u_a and u_c are conflicting users for role r_c and are not allowed to access r_c simultaneously.

3.4. Security Requirements in a Multi-domain RBAC System

The goal of policy composition is to enable information and resource sharing without violating the security 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 RBAC policies. Additional security requirements of the multi-domain system can be specified by administrators with global security responsibility. The global security policy constructed from the domains' policies and administrator specified role mappings may be inconsistent and may violate the security constraints of constituent domains as well as of the multi-domain system.

We focus on three types of security policy violations:

1. violation of role assignment constraint.
2. violation of role-specific SoD constraint.
3. violation of user-specific SoD constraint.

Definition 3.1 (role assignment violation): *A global meta-policy causes a violation of role assignment constraint of domain k if it allows a user u of domain k to access a local role r even though u is not directly assigned to r or any of the roles that are senior to r in the role hierarchy of domain k.*

Definition 3.2 (role-specific SoD violation): *A global meta-policy causes a violation of role-specific SoD constraint of domain k if it allows a user to simultaneously access any two conflicting roles r_i and r_j of domain k in the same session or in concurrent sessions.*

Definition 3.3 (user-specific SoD violation): *Let U_r^c denote the conflicting set of users for role r belonging to domain k. A global meta-policy causes a violation of user-specific SoD constraint of domain k if it allows any two distinct users from the set U_r^c to access role r in concurrent sessions.*

Following example illustrate the three types of security violations defined above.

Example 3.1: Fig. 3.5 shows a meta-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 specified for 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 meta-policy shown in Fig. 3.5 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*.

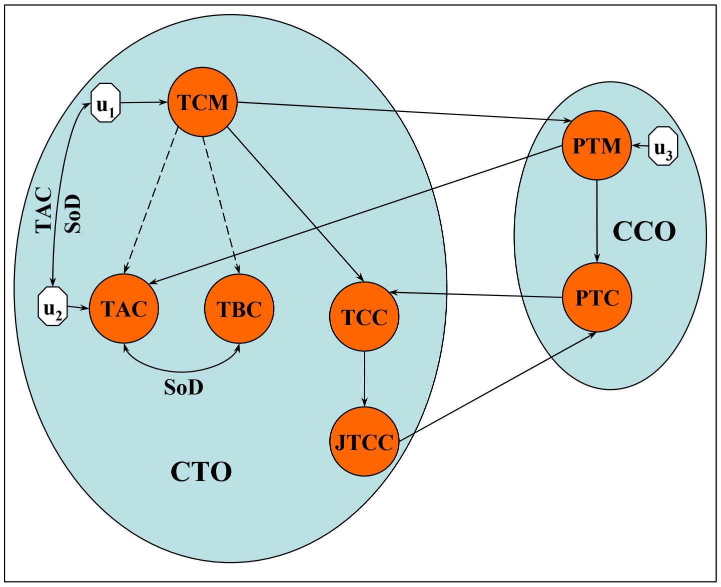


Fig. 3.5 A multi-domain access control policy defining interoperation between CTO and CCO

Example 3.1 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.

Example 3.2: Consider Corporate Audit Department that performs tax auditing of public companies for Internal Revenue Service (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 meta-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 Fig. 3.6.

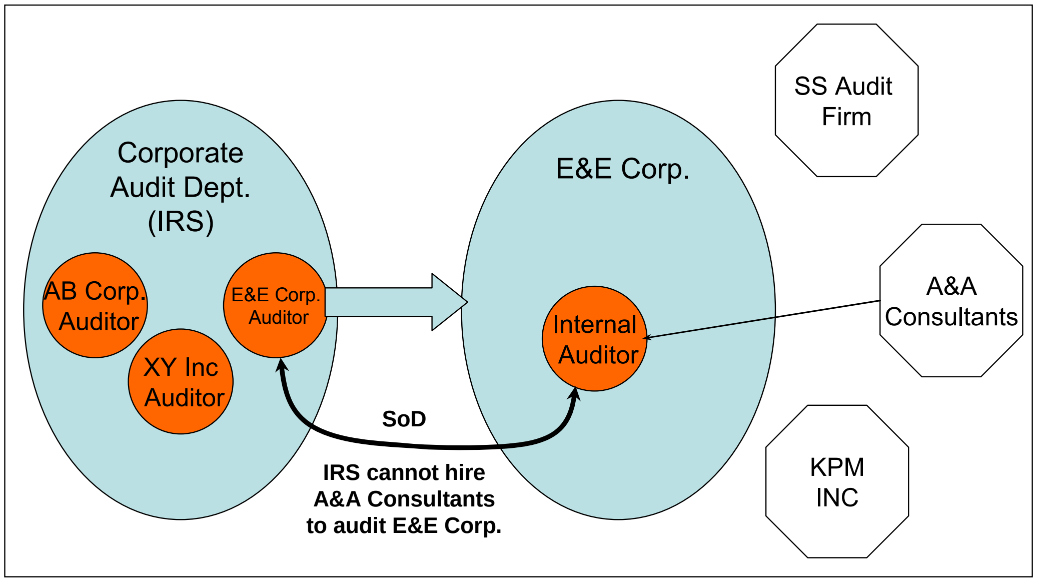


Fig. 3.6 Example of a cross-domain separation of duty (SoD) constraint

3.5. Information Sharing Policy

In the policy composition step of Fig. 3.2, domain policies are composed to form a global meta-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.*

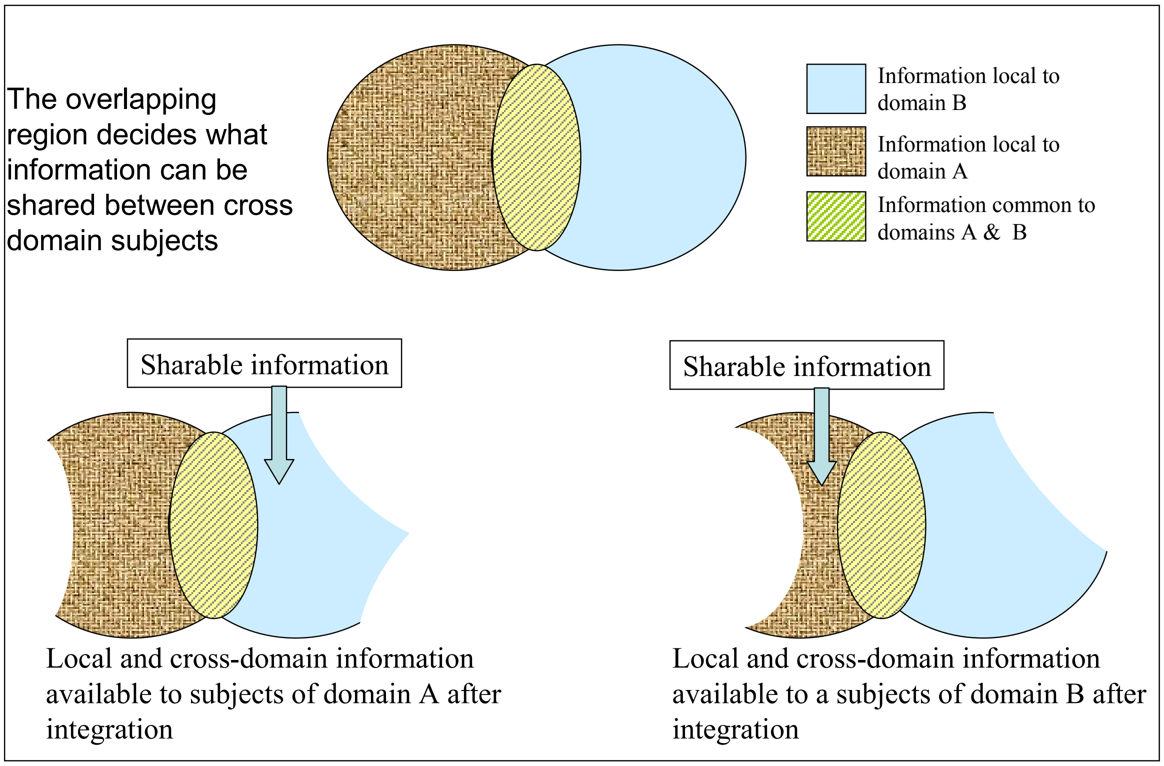

The overlapping region decides what information can be shared between cross domain subjects
Information local to domain B
Information local to domain A
Information common to domains A & B
Sharable information
Sharable information
Local and cross-domain information available to subjects of domain A after integration
Local and cross-domain information available to a subjects of domain B after integration

Fig. 3.7 An abstract view of inter-domain information sharing.

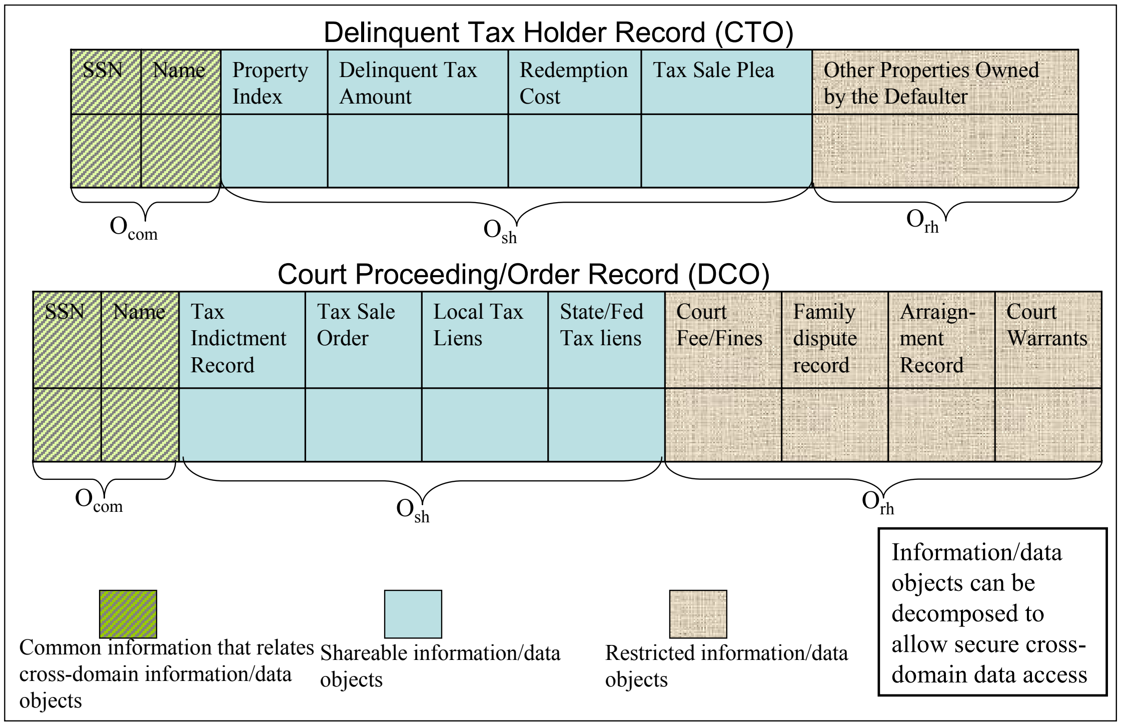

Delinquent Tax Holder Record (CTO)
SSN
Name
Property Index
Delinquent Tax Amount
Redemption Cost
Tax Sale Plea
Other Properties Owned by the Defaulter
O_{com}
O_{sh}
O_{rh}
Court Proceeding/Order Record (DCO)
SSN
Name
Tax Indictment Record
Tax Sale Order
Local Tax Liens
State/Fed Tax liens
Court Fee/Fines
Family dispute record
Arraign-ment Record
Court Warrants
O_{com}
O_{sh}
O_{rh}
Common information that relates cross-domain information/data objects
Shareable information/data objects
Restricted information/data objects
Information/data objects can be decomposed to allow secure cross-domain data access

Fig. 3.8 Information exchange between the CTO and DCO

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 merging module. Fig. 3.7 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. 3.8 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.

3.6. 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 [63, 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 [90].

Schema mismatch is another type of semantic heterogeneity that is characterized by representation conflicts, meta-model conflicts and meta-meta-model conflicts [104]. 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 [13, 118, 104] 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 [76]) 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 3.3: Consider the Senior Clerk (SC) and Junior Clerk (JC) roles of the City Clerk Office shown in Fig. 3.9(a). The hierarchical relationship between SC and JC is given by A-hierarchy, $SC \geq_A JC$, 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 3.9(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, $C \geq_I AC$. 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 Fig. 3.9 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.

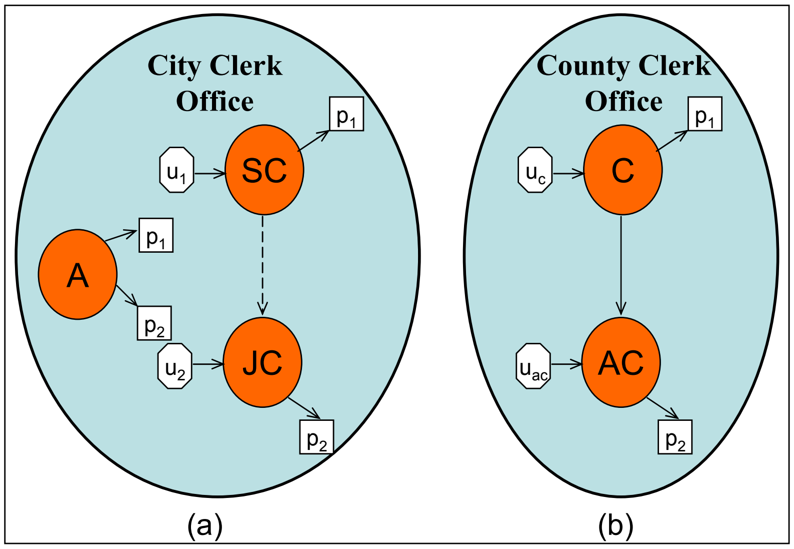


Fig. 3.9 Hierarchical heterogeneity

3.7. RBAC Policy Integration

In this section, we focus on the issue of composing a global meta-policy from the access control policies of collaborating domains. The global meta-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.

3.7.1. Policy integration requirements (PIR)

The following PIRs define the correctness semantics of a global meta-policy composed from the RBAC policies of collaborating domains. In order to be consistent with the RBAC semantics, we define the policy integration requirements using the graph based formalism described in Section 3.3.

1. *Element preservation*: Each element (role, user, permission) in the input RBAC graph should have a corresponding element in the multi-domain graph G.
2. *Relationship preservation*: Each relationship in the input graph should be preserved in the multi-domain graph G.
3. *User authorization preservation*: In the multi-domain graph G, for any user *u* of a domain *k*, the authorization set of *u* over the objects of domain *k* should not be different from the authorization set specified or implied in the input RBAC policy of domain *k*.
4. *Order independence*: The order in which policies are integrated should not influence the output of policy integration operation.
5. *Constraint satisfaction*: The multi-domain RBAC graph G must satisfy all the constraints of the input RBAC policies. In particular, no access that results in a violation of security constraints of collaborating domains can be enabled from the multi-domain RBAC graph. The security constraints in RBAC policy include role assignment, role hierarchy, and SoD constraints.

The first three PIRs are important in ensuring that the authorizations of users to local resources remain unaffected in the multi-domain environment and any modification in the domain policies as a result of interoperation remains transparent to the users. In particular, the access privileges of users to local resources and the access methods by which such privileges are acquired prior to interoperation should not be changed in the meta-policy. PIR 4 entails 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.

Table 3.1
Functions/predicates used for policy composition

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 $\left(p \in Pset_{assign}(r_1) \Rightarrow p \in Pset_{assign}(r_2) \right) \wedge \left((r_1 \underset{I}{\geq} r_k \wedge r_k \neq r_2) \Rightarrow r_2 \overset{*}{\underset{I}{\geq}} r_k \right)$ *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

PIR 5 defines the security requirements of the meta-policy in terms of the constraints of access control policies of collaborating domains. In the context of RBAC, the security constraints are defined with respect to user-role assignment, role hierarchy, and SoD constraints. All these security constraints of collaborating domains need to be preserved in the composed meta-policy.

3.7.2. Merging of RBAC policies

In this sub-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. In RBAC context, integration of access control policies involves defining a mapping between cross-domain roles. *A role mapping M_{AB} is a function that maps a role of domain A to a role of domain B ($M_{AB} : R_A \rightarrow R_B$).* By virtue of this role mapping, any user authorized for a role, say r_a, in domain A is allowed to access all the permissions of the *mapped* role, say r_b, in domain B ($M_{AB}(r_a) = r_b$).

We propose a policy merging algorithm, *RBAC-integrate*, that merges the RBAC policies of component domains by comparing and mapping cross-domain roles. The proposed policy merging algorithm finds an inter-domain role mapping based on the permission assignment and hierarchical ordering of corresponding roles. The permission assignment includes both directly assigned permissions as well as inherited permissions. A permission *p* is a pair *p*(*o*, *a*), where *o* is the object and *a* is the access mode. We 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 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 listed 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.
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:

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

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:

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

```
RBAC-integrate($G_1, G_2, \ldots, G_n$)
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 $r_c \in$ 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 $r_c \in$ 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 $\left(p \in Pset_{assign}(r_i) \Rightarrow p \in Pset_{assign}(r_j)\right) \wedge \left((r_i \underset{I}{\geq} r_k \wedge r_k \neq r_j) \Rightarrow r_j \underset{I}{\geq^*} r_k\right)$
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 $\left(\exists p \mid p \in Pset_{assign}(r_i) \Rightarrow p \in Pset_{assign}(r_j)\right) \vee \left(\exists r_k, r_m \mid r_i \underset{I}{\geq} r_k \Rightarrow (r_j \underset{I}{\geq} r_m \wedge already-linked(r_k, r_m)\right)$
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. 3.10 Policy merging algorithm

```
split(r, common-permissions, common-juniors)
1. $r_j \leftarrow$ createrole()
2. insert(r->childrenlist-I,$r_j$)
3. for each p $\in$ common-permissions
4.    do remove(r->plist, p)
5.       insert($r_j$->plist,p)
6. for each $r_c \in$ common-juniors
7.    do remove(r->childrenlist-I, $r_c$)
8.       insert($r_j$->childrenlist-I, r)
9.       return $r_j$

link($r_1$, $r_2$)
1. insert($r_1$->childrenlist-I,$r_2$)
2. insert($r_2$->childrenlist-I,$r_1$)
3. for each $r_i$ s.t. $(r_i = r_1 \vee r_i \underset{I}{\geq}^* r_1)$
4.    do for each $r_j$ s.t.
      $(r_j \in conf-rset(r_1)) \vee (r_j \underset{I}{\geq}^* r_c \wedge r_c \in conf-rset(r_1))$
            do conf-rset($r_i$)=conf-rset($r_i$)$\cup r_j$
5.             conf-rset($r_j$)=conf-rset($r_j$)$\cup r_i$
6. for each $r_i$ s.t. $(r_i = r_2 \vee r_i \underset{I}{\geq}^* r_2)$
7.    do for each $r_j$ s.t.
      $(r_j \in conf-rset(r_2)) \vee (r_j \underset{I}{\geq}^* r_c \wedge r_c \in conf-rset(r_2))$

8.          do conf-rset($r_i$)=conf-rset($r_i$)$\cup r_j$
9.             conf-rset($r_j$)=conf-rset($r_j$)$\cup r_i$
return

Remove-role($r_d$)
1.    $R_p \leftarrow R_p \cup \{r\}$, for all $r$ such that
2.    $R_c \leftarrow R_c \cup \{r\}$, for all $r$ such that
3.    for each $r_p \in R_p$
4.       for each $r_c \in R_c$
5.          If $\exists r'$ : $r' \neq r_d \wedge r_p \underset{I}{\geq} r' \wedge r' \underset{I}{\geq} r_c$
6.             continue
7.          insert($r_p$->childrenlist-I, $r_c$)
8.          remove($r_c$->parentlist-I, $r_d$)
9.          insert($r_p$->parentlist-I, $r_p$)
10.   for all $r_e$: $r_e \in$ equivalent($r_d$)
11.      remove($r_e$->parentlist-I, $r_d$)
12.      remove($r_e$->childrenlist-I, $r_d$)
13.   for all $r_s$: $r_d \in$ conf-rset($r_s$)
14.      remove($r_s$->conf-rset, $r_d$)
15.   for each $r_p \in R_p$
16.      for each $p \in$ Pset($r_d$)
17.         insert($r_p$->Pset, $p$)
18.      deallocate($r_d$)
```

Fig. 3.11 Procedures used by Role-Integrate during policy integration.

3.7.3. Policy merging algorithm

Fig. 3.10 shows the proposed policy merging algorithm, *RBAC-integrate*, that merges RBAC policies of *n* domains to produce a global meta-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 merged RBAC policy obtained in previous iteration. After *n*-1 iterations, the RBAC policies of all *n* domains are merged to produce a global meta-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 3.7.4, are roles that do not have any permissions assigned to them nor can any user activate them. Removal of redundant roles is essential to satisfy the order independence (PIR 4) requirement in the merged policy.

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 by defining a bidirectional role mapping between r_1 and r_2, *i.e.*, $r_1 \geq_I r_2$ and $r_2 \geq_I r_1$. A role mapping $r_1 \geq_I r_2$ is represented in the RBAC graph by an *I-hierarchy* edge from r_1 to r_2. Linking two inter-domain roles r_1 and r_2 through a bidirectional role mapping 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 Fig. 3.11) for bidirectional mapping cross-domain equivalent roles r_1 and r_2. In addition, conflicting role sets of r_1 and r_2 and all their senior roles that have an inheritance 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 in the *link* function. 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 [56]. 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 3.8.3.

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 Fig. 3.12 which shows how mapping inter-domain roles change the conflicting set of linked roles. Fig. 3.12(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 Fig. 3.12(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 through bidirectional mapping. Fig. 3.12(b) shows the merging of RBAC graph of Fig. 3.12(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 Fig. 3.12(d) which have a common ancestor role r_1 in the activation hierarchy semantics. The meta-policy shown in Fig. 3.12(b) is conflicting and can be made consistent by removing one of the role mappings $r_2 - r_4$ 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 Fig. 3.11. 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 through a bidirectional mapping i.e., $r_1 \geq_I r_{2j}$ and $r_{2j} \geq_I r_1$. 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, a bidirectional mapping ($r_{1j} \geq_I r_{2j}$ and $r_{2j} \geq_I r_{1j}$) is defined between r_{1j} and r_{2j}.

In Section 3.9, we provide an example of a global meta-policy generated by merging the access control policies of various county offices including *County Clerk Office* (CCO), *County Treasurer Office* (CTO), and *County Attorney Office* (CAO). These county offices collaborate with each other for collection and sale of real-estate taxes on property parcels located within the jurisdiction of the concerned county. Fig. 3.16(a – c) show the graphical representation of RBAC policies of CCO, CTO, and CAO domains prior to merging and Fig. 3.16 (d – e) depict the RBAC graph of these domains after defining cross-domain role mappings.

Note that some of the roles in Fig. 3.16 are split into two or more roles with their permissions redistributed among the newly created junior roles. For instance, the DTM role in Fig. 3.16(a) gets split into three roles DTM, DTM_{10} and DTM_{12} with DTM as the senior of the remaining two (shown in Fig. 3.16(d)). 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 [77] 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.

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 A.

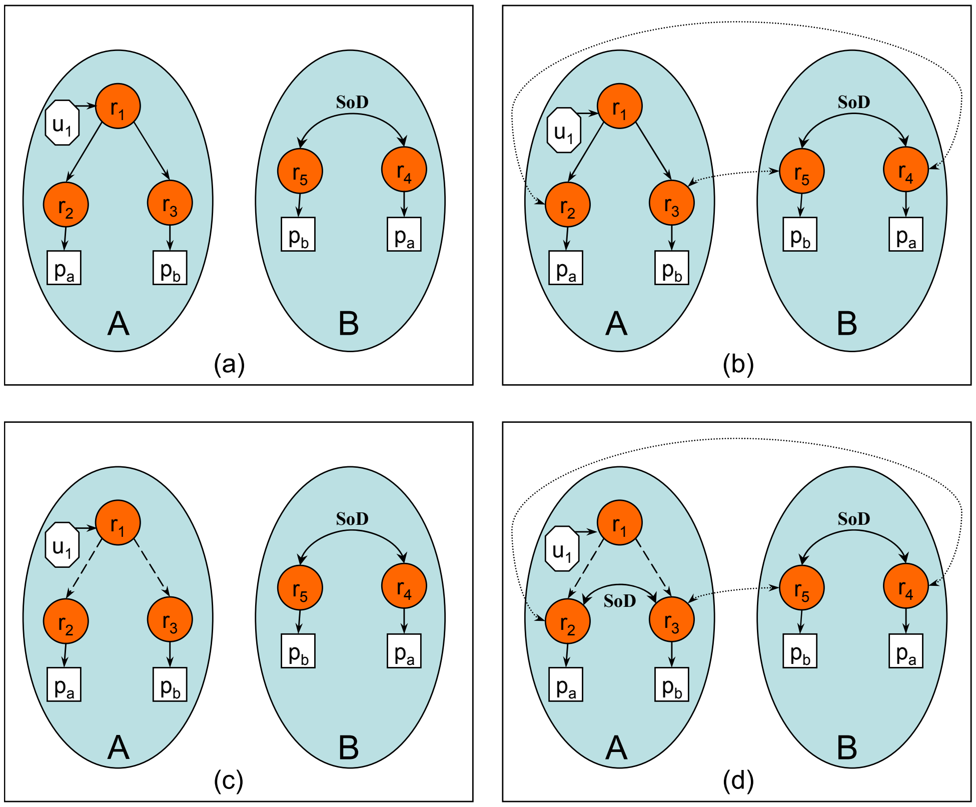


Fig. 3.12 Example of induced SoD

3.7.4. 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 3.7.1. *RBAC-integrate* satisfies all the policy integration requirements (PIRs) except PIR5. Since conflict resolution is not included in RBAC-integrate, therefore the resulting meta-policy may not satisfy all the constraints of input RBAC policies. However, the meta-policy obtained after conflict resolution, discussed in Section 3.8, satisfies all the integration requirements. Theorems 3.1, 3.2, and 3.3 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 5.

Definition 3.4: 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 Fig. 3.11. 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 3.2: 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 Fig. 3.11. The following properties hold with respect to G and G':

- *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.*
- *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-rset (r_x) before the removal of r_d, then $r_y \in$ conf-rset (r_x) after the removal of r_d.*
- *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.*
- *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.*

- *For any two roles $r_x \neq r_d$ and $r_y \neq r_d$, if $r_y \in$ conf-rset (r_x) before the removal of r_d, then $r_y \in$ conf-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 3.2 is given in Appendix A.

Lemma 3.3: The meta-policy produced by *RBAC-integrate* satisfies PIRs 1 – 3.

Proof of Lemma 3.3 is given in Appendix A.

One key requirement in composing a meta-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 3.1 (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 through bidirectional mapping 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 $r_A \geq_I r_{Aj}$, 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 3.2 (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 [104]. To show that two final integrated policy models X and Y are isomorphic, we define a morphism φ(X→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 $R(\varphi(U)) \in R_Y$ (R_X and R_Y are relations in X and Y respectively).*

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

Theorems 3.1 and 3.2 imply that the meta-policy composed by *RBAC-integrate* is independent of the order in which domain policies are integrated.

3.7.5. Time complexity of RBAC-integrate

The algorithm *RBAC-integrate* runs in polynomial time, as evident from the following two Lemmas.

Lemma 3.4: 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 3.5: 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 $|R1| \times |R2|$.

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

3.8. Optimal Conflict Resolution

The policy merging algorithm described above takes as input the RBAC policies of the domains and composes a meta-policy which allows inter-domain role accesses and is homogeneous in terms of role hierarchies and permission assignments. However, the meta-policy created in this phase may be inconsistent and may not completely satisfy the collaborating domains' security requirements. Moreover, security administrator(s), in charge of the global security policy, can define additional inter-domain accesses in form of role mappings. These administrator-specified role mappings may also conflict with the access control policies of individual domains. For instance, in Fig. 3.16(d, e), mapping the role *LSO* of *CCO* domain to the role *DTA* of *CTO* domain violates the role-specific SoD constraint between roles *DTA* and DTM_{10} of *CTO*. This role mapping enables user u_6 to access role *DTA* through the role *LSO*. Also the bidirectional role mapping between $R10_{11}$ and DTM_{10} allows user u_6 to access role DTM_{10} through $R10_{11}$. This is a violation of the SoD constraint defined between roles *DTA* and *DTM* in the original RBAC policy of CTO domain shown in Fig. 3.16(a).

The solution to this problem is to remove one of the following role mappings: i) $LSO{:}CCO \underset{I}{\geq} DTA{:}CTO$, ii) $R10_{11}{:}CCO \underset{I}{\geq} DTM_{10}{:}CTO$. This raises an important question: which role mapping from the set of conflicting mappings should be removed so that the security and autonomy constraints of collaborating domains are not violated?

Although, removal of cross-domain role mappings resolves conflicts in the given meta-policy, it also changes the set of allowable accesses and an arbitrary selection of removable role mappings may significantly reduce interoperation. A conflict resolution mechanism is needed that resolves policy conflicts among the collaborating domains in an optimal manner. The problem of conflict resolution in a given meta-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 maximum data sharing, maximum prioritized accesses, and minimum representation [61] can be used.

3.8.1. IP formulation of a multi-domain RBAC policy

In the following, we describe an approach for formulating the meta-policy integration problem into an integer program (IP) [131]. 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 meta-policy, all the constraints such as role-assignment, 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 role mapping edge that lies in the path between the user node u_i and role node r_j.

3.8.1.1. Constraint transformation rules

In the following, we list the transformation rules to generate IP constraint equations for an RBAC meta-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 $domain(u_i) \neq domain(r_j)$ 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. At least one user from the set A_u must be able to access role r_j. Formally, $\sum_{u_i \in A_u} u_{ir_j} > 0$.
4. Let $u_{ir_j} = 1$ and a role r_k exists such that $domain(r_j) = domain(r_k)$ and $r_j \underset{I}{\geq} r_k$, 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 $domain(u_i) \neq domain(r_k)$. 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 $r_m \underset{I}{\geq} r_k$. We define two roles sets R_c and R_{pc} as follows:

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

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

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

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 $r_m \underset{I}{\geq} r_k$.

ii. u_i is authorized for role r_n and there is an inter-domain role mapping 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 $u_{ir_n} \in \mathrm{R_c}$

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 role mapping exists from role r_k to role r_l. If user u_i is able to access r_l through the cross domain mapping link (r_k, r_l), then user u_j, if authorized for role r_k, can also access r_l through the mapping link (r_k, r_l). Formally:

if *domain*(u_i) = *domain*(u_j) = *domain*(r_k) 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 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:

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

3.8.2. Optimality criteria and weight assignment

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.

$$\begin{aligned} &\text{maximize} \quad c^T u_r \\ &\text{Subject to} \quad Au_r \leq b \\ &\qquad\qquad\quad \forall u_{ir_j} \in u_r, u_{ir_j} = 0 \text{ or } 1 \end{aligned}$$

where, A is the constraint matrix and c is a vector defining the optimality criteria in terms of the weight of the decision variables corresponding to user-role authorizations. The main purpose of formulating the meta-policy into an IP problem is to find a feasible solution (a set of users to role authorization) that maximizes the objective function according to the given optimality criterion without violating the security constraints of underlying domains. Various optimality measures such as maximum data sharing and maximum prioritized accesses can be used. Maximum data sharing does not consider any priority among the inter-domain accesses and involves maximizing the overall inter-domain accessibility. Maximum data sharing can be specified in the objective function as a sum of all decision variables representing inter-domain user to role accesses, i.e., all c_is corresponding to the cross-domain user-role variables are assigned a value of '1' and the remaining c_is are set to '0'.

In some cases, certain cross-domain accesses have a higher priority than the others. Therefore, such accesses need to be assigned a higher weight for increasing their chances of retention in the final policy. The weight of a given cross-domain access is defined relative to the weights of conflicting accesses that can be removed in favor of the given access. We assume that domains may specify their preference for retention of some of the cross-domain accesses by indicating which accesses should supersede conflicting accesses. Based on this priority specification, the weights of the corresponding user-role access variables in the objective function are determined. For instance, consider the following four conflicting cross-domain accesses represented by the following user-role variables: u_{r1}, u_{r2}, u_{r3}, and u_{r4}. Let c_1, c_2, c_3 and c_4 respectively denote their weights. Suppose the following rules specify the relative priorities of these accesses: i) u_{r1} supersedes the individual accesses u_{r2}, u_{r3}, and u_{r4}, implying that either u_{r2} or u_{r3} or u_{r4} can be removed in favor of $u_{r1.}$ ii) u_{r1} also supersedes $u_{r2} + u_{r3}$, implying that if there is a choice of retaining the single cross-domain access u_{r1} or two cross-domain accesses u_{r2} and u_{r3}, then u_{r1} is retained and both u_{r2} and u_{r3} are removed. iii) $u_{r2} + u_{r4}$ and $u_{r3} + u_{r4}$ supersede the cross-domain access $u_{r1,}$ implying that the single cross-domain access u_{r1} can be removed in favor of joint accesses u_{r2} and u_{r4} or u_{r3} and u_{r4}. iv) u_{r4} supersedes the individual accesses u_{r2} and u_{r3}. The weight assignment corresponding to this priority specification is given by: $c_4 > \max\{c_2, c_3\}$ and $\max\{c_4, c_2 + c_3\} < c_1 < (c_2 + c_3 + c_3)$.

It can be noticed that changing the weights of decision variables impact the degree of interoperability and autonomy of individual domains. In Section 3.9, we explain that a trade-off exist between the two metrics which depends on weight selection.

3.8.3. Autonomy consideration

One key requirement of policy composition 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 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 cross-domain roles r_c and r_d for which the following hold:

a. $domain(r_c) \neq domain(r_a) = domain(r_b)$

b. $domain(r_d) \neq domain(r_a) = domain(r_b)$

c. $conf-rset(r_c, r_d) \wedge \left[(r_a \underset{I}{\geq} r_c \wedge r_b \underset{I}{\geq} r_d) \quad \vee \quad (r_b \underset{I}{\geq} r_c \wedge r_a \underset{I}{\geq} r_d) \right]$

Fig. 3.12(b) 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 A, shown in Fig. 3.12(a), 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 concurrently in the multi-domain system.

Asymmetric cardinality of mapped 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 [78, 79]. 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. For a consistent RBAC policy, the cardinality of a senior role should not be greater than the cardinality of any of the junior roles that are related to the senior role in the *I*-hierarchy semantics [56]. Accordingly, a role mapping relation r_a:A $\geq_I$ r_b:B between the cross domain roles r_a and r_b of domains A and B respectively, becomes inconsistent if the cardinality of r_a is greater than the cardinality of r_b. In order to avoid an inconsistent role mapping due to asymmetric cardinalities of mapped roles, the cardinality of the senior role in the mapping relation is reduced to the cardinality of the junior role. For instance, in the mapping relation r_a:A $\geq_I$ r_b:B, if r_a has a cardinality constraint of three and r_b has a cardinality constraint of one, then the cardinality of r_a is decreased to one to ensure a consistent mapping. This reduction in the role cardinality of r_a can be considered as a violation of domain A's autonomy as the number of concurrent accesses of r_a allowed in the original RBAC policy of domain A are not permitted under this meta-policy. On the other hand, retaining the original cardinalities of interoperable roles may lead to security violations. 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. Fig. 3.18 depicts the trade-off between interoperability and autonomy in a graphical manner. A discussion of this graph is presented in Section 3.9.

In general, composition of a global meta-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. However, domains may be willing to compromise their autonomy for the sake of establishing more interoperability provided the autonomy losses remain within the acceptable limits. In the following, we describe how this autonomy relaxation condition can be incorporated as a constraint in the IP problem.

Let L_A denote the set of all cross-domain role mappings that either reduces role cardinalities of domain A or adds induced SoD constraints between roles of domain A. The overall autonomy loss of domain A caused by L_A is given by:

$$AL(L_A) = \frac{\begin{pmatrix}\text{Total number of local accesses without} \\ \text{any cross-domain role mapping link}\end{pmatrix} - \begin{pmatrix}\text{Total number of local accesses in} \\ \text{presence of all role-mapping links in } L_A\end{pmatrix}}{\begin{pmatrix}\text{Total number of local accesses without} \\ \text{any cross-domain role mapping link}\end{pmatrix}}.$$

The above expression can also be used for computing autonomy losses of a domain caused by individual role-mappings. However, the aggregate of all link level autonomy losses may be greater than the overall autonomy loss of a domain, *i.e.*, $AL(L_A) \le \sum_{l \in L_A} AL(\{l\})$. The reason for this discrepancy is that some common local accesses may be reduced by multiple cross-domain role-mapping links; therefore, reduction of these accesses is considered multiple times in the link-level aggregate. Based on the commonality of reduction of local accesses, we define a set S_i ($S_i \subseteq L_A$) for every cross-domain role mapping link l_i such that all local accesses of domain A reduced by l_i are also reduced by each role mapping link $l_k \in S_i$. In order to keep the autonomy losses of a domain within a certain threshold value, say α, the following autonomy constraint can be added in the IP problem:

$$\sum_{l_i \in L_A} \left(\prod_{l_k \in S_i} (1 - u_{rk}) \right) AL(\{l_i\}) u_{ri} + \sum_{\substack{(l_p, l_q \in L_A) \wedge \\ ind-sod(l_p, l_q)}} AL(\{l_p, l_q\}) u_{rp} u_{rq} \le \alpha .$$

The first sum in the above constraint captures the autonomy losses due to role cardinality reduction. The decision variable u_{ri} (u_{rk}) corresponds to the retention of cross-domain role mapping link l_i (l_k), i.e. the link l_i (l_k) is retained in the final policy if u_{ri} =1 (u_{rk} =1). The term $[\prod(1- u_{rk})]\mathrm{AL}(\{l_i\})\, u_{ri}$ implies that the role mapping link l_i causes an autonomy loss of $AL(\{l_i\})$ if no other role mapping link in the set S_i is retained in the final policy. If a role mapping link $l_k \in S_i$ is retained, then the autonomy loss due to l_i is not considered in computing the overall autonomy loss, because all the local accesses reduced by l_i are also reduced by l_k, implying that the autonomy loss due to the link l_i is covered by the autonomy loss due to link l_k. The second sum $\sum AL(\{l_p, l_q\})u_{r_p}u_{r_q}$ in the above constraint captures the autonomy loss caused by all role mapping pairs which results in addition of induced SoD constraints in domain A. The binary predicate *ind-sod* holds for any two cross-domain mappings l_p, and l_q if their retention in the final policy requires addition of induced SoD constraint.

The following example illustrates the formulation of IP constraints including the autonomy relaxation constraint for the multi-domain RBAC policy of Fig. 3.14.

Example 3.4: Consider two collaborating domains A and B with their respective RBAC policies shown in Fig. 3.13(a). The multi-domain RBAC policy that allows inter-domain accesses between A and B is shown in Fig. 3.13(b). The bidirectional role mapping between r_3 and r_5 and the administrator-specified mapping r_3:A $\geq_I$ r_5:B that allows role r_5 to inherit permission of role r_1 makes this meta-policy inconsistent. These two role mappings enable user u_3, assigned to the junior role r_3, to assume the senior role r_1, which is a violation of role-assignment constraint. This conflict can be resolved by either removing the role mapping r_3:A $\geq_I$ r_5:B or r_5:B $\geq_I$ r_1:A. In both cases the number of cross-domain accesses will remain the same. Note that the *SoD* constraint between r_2 and r_3 is an *induced SoD* constraint. This SoD constraint is caused by the role mappings r_3:A $\geq_I$ r_5:B and r_2:A $\geq_I$ r_4:B, and reduces local accesses of domain A from six to five, causing an autonomy loss of 16.67%. Suppose the maximum autonomy loss allowed by domain A is 10%. This autonomy relaxation constraint can be specified as: $16.67(u_{3r5}\, u_{2r4}) \leq 10$, where u_{3r5} = 1 (u_{2r4} = 1) implies that the role mapping r_3:A $\geq_I$ r_5:B (r_2:A $\underset{I}{\geq}$ r_4:B) is retained in the final meta-policy. The IP formulation of the meta-policy of Fig. 3.13(b) is shown in Fig. 3.13. Note that in the objective function, all the decision variables representing cross-domain role accesses are assigned a weight of one, implying that the optimality criterion is to maximize all cross-domain role accesses. An optimal solution to the IP problem shown in Fig. 3.13, has following values of cross-domain variable: 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, and u_{5r6}=1. Since u_{3r_3}=1 (constraint c9 in Fig. 3.13), and u_{3r_5}=0, the cross-domain role mapping $r_3 \underset{I}{\geq} r_5$ needs to be removed from the multi-domain RBAC graph of Fig. 3.13(b). Removal of the role mapping $r_3 \geq_I r_5$ also invalidates the induced SoD constraint between r_2 and r_3. Thus, the resulting meta-policy does not cause any autonomy loss of domain A.

3.8.4. Conflict resolution algorithm

Fig. 3.15 shows an algorithm *ConfRes* for resolving conflicts from the RBAC graph G representing the meta-policy. This algorithm first transforms the RBAC policy constraints into IP constraints using the rules given in Section 3.8.1**.** 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 meta-policy graph G is modified by removing the conflicting cross-domain edges and induced SoD constraints. The resulting graph defines the meta-policy that satisfies the security requirements of all collaborating domains. This is formally proved in Section 3.10.

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

Autonomy Relaxation constraint

c43: $16.67(u_{3r_5}u_{2r_4})\leq 10$

Fig. 3.13 IP formulation of multidomain RBAC policy shown in Fig.3.14

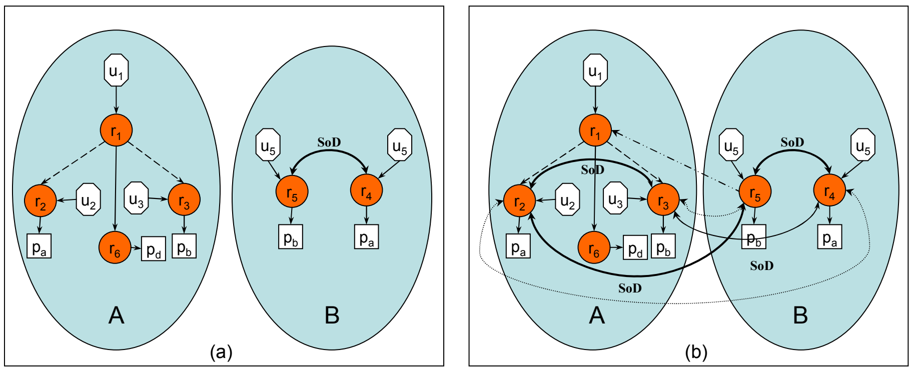


Fig. 3.14 (a) RBAC policy graph of domain A and B in example 4, (b) Integrated RBAC policy defining interoperation between domains A and B.

```
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 $r_k \underset{I}{\geq} r_i$.
2.  Assign a dummy user $u_{dj}$ to all roles $r_j$ for 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:
      $\exists\, u_i \in U$, $r_l \in R$ such that $[(r_l \underset{A}{\geq^*} r_j) \wedge conf\text{-}user(u_i, u_k, r_l) \wedge (u_i \in conf\text{-}user(r_j))]$
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.
    $G = G - \{(r_i, r_j) \mid \exists u_k \in U$ such that $u_{kr_i} = 1$ and $u_{kr_j} = 0$ 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. 3.15 Conflict resolution algorithm

3.9. An Illustrative Example

In this section, we illustrate the proposed policy composition 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 3.2 lists the roles, job description and permissions associated with each role of all three county offices. The permission authorization in Table 3.2 defines the access rights or permissions available to the corresponding roles on local as well as cross-domain information objects. As mentioned in Section 3.5, 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 3.3 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 3.3, 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. Identifying the prospective cross-domain roles that can access a given object 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 map them accordingly.

The RBAC policy graphs of the county offices (CTO, CCO, and CAO) prior to role mapping are shown in Fig. 3.16 (a), (b), and (c). Fig. 3.16 (d), (e), and (f) depict the policy graphs of these county offices after mapping cross-domain roles. The proposed role-mapping algorithm RBAC-integrate generates a bidirectional mapping between cross-domain roles that are equivalent in their permission assignment and have similar role hierarchy. In addition, the global security policy administrator(s) may also define cross-domain role mapping for specifying interoperation requirements. In Fig. 3.16 (d – f) the edge from a local role to a foreign role defines the cross-domain role mapping. A local role, in Fig. 3.16 (d – f) is shown as a shaded oval with solid outline, whereas a foreign role is depicted with a dashed-outlined oval. The annotations within the dashed oval describe both the names and domains of the foreign roles to which a local role is mapped. For instance, in Fig. 3.16 (d) the dashed oval with annotations PLAT_{09}:CAO and ACAT:CAO represent two foreign roles PLAT_{09} and ACAT of domain CAO. A local role DTM of Domain CTO is mapped to both PLAT_{09} and ACAT as shown by the edge from DTM to the corresponding foreign roles PLAT_{09}:CAO and ACAT:CAO. The mapping from DTM to ACAT:CAO is an administrator-specified mapping as indicated by the annotation 'admin'. The role mapping defines inheritance relationship between cross-domain roles. For instance, in Fig. 3.16(d) the role mapping from DTM:CTO to ACAT:CAO ($\text{DTM:CTO} \underset{I}{\geq} \text{ACAT:CAO}$) implies that a user, say u_1, authorized for the local role DTM can inherit the permissions of a foreign role ACAT of domain CAO through DTM. 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 ACAT without gaining access to role DTM.

Table 3.2
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 3.3
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

The role mappings shown in Fig. 3.16 (d – f) represents an inconsistent meta-policy and do not satisfy the security requirements of the collaborating county offices. For instance, the administrator-specified mappings $TA:CTO \underset{I}{\geq} TAO:CCO$ (Fig. 3.16(d)) and $PIO:CCO \underset{I}{\geq}$ TRA:CTO (Fig. 3.16(e)) causes a violation of role-specific SoD constraint defined between roles TRE and TRA of CTO domain. These mappings allow user u_2 to access role TRA via the cross-domain path TA:CTO $\underset{I}{\geq}$ TAO:CCO $\underset{I}{\geq}$ PIO:CCO $\underset{I}{\geq}$ TRA:CTO. Moreover, u_2 by accessing the local role TA inherits the permission of TRE because of the intra-domain relationship TA $\underset{I}{\geq}$ TRE. As a result, u_2 by accessing role TA inherits the permission of conflicting roles TRE and TRA. Similarly the role mapping DTLO:CCO $\underset{I}{\geq}$ DTC:CTO, LSO:CCO $\underset{I}{\geq}$ DTA:DTC, and $R10_{11}$:CCO $\underset{I}{\geq}$ DTM_{10}:CTO (Fig. 3.16(e)) enables user u_6 to access conflicting roles DTA and DTM_{10} of CTO domain (Fig. 3.16(d)). Note that in the original RBAC policy of CTO, an *SoD* constraint is defined between DTA and DTM (Fig. 3.16(a)). Since DTM splits into roles DTM_{10} and DTM_{12}, therefore these roles also conflict with DTA as shown in Fig. 3.16(d). Another violation of role-specific SoD constraint between roles DTM and DTA of CTO domain occur because of the role mappings DTM:CTO $\underset{I}{\geq}$ ACAT:CAO (Fig. 3.16(d)), ACAT:CAO $\underset{I}{\geq}$ TAC:CCO (Fig. 3.16(f)), and TAC:CCO $\underset{I}{\geq}$ DTA:CTO (Fig. 3.16(e)). These cross-domain role mappings enable u_1 to access the conflicting roles DTM and DTA of domain CTO. A *role-assignment* violation occurs because of cyclic hierarchy created by the mappings DTA:CTO $\underset{I}{\geq}$ ACAT:CAO (Fig. 3.16(d)), $PLAT_{09}$:CAO $\underset{I}{\geq}$ DTM:CTO and the intra-domain hierarchy constraint ACAT $\underset{I}{\geq}$ PLAT $\underset{I}{\geq}$ $PLAT_{09}$ of CAO domain (Fig. 3.16f)). This cycle in role hierarchy allows user u_4 assigned to role DTA to access the permissions of the senior role DTM. The security vulnerabilities caused by the role-mappings of Fig. 3.16(d), (e), and (f) are tabulated in Fig. 3.17.

Conflicts in the meta-policy shown in Fig. 3.16 (d – f) 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 meta-policy of Fig. 3.16. The resulting IP problem is solved with the objective of maximizing all cross-domain accesses. The solution thus obtained removes the following cross-domain role mappings from the meta-policy graphs of Fig. 3.16 (d – f): DTM:CTO $\underset{I}{\geq}$ ACAT:CAO, TAC:CCO $\underset{I}{\geq}$ DTA:CTO, DTA:CTO $\underset{I}{\geq}$ ACAT:CAO, PIO:CCO $\underset{I}{\geq}$ TRA:CTO, and LSO:CCO $\underset{I}{\geq}$ DTA:CTO. A maximum of 102 cross-domain accesses are obtained if the above role mappings 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 meta-policy as discussed in 3.8.2. However the total number of accesses cannot exceed the maximum value obtained by assigning uniform weights to all cross-domain accesses.

Fig. 3.18(a-b) shows the trade-off between interoperability and autonomy for the domains CTO and CCO respectively. For this study/analysis, interoperability of a domain is defined as a measure of the number of cross-domain accesses to that domain. The autonomy losses of domains CTO and CCO for the given meta-policy with cross-domain links L_{CTO} and L_{CCO} are determined using the AL expression given in Section 3.8.3. In the interoperability versus autonomy loss graph, depicted in Fig. 3.18, the acceptable limit for autonomy loss for both domains is set to 50% and the level of interoperability is varied by varying the weights of decision variables in the objective function. The maximum interoperability occurs when all cross-domain accesses have a uniform weight.

The graph shown in Fig.3.18 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 role-mappings. However, all the autonomy loss values are confined to the region bounded by the upper bound and lower bound curves shown in Fig. 3.18.

It can be noticed that the trade-off between the level of interoperation and degree of autonomy depends on the selection of weights in the objective function. Weight selection is an important issue and depends on the type of application. For example, in digital government application, achieving a high degree of interoperability among government agencies is preferable than maintaining autonomy of individual domains. In this case, uniform weights can be selected for maximizing interoperability. On the contrary, for collaborations requiring higher degree of autonomy for participating domains such as health-care applications, higher weights can be assigned to those cross-domain access variables that do not cause any autonomy loss. In addition, an upper bound on the autonomy loss can be specified as an additional constraint in the IP problem formulation. In summary, weight selection is an open research issue requiring further exploration.

3.10. Verification of Meta-policy

In this section, we formally analyze the proposed policy integration mechanism with respect to the five policy integration requirements (PIRs) discussed in Section 3.7.1. The PIRs define the correctness criteria for verifying the consistency of meta-policy. The first four PIRs state the conformance requirements for the meta-policy in terms of authorization preservation, relationship preservation, and order independence. The last PIR stipulates the security aspect of meta-policy. The meta-policy generated by the proposed policy composition framework satisfies all the above integration requirements. To prove this claim, we first analyze the compliance of proposed framework with respect to non-security PIRs (PIR 1 – 4) and then assess the correctness of the composed meta-policy with respect to the security constraints of collaborating domains.

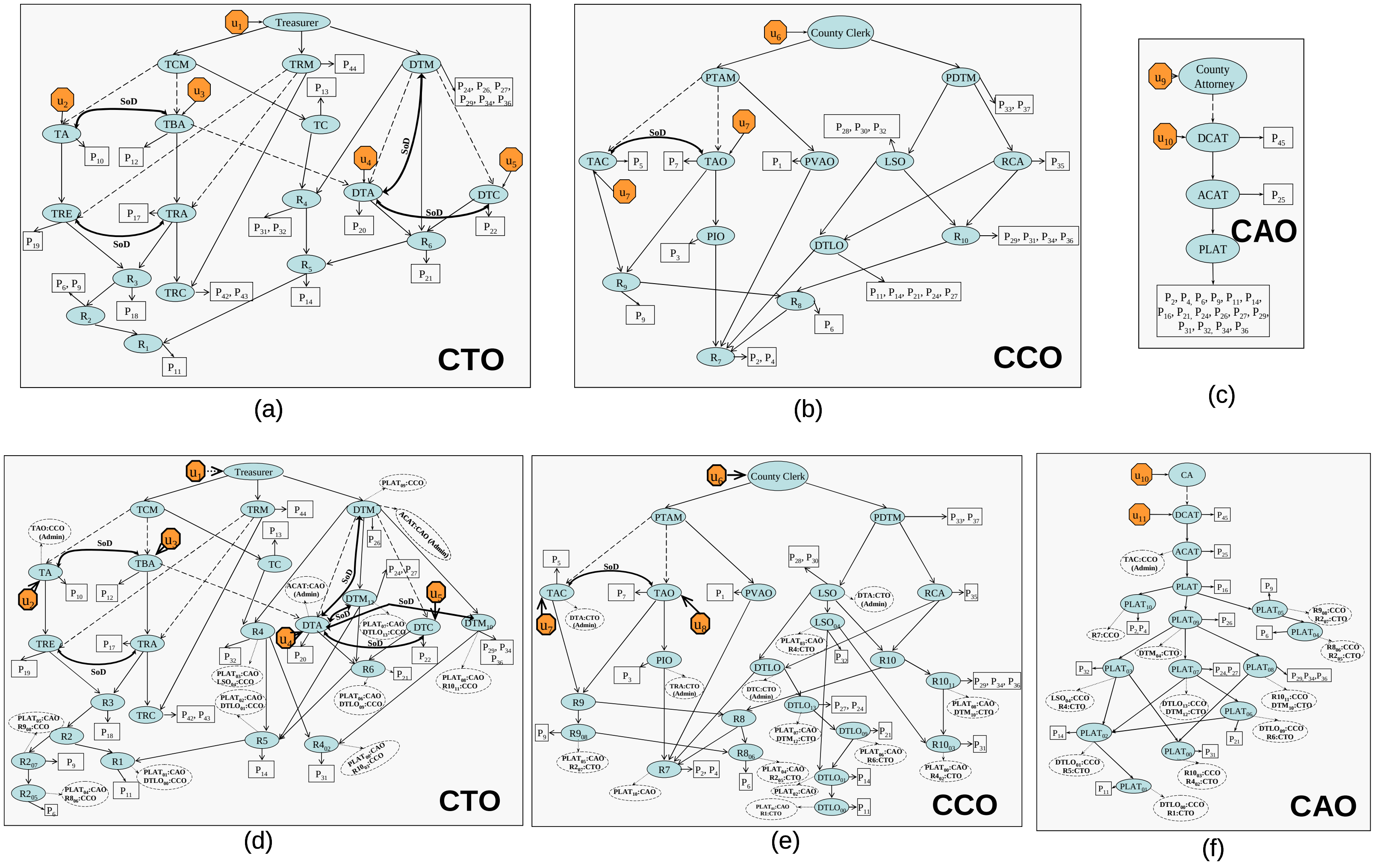
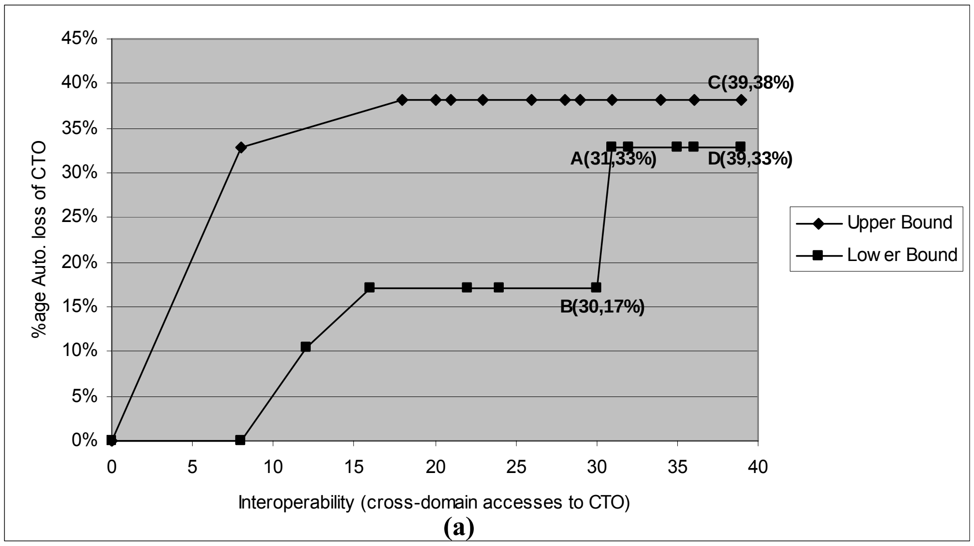


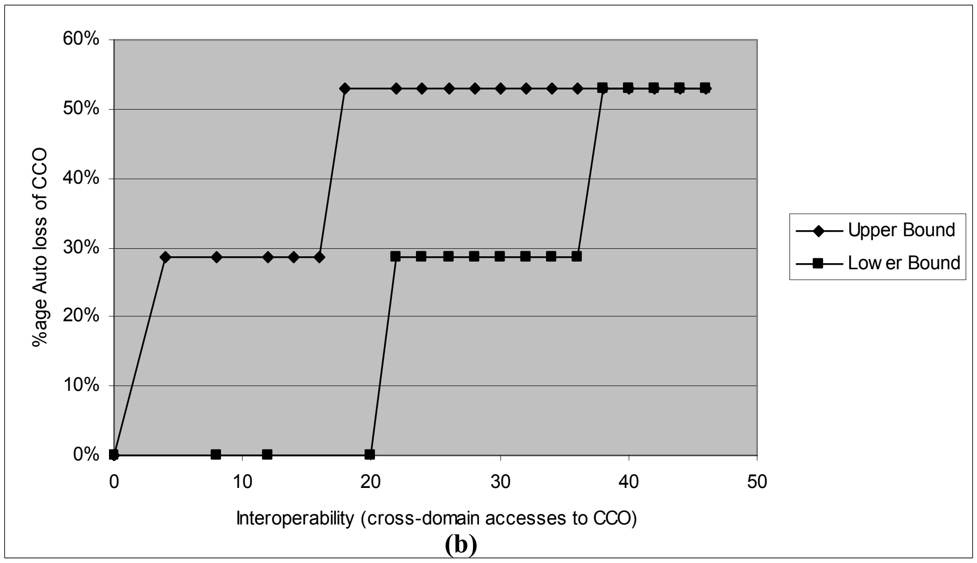
Fig. 3.16 (a) RBAC policy graph of County Treasurer Office (CTO), (b) of County Clerk Office (CCO), of County Attorney Office (CAO) prior to role mapping. (d) RBAC policy graph of CTO, (e) of CCO, and (f) of CAO after to role mapping.

Role mapping	Security Violation	Violation Type	Affected Domain
TA:CTO $\underset{I}{\geq}$ TAO:CCO PIO:CCO $\underset{I}{\geq}$ TRA:CTO	Enables u_2 to inherit the permissions of conflicting role TRE and TRA by accessing the role TA	Role-specific SoD	CTO
DTLO:CCO $\underset{I}{\geq}$ DTC:CTO LSO:CCO $\underset{I}{\geq}$ DTA:DTC $R10_{11}$:CCO $\underset{I}{\geq}$ DTM_{10}:CTO	Enables u_6 to access conflicting cross-domain roles DTA and DTM_{10}.	Role-specific SoD	CTO
DTM:CTO $\underset{I}{\geq}$ ACAT:CAO ACAT:CAO $\underset{I}{\geq}$ TAC:CCO TAC:CCO $\underset{I}{\geq}$ DTA:CTO	Enables u_1 to inherit the permissions of conflicting roles DTM and DTA concurrently.	Role-specific SoD	CTO
DTA:CTO $\underset{I}{\geq}$ ACAT:CAO $PLAT_{09}$:CAO $\underset{I}{\geq}$ DTM:CTO	Allows user u_4 assigned to role DTA to access the permissions of the senior role DTM.	Role-assignment	CTO

Fig. 3.17 Security violations of the meta-policy of Fig. 3.16.



(a)



(b)

Fig. 3.18 Interoperability versus autonomy loss

Table 3.4
Cardinality and user assignment of roles used in autonomy loss measurement of Fig. 3.16

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	

3.10.1. Authorization and order-independence

During the process of policy composition the access control policies of collaborating domains may get modified; however, such modifications should not change the access privileges of local users over local objects. In addition, the integrated policy should be independent of the order in which the collaborating domains' policies are merged. These requirements are stated for RBAC policy composition in PIRs 1 – 4 in Section 3.7.1.

The first PIR, stipulating element preservation, holds trivially in the merged policy graph as the policy merging algorithm, *RBAC-integrate*, does not remove any element except the newly created redundant roles which are not present in the original RBAC policy graphs of collaborating domains. Similarly, all the relations specified in the original RBAC policy graphs of collaborating domains are implied in the meta-policy graph. These relations include user-role assignment, role-permission assignments, separation of duties, and role hierarchy. The user-role assignment, permission assignment, and SoD relations remain unaltered in the multi-domain RBAC graph.

However, the role hierarchy and permission assignment may get change in the process of mapping equivalent cross-domain roles. During this process, new roles may be created by splitting existing roles. As a result of this role splitting, some of the permissions assigned to the parent role, say r, may get reassigned to the newly created child role, say r_j. Also, the newly created junior role r_j may inherit the permissions of some of the roles junior to the parent role r in the *I-hierarchy* semantics. However, the *I-hierarchy* relation $r \underset{I}{\geq} r_j$ preserves all the hierarchy relationships between the parent role r and all its junior roles. This means that all the permissions that can be acquired through r prior to role splitting can also be acquired after splitting of r. Hence, the user authorizations specified in the original RBAC policies of collaborating domains are preserved in the meta-policy graph.

To prove that the composed meta-policy is independent of the order in which domains' policies are merged, we need to show that the policy integration algorithm, *RBAC-integrate*, is both commutative and associative. The commutativity and associativity properties of *RBAC-integrate* are discussed in Section 3.7.4.

3.10.2. Security constraints

In this sub-section, we formally prove that the meta-policy composed by the proposed policy integration mechanism is secure. In particular, we show that no security vulnerability due to role-assignment violation (Definition 3.1), role-specific SoD violation (Definition 3.2), and user-specific SoD violation (Definition 3.3) can occur in the meta-policy. Note that in the context of RBAC, these are the only three security vulnerabilities that may lead to unauthorized accesses. The consistency conditions defined in [56] also check the correctness of RBAC policy specification against the violation of the above three constraints.

3.10.2.1. Notations

For stating the above claim about security of meta-policy in a formal manner, a state-based representation is needed. Since it is difficult to comprehend the state-transition semantics of RBAC policy from the graph-based specification, we introduce some matrix based notations and definitions for specifying the security properties of meta-policy.

Let A_k denote the adjacency matrix corresponding to the RBAC graph (with only user-role nodes) of domain *k* and A_k^+ be the transitive closure of adjacency matrix A_k. $\dim(A_k) = \dim(A_k^+) = (|U_k|+|R_k|) \times (|U_k|+|R_k|)$, where U_k is the set of user and R_k is the set of roles of domain *k*. The authorization of users over roles can be determined by applying the projection operator π_{ur} over the corresponding closure matrix.

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.π_{ur}: $\{A_k, A_k^+\} \rightarrow U_k \times R_k$. Projection of a closure matrix A_k^+ defines all possible user to role authorizations in domain *k*.

$$\forall a_{ij} \in \pi_{ur}(A_k^+), \quad a_{ij} = \begin{cases} 1, & \text{if there is an access path from } u_i \text{ to } r_j \\ 0, & \text{otherwise} \end{cases}$$

Note that *a 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 $|U| \times |R|$ $(U=\bigcup_k U_k, R=\bigcup_k R_k)$ 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.

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

3.10.2.2. Verification of security constraints

Any access control state derived from the meta-policy is secure if it does not violate the security constraints of collaborating domains' RBAC policies. This is formally stated in the following definition.

Definition 3.5: *A state S is secure with respect to the role-assignment, role-specific SoD, and user-specific SoD constraints of domain k if and only if following conditions hold in S:*

1. $a_{ij} \in \pi_{ur}(A_k^+)$ *and* $s_{ij} \in S$, $s_{ij} \le a_{ij}$.
2. *There does not exist any user* $u_i \in U$ *who accesses two or more roles in the conflicting role set* $R_{con} = \{r_1,..,r_n | conf\text{-}role(r_i,r_j),\ 1 \le i,j \le n \text{ and } i \ne j\}$. *Formally,* $\forall u_i \in \mathrm{U}, \sum_{r_j \in R_{con}} s_{ij} \le 1$.
3. *Let* U_{r_con} *be the set of conflicting user sets* (λ_r) *of role r.* $U_{r\_con} = \{\bigcup_m \lambda_r^m\}$ *and* $\lambda_r^m = \{u_{m_1},..,u_{m_p} \mid conf\text{-}user(u_{m_i}, u_{m_j}, r),\ 1 \le i,j \le p \text{ and } i \ne j\}$. *For each role* $r \in R_k$ *which have a non-empty set* U_{r_con}, *at most one user from each of the conflicting user sets* ($\lambda_r \in U_{r\_con}$) *accesses role r in sate S. Formally,* $\forall r_j \in \mathrm{R}_k \left(\forall \lambda \in \mathrm{U}_{r\_con}, \sum_{u_i \in \lambda} s_{ij} \le 1 \right)$.

The first condition in the above definition captures the role assignment constraint, i.e., in any secure state a user can access a local role if and only if there is an intra-domain access path from the user node to the role node in the local access control policy of corresponding domain. The second condition specifies that conflicting roles cannot be accessed by same user in any secure state, and the third condition defines the user-specific SoD constraint implying that conflicting users of a role cannot access that role concurrently in any secure state.

Having defined the necessary and sufficient conditions for a secure access control state, we claim in the following theorem that the meta-policy generated by the proposed policy integration framework is secure. In particular, any state that can be derived from the meta-policy will not cause any violation of role-assignment, role-specific SoD, and user-specific SoD constraints specified in the local RBAC policies of collaborating domains.

Theorem 3.3: Given $G_1,\ldots,G_n$, $n \ge 2$, the RBAC policy graphs of *n* collaborating domains. Let G be the multi-domain RBAC graph composed from $G_1,\ldots,G_n$ by applying the role-mapping algorithm, *RBAC-integrate*, and conflict resolution algorithm, *ConfRes*.

Assuming all G_is are consistent and conflict-free, any state S reachable from the meta-policy graph G is secure with respect to the *role-assignment*, *role-specific SoD*, and *user-specific SoD* constraints defined in each G_i ($1 \leq i \leq n$).

Proof of Theorem 3.3 is given in Appendix A.

3.11. Meta-policy Composition and Mediation Process

In this section, we describe an overall process of policy integration and mediation. The process, shown in Fig. 3.19, consists of following phases: *policy comparison*, *merging and resolution*, and *policy mediation*.

The policy comparison phase deals with the reconciliation of semantic differences among the access control policies of collaborating domains. In this phase, domains' access control policies are analyzed to identify shareable cross-domain objects and to resolve the semantic conflicts among these objects. The technical challenges related to the resolution of semantic heterogeneity are discussed in Sections 3.6 and 3.7. We have not described any specific strategies for resolution of semantic heterogeneity as it is beyond the scope of this dissertation. However, this issue has been extensively investigated by the database community [132, 90, 104, Vet98, 13].

In the merging and resolution phase, the access rights of users over the cross-domain objects are established and the resulting authorization conflicts are resolved. In the context of RBAC, inter-domain authorizations are defined by mapping cross-domain roles. Role mappings can be established automatically based on the correspondence among cross-domain shareable objects as discussed in Section 3.7.2. In addition the security policy administrators, responsible for global meta-policy, may also specify such mapping. For resolution of authorization conflicts due to inconsistent role mapping, the IP based approach discussed in Section 3.8, can be used. The solution to the underlying IP problem may not be feasible implying that a meta-policy with the given security, autonomy, and interoperability constraints cannot be composed. In this case, a new meta-policy needs to be composed after revising the local policies and interoperability constraints. Such revision may include relaxation of autonomy requirements, relaxation of local privileges and constraints, and reduction in the degree of interoperability. The revised policies and interoperability constraints are analyzed in the mediation phase and need to be approved by the respective domains' policy administrators. For automating the mediation process, the policy administrators may specify the possible policy relaxations a priori, in decreasing order of preference, along with the acceptable bounds or thresholds on such relaxations.

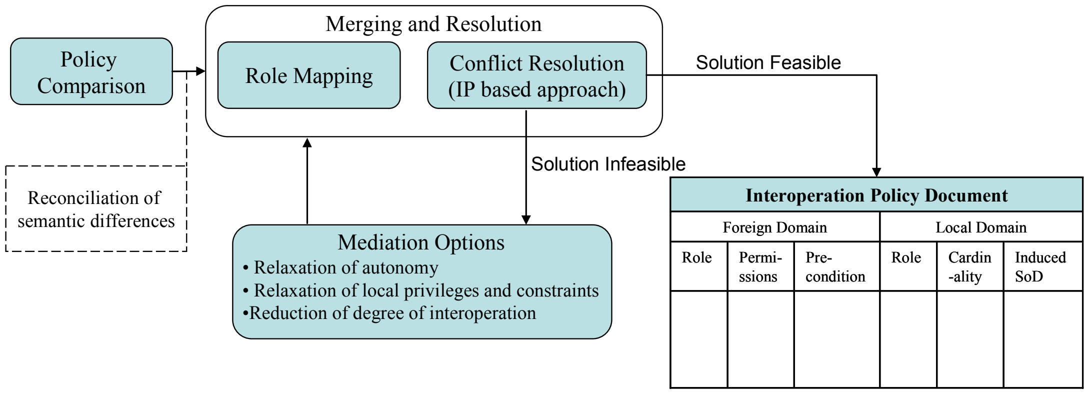


Fig. 3.19 Overall process of policy composition and mediation.

If a feasible solution to the IP problem exists, the IP-based conflict resolution module generates a consistent and secure meta-policy with maximal interoperation support under the given optimality measure and interoperation constraints. For analyzing the implications of the resulting policy, a global policy document can be generated from the composed meta-policy for each collaborating domain. A possible schema for such document is shown in Fig. 3.19. This document facilitates a domain policy administrator in assessing the degree of interoperability and the level of autonomy offered by the composed meta-policy. For instance, the document shown in Fig. 3.19 contains information about the cross-domain roles that can be accessed by local users of a domain, the permissions associated with the accessible cross-domain roles, and the pre-conditions for accessing shareable cross-domain roles. The pre-condition may specify what local role a user must assume before accessing a cross-domain role. In addition, the implications of the composed policy with respect to a domain's autonomy can be assessed from the local domain sub-schema of the policy document of Fig. 3.19. This sub-schema specifies the local roles with reduced cardinalities or local roles with induced SoD constraints. As mentioned above, both reduction in role cardinalities and addition of induced SoDs, amount to autonomy loss for a given domain.

3.12. Related Work

Several research efforts have been devoted to the topic of policy composition in federated multi-domain environment [61, 93, 40, 26]. In particular, major emphasis is given on the detection and resolution of conflicts in interoperation policies. Conflicts appearing in a multi-domain meta-policy can be divided into four types: i) modality conflicts, ii) multiple management, iii) cyclic inheritance, and iv) separation of duties (SoD).

Modality conflicts in a policy arise because of the existence of both positive and negative authorizations for a given subject-object pair. Multiple management conflicts occur when multiple administrators having authority over a common set of subjects and objects, specify conflicting authorizations in their respective policies. In that sense, multiple management conflicts are similar to modality conflicts. Modality conflicts are resolved based on the policy precedence relationship which implies that the most specific authorization overrides the less specific one [93, 26, 22]. In case the precedence relationship cannot be established between the conflicting authorizations, the negative authorization dominates the positive one. Modality conflicts cannot occur in RBAC policy specification because negative authorizations are not supported in RBAC model.

Cyclic inheritance conflicts mainly occur in interoperation of systems employing multi-level security policies such as lattice-based access control (LBAC) and role-based access control (RBAC) [61, 40]. In such interoperation, the cross-domain hierarchy relationship may introduce a cycle in the interoperation lattice enabling a subject lower in the access control hierarchy to assume the permissions of a subject higher in the hierarchy. Dawson *et. al.* [40] have discussed a mediator-based framework for establishing secure interoperation among heterogeneous systems with LBAC policies. In this framework, cyclic conflicts in the interoperation lattice are resolved by the manual intervention of a policy editor. The policy editor allows the integrator/administrator to incrementally specify the cross-domain lattice relationships. After addition of each relationship, the editor determines the consistency of resulting meta-policy and identifies all relationships involved in potential security violation. Interoperation conflicts are thus resolved by withdrawing all cross-domain relationships resulting in potential security violation or removing one or more relationships until the violation is corrected. Resolution of interoperation conflicts by manual intervention of policy administrator is a slow and ad-hoc process and provides no guarantee on the optimality of the resulting interoperation system. In case there are multiple policy administrators, a consensus on the resolution needs to be obtained. Gong *et. al*. [61] have investigated interoperation of systems employing multi-level access control policies. They have proposed several optimization techniques for resolution of interoperation conflicts. However, these resolution techniques are specific to cyclic inheritance conflicts and do not consider other types of interoperation conflicts.

Separation of duties (SoD) prevent two or more subjects from accessing an object that lies within their conflict of interests or disallow a subject from accessing conflicting objects or permissions, e.g., same managers cannot authorize payments and sign the payment checks [93, 76]. Violations of SoD constraints may occur in a global meta-policy because of the interplay of various policy constraints across domains. The resolution of interoperation inconsistencies related to separation of duty constraints has not been adequately investigated and the existing approaches rely on manual intervention of policy administrators to resolve SoD conflicts [93]. As mentioned earlier, manual resolution is a tedious process and may not yield optimal interoperation. The policy composition methodology proposed in this chapter provides a single framework for optimal and automated resolution of interoperation conflicts related to RBAC policies. These conflicts include both cyclic inheritance and SoD violations.

Gavrilla et. al. [56] have defined a set of necessary and sufficient conditions for composition of a consistent RBAC policy. The criterion for consistent policy composition is defined in terms of cardinality, hierarchy, and SoD constraints. Accordingly, a consistent meta-policy can be composed by incrementally checking the consistency of cross-domain role mappings. A role mapping that satisfies all the consistency conditions with respect to the resulting policy can be added to the final meta-policy. Such incremental composition of meta-policy depends on the order in which role mappings are evaluated and therefore the resulting meta-policy may not be optimal.

3.13. Conclusions

In this chapter, we have addressed the issue of secure interoperation in a multi-domain environment. In particular, we focused 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 proposed policy composition mechanism is a two step process including composition of a global meta-policy from the access control policies of collaborating domains and removing conflicts from the global policy in an optimal manner. Another key requirement of policy composition is to maintain the autonomy of all collaborating domains. However, 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, domains may tolerate some autonomy loss for establishing more interoperability. In this chapter, we have formulated the problem of secure interoperation as an optimization problem with an objective of maximizing interoperability without causing any security violation of collaborating domains and keeping the autonomy losses within acceptable limits

4. VERIFICATION OF DISTRIBUTED WORKFLOWS IN AN AUTONOMOUS MULTI-DOMAIN ENVIRONMENT

In Chapter 3, we proposed a global meta-policy based approach for establishing secure interoperation in a federated multi-domain environment. The meta-policy based approach requires complete disclosure of the local policies of all collaborating domains. However, in an autonomous and loosely-coupled multi-domain environment, domains may not disclose their policies due to security and privacy concerns. The key challenge in absence of a global meta-policy is to design collaborative applications that are consistent with the policies of all collaborating domains.

To address this challenge, we propose an approach for verifying secure composibility of distributed applications requiring interactions among autonomous domains in a loosely-coupled multi-domain environment. This approach is designed for verifying the specification of distributed workflows for conformance with the time-dependent access control policies of collaborating but autonomous domains. The objective of workflow composibility verification is to ensure that all the users or processes executing the designated workflow tasks conform to the security policy specifications of all collaborating domains.

4.1. Issues and challenges in workflow verification

Supporting distributed workflow based applications in an autonomous multi-domain environment in the absence of a global meta-policy is a challenging task. The individual domains in such environment operate according to their own security and access control policies which may be context driven [124, 78]. Depending upon the type of workflow applications, several contextual parameters such as time, location, environment, and agenda may be considered and can pose substantial challenges in assuring secure execution of such applications [9, 19a, 126]. In particular, the resource access requirements of distributed workflows may conflict with the access control policies of collaborating domains, which may cause deadlock or erroneous execution. For ensuring secure and correct execution of a distributed workflow, the workflow specifications need to be verified for conformance with the context-driven access control policies of all collaborating domains.

The proposed verification approach is designed for distributed workflow applications that require long-term interactions among various domains and are executed on a recurrent basis. Examples of such recurrent workflow applications include: check clearance processing among banks, insurance claim processing, health-care administration, real-time process control systems, and distributed data processing for stream data warehouses [57, 130, 96, 80, 135, 123]. In all these applications, a predefined workflow specifies a logical sequence of activities or tasks that needs to be performed by collaborating and possibly autonomous domains. Some of these applications have strict deadlines for workflow completion which may not always be satisfied because of the time dependent access control policies of domains. These workflow applications are recurrent in a sense that they need to be invoked repeatedly after a fixed or variable time interval. For instance, the check clearance workflow among banks is invoked regularly to process a batch of check clearance requests [57, 34]. Similarly, the distributed data processing workflow for zero latency data stream warehouse is periodically invoked for mining the continuous data streams in near real-time [96]. For verifying secure composibility of such workflows, the following two questions need to be answered:

- *Does the security and authorization policies of collaborating domains, support execution of the distributed workflow under the given timing constraints?*
- *What are the possible time instants at which the distributed workflow can be invoked recurrently?*

A major challenge in the verification of distributed workflows is posed by the time-dependent *non-reentrant* behavior of collaborating domains [2, 53]. The behavior of a domain is characterized as *reentrant* or *non-reentrant* based on the underlying software system enforcing a time-dependent access control policy. We use Generalized Temporal Role Based Access Control (GTRBAC) [78] model to specify the time dependent access control policy of a domain. In the software engineering terminology, a *non-reentrant* system does not allow its multiple simultaneous, interleaved, or nested invocations and only one instance of such system exists at any time [53]. The *non-reentrant* behavior of a system is governed by its finite state model (FSM) and any interaction with such system has to be compatible with its current state [53]. At any time, a domain can have only one instance of its GTRBAC policy against which all access requests are evaluated. In addition, the GTRBAC policy instance has a finite number of authorization states. Therefore, according to the above criterion a domain is a *non-reentrant* system.

For verifying workflow composibility, the distributed workflow specifications need to be analyzed for being consistent with the individual as well as with the collective behavior of collaborating domains. Accordingly workflow composibility verification entails two steps: i) verification of workflow specifications with respect to the FSM of individual domains, and ii) verification of dependencies among domains for execution of workflow tasks. These two steps can either be carried out separately in the above order, or can be performed simultaneously by using a unified global meta-policy that captures all intra-domain and inter-domain authorizations [14, 117, 118]. The methodology proposed in this chapter uses the two step verification approach and does not consider the meta-policy based approach for the following reasons:

The unified global meta-policy is composed by integrating the access control policies of all collaborating domains; however, domains may not disclose their policies due to privacy concerns.

More importantly, domains are autonomous in deciding when to join or leave the collaborative environment. Whenever a new domain joins the collaboration, the meta-policy needs to be reconfigured. Consequently, all the workflows verified with respect to the previous meta-policy need to be verified again. Such re-verification of existing workflows due to joining of new domains is not needed in the proposed methodology. The meta-policy is also reconfigured when any domain leaves the collaboration or changes/updates its access control policy. Again such reconfiguration of meta-policy triggers re-verification of all previously verified workflows including the ones that do not have any task assigned to the departing domains or domains that have changed/updated their policies. In the proposed approach, a workflow is re-verified only if a domain participating in workflow execution updates its policy or leaves the collaboration.

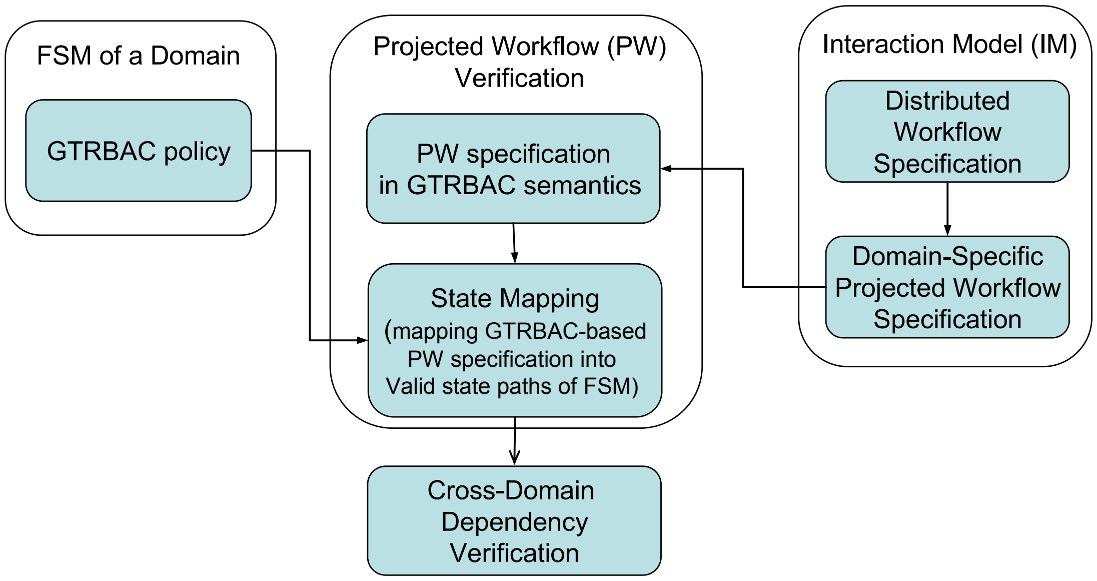

FSM of a Domain
GTRBAC policy
Projected Workflow (PW) Verification
PW specification in GTRBAC semantics
State Mapping (mapping GTRBAC-based PW specification into Valid state paths of FSM)
Interaction Model (IM)
Distributed Workflow Specification
Domain-Specific Projected Workflow Specification
Cross-Domain Dependency Verification

Fig. 4.1 Overall process for workflow verification

Fig. 4.1 depicts the proposed two step approach for verification of secure workflow composition. The approach relies on decomposition of a distributed workflow into domain-specific workflows called *projected workflows*. These projected workflows are verified by the respective domains in terms of the authorization and execution time requirements. After verification of projected workflows, the cross-domain dependencies amongst the workflow tasks performed by different collaborating domains are verified. The timing information computed in the projected workflow verification phase is used to determine an interleaving of projected workflow tasks that satisfies the cross-domain dependencies of the distributed workflow. This timing information is also used to determine a feasible schedule for the overall verified distributed workflow. For workflow composibility verification, we assume that the FSM of each domain's GTRBAC policy is given and the distributed workflow is specified using interaction model (IM).

4.2. Interaction Model for Workflow Specification

For verifying workflow composibility, a formal and precise specification of the distributed workflow is needed. In particular, the specification should be able to capture the collaboration requirements among the domains performing the tasks of the distributed workflow. We use the term *component service* to refer to a task or set of tasks in a distributed workflow that can be executed by a domain independently. More precisely, a *component service* encapsulates a set of domain-specific tasks that are advertised to other interacting domains as a single capability/functionality of the domain.

Interaction models, such as Unified Modeling Language (UML) sequence diagrams [100] and International Telecommunication Union (ITU) message sequence charts (MSC) [69], have been widely used to model specifications of distributed workflows requiring communication among collaborating domains for service provisioning [49, 83]. We use UML 2.0 sequence diagrams to model the distributed workflow specification. A sequence diagram, shown in Fig. 4.2, specifies the communication among the interacting entities as message exchanges. The vertical line in a sequence diagram represents time and is called the *lifeline* of the corresponding interacting entity. Message exchange between two entities is shown by an arrow from the sender to the receiver. The communication between the interacting entities can be either *synchronous* or *asynchronous.* In *synchronous* communication the sender blocks for the subsequent action to complete, whereas, there is no nesting of control in *asynchronous* communication. We consider all the message exchanges to be *synchronous* for the sake of simplicity.

In the following, we provide a formal definition of workflow sequence diagrams.

4.2.1. Workflow sequence diagram (WSD)

In WSD, we consider the interacting entities of a sequence diagram as interacting domains (IDs) defined in the following definition. In this definition, the incoming and outgoing messages at an ID corresponds to input and output events respectively.

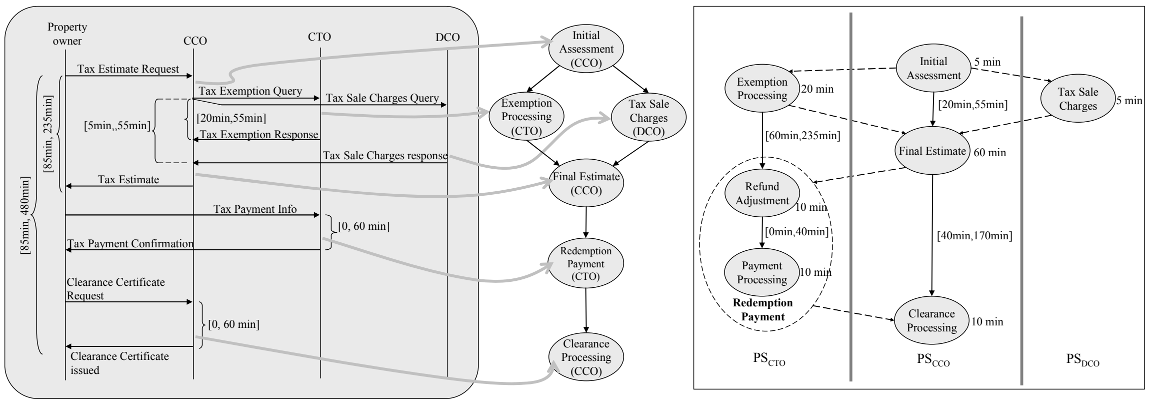
Definition 4.1 (ID). An Interacting Domain (ID) is a tuple {EV,$\leq$, CS, T } where,

- *EV= In $\cup$ Out is a set of events which are partitioned into input and output events.*
- *$\leq \subseteq EV \times EV$ is a partial ordering of events such that $ev_i < ev_j \Rightarrow ev_i$ occurs before ev_j*
- *CS: {c_1,c_2, ...c_r} is a set of component services offered by the ID*
- *T: In$\rightarrow$ (CS $\times$ 2^{Out}) maps the input event to the corresponding CS and set of output events.*

Definition 4.2 (WSD). A Workflow Sequence Diagram (WSD) is a tuple WSD= {**ID**, TR} such that:

- ***ID**= {ID_1, ID_2, ID_3 ,……….ID_r } is a finite set of interacting domains ID_i={EV_i ,$\leq_i$, CS_i, T_i} $1 \leq i \leq r, r \leq d$ where d is the total number of domains and EV_i are disjoint sets of events. Without loss of generality that ID_1 always initiates the interaction.*
- *For a given ID_j, TR maps a pair of events to the minimum and maximum duration allowed between them. TR(ev_i, ev_j)= [d_l, d_u] where d_l, d_u $\in Z^+$, ev_i < ev_i and i $\neq$j.*

The set **ID** in a WSD contains all the interacting domains that provide the required *component services* for workflow composition. The set of *component services* offered by an ID is specified in its definition. A *component service* is associated with one or more input events. An input event occurs with the arrival of an incoming message. The mapping function *T* in the ID definition maps the input event to a *component service* and a set of output events (output messages). The second element in the *WSD* tuple *TR* is a function that maps any pair of events (*ev_i, ev_j*) to a finite time interval. This interval specifies the minimum and maximum duration allowed between *ev_i* and *ev_j* provided that *ev_i* occurs before *ev_j*. The WSD considered in this chapter supports the notion of parallel interactions through concurrent message transmission as specified in UML 2.0 sequence diagrams [103]. Such parallel interactions are needed to model the parallel invocation of *component service*s in different IDs. For instance, the concurrent messages "Tax Exemption Query" and "Tax Sale Charge Query" corresponds to parallel invocation of the *tax exemption processing* service in County Treasurer Office (CTO) and *tax sale charges processing* service in District Clerk Office (DCO).



(a) (b) (c)

Fig. 4.2. (a) WSD of a distributed workflow involving urgent processing of tax redemption request for delinquent real-estate property. (b) *component service*s required for performing tax redemption processing. (c) PW specification for each domain.

Example 4.1: Fig. 4.2(a) shows the WSD of a distributed workflow involving urgent processing of tax redemption request for delinquent real-estate property. The urgent processing entails that the entire business process of tax redemption be completed in one business day. The domains involved in provisioning of this distributed workflow include: property owner, County Clerk Office (ID_{CCO}), County Treasurer Office (ID_{CTO}), and District Clerk Office (ID_{DCO}) as shown in Fig. 4.2(a). The distributed workflow of Fig. 4.2(a) is initiated by the property owner by filing a tax estimate request for the delinquent property with ID_{CCO}. This request invokes the *initial assessment* service in ID_{CCO}. After completion of the *initial assessment*, the ID_{CTO} and ID_{DCO} are queried for *exemption processing* and *tax sale charges* for the given delinquent property index. Based on the exemption amount and tax sale charges returned by ID_{CTO} and ID_{DCO} respectively, the *final estimate* for the tax redemption amount is calculated and is submitted to the property owner. Upon receiving the redemption cost estimate, the property owner initiates the *payment processing* service for tax redemption with ID_{CTO}.

After the *payment processing* is completed, the property owner requests for issuance of delinquent tax clearance certificate which launches the *clearance processing* service in ID_{CCO}.

The *component service*s associated with the events of the WSD of Fig. 4.2(a) are shown in Fig. 4.2(b). The time interval between events in the WSD of Fig. 4.2(a) corresponds to the interval returned by the *TR* function for the corresponding events pair as specified in the WSD definition. For instance, the time interval [85min, 480min] between the tax exemption request, initiating the *tax redemption processing* workflow, and the clearance certificate issued event implies that the distributed workflow must complete within 480 minutes (8 hours) relative to the initiation time of the workflow. The lower bound of 85 minutes implies that processing of this distributed workflow takes at least 85 minutes.

4.2.2. Domain-specific projected workflow specification

The WSD specifies a high level description of the distributed workflow and considers the *component service*s as atomic operations provided by domains. However, a *component service* may encapsulate a workflow process comprising multiple tasks. For example the *redemption payment* service, shown in Fig. 4.2(c) comprises two tasks, namely, *refund adjustment* and *payment processing*. The *refund adjustment* task precedes the *payment processing* task in the workflow associated with the *redemption payment* service. This low level description of the *component service* is specific to a domain and is not required for distributed workflow specification. However, as discussed later, such description is needed for composibility verification, which is performed for each domain separately. In the following we formally define a *component service* of an ID.

Definition 4.4 (CS). *For an ID, a component service* ($c \in CS$) *is a tuple* $c = \{T_c, \leq, \lambda, \beta\}$ *where* T_c *is a set of tasks included in the workflow of c, and* $\leq \subseteq T_c \times T_c$ *specifies partial ordering between the tasks such that* $\tau_1 \leq \tau_2$ ($\tau_1, \tau_2 \in T_c$) *implies that* τ_1 *precedes* τ_2 *in the execution order.* λ: $T_c \rightarrow Z^+$ *is a function that maps a task to the time duration required for its completion.* β: $T_c \times T_c \rightarrow Z^+ \times Z^+$ *maps a task pair* (τ_1, τ_2) *to an interval*

$[d_l, d_u]$ ($d_l, d_u \in Z^+$) that specify the minimum and maximum duration between the completion and initiation of τ_1 and τ_2 respectively.

Each task of the *component service* has certain authorization constraints specified in the corresponding domain's access control policy. In order to verify workflow composibility, we need to ensure that all the authorization constraints associated with each task of the *component service* are satisfied. As discussed in the Introduction, each domain is autonomous and may not reveal its access control policy to other domains for security and privacy concerns. To facilitate workflow composibility verification with respect to the access control policy of each domain, a domain-specific projected workflow (PW) specification is generated from the WSD of the distributed workflow. The PW specification provides the following information related to the corresponding domain's involvement in the distributed workflow: i) *component service*s provided by the domain in the WSD, ii) temporal constraints between the *component service*s, and iii) the task workflow associated with each *component service* as defined above. We model the PW of a domain as a directed acyclic graph, which is constructed from the WSD and the *component service* specification. Each node of PW graph represents a task and an edge (τ_1, τ_2) represents the precedence relationship between the tasks τ_1 and τ_2. The node of a PW graph is annotated with a non-negative integer specifying the time duration for completion of the corresponding task. This information is obtained from the task duration mapping function λ given in the respective CS specification. Each edge in a PW graph is annotated with an interval that specifies the minimum and maximum duration between the completion and initiation of successive tasks connected by the edge. In case the successive tasks belong to the same CS the interval mapping function β provides this information; otherwise, the interval is computed from the WSD specification.

Fig. 4.2(c) shows the PW graph of ID_{CTO}, ID_{CCO} and ID_{DCO} for the tax redemption workflow described in Example 4.1. The dashed arrows in Fig. 4.2(c) are not a part of any PW graph and are used to illustrate the temporal ordering of cross-domain *component service*s in the distributed workflow specification.

4.3. FSM of a domain's Access Control Policy

To analyze the consistency of the workflow specification against a domain's dynamic and *non-reentrant* behavior, a state based representation of the domain's access control policy is needed. We use GTRBAC model for specification of the time-dependent access control policies. In the following we first describe the GTRBAC semantics and constraints used for modeling access control policies of domains and then describe a finite state model (FSM) for state based representation of GTRBAC policies.

4.3.1. Preliminaries and assumptions

A domain's GTRBAC policy specifies the authorizations for its *component services*. As mentioned in Section 4.2, a *component service* is essentially an encapsulation of one or more tasks with certain temporal and ordering constraints. At the interaction modeling level, a task is viewed as an operation on a resource by an authorized subject without considering who is authorized for the resource access and how such operation can be executed. In the GTRBAC model, a task can be represented as an activation of a particular role by an authorized user, where the role is a collection of permissions required to perform the requested operation on the underlying resource object(s) and the user corresponds to the subject executing the task. For establishing the semantics relationship between a *component service* and the underlying user-role activation in the GTRBAC policy, we define a function called *domain role mapper* (DRM) that maps each task of the *component service* to a set of user role activation pairs (*u*, *r*) such that each role *r* in the pair (*u*, *r*) has all the relevant permissions required for processing of the task and the user *u* is authorized for role *r*. Formally: $DRM(\tau) = \{(u, r) \mid u \text{ is authorized for } r \text{ and } r \text{ contains all permission required for processing } \tau\}$.

As discussed in the Section 4.1, we are interested in verification of distributed workflow*s* that are executed recurrently. For supporting such workflows a domain's GTRBAC policy must allow access to its roles on a recurrent basis. Therefore, we consider only those roles that can be accessed infinitely often and have a periodic enabling time. For instance, a *tax filing* role in the tax payment workflow is enabled daily from 9:00 am to 5:00pm and can be accessed any time within its enabling interval. GTRBAC allows specification of periodic time intervals for various role-related events including role enabling. These periodic intervals are specified using periodic expressions described in Section 2.2 of Chapter 2.

We assume that the periodic expressions corresponding to enabling of each role are specified using the same number of calendars. For instance, if $PE_1 = \sum_{i=1}^{n} O_i Cal_i \triangleright x.Cal_d$ and $PE_2 = \sum_{j=1}^{m} O_j Cal_j \triangleright y.Cal_d$ denote the periodic expressions for enabling of any two roles then $Cal_n = Cal_m$. If $Cal_m < Cal_n$, the left slicing operation defined in [97] can be used to expand PE_2 to Cal_n or vice versa. The duration of the calendar Cal_n in terms of the basic calendar Cal_1 is given by $duration(Cal_n/Cal_1)$. Let $I_n = [0, duration(Cal_n/Cal_1)]$ denote the interval associated with the calendar period Cal_n. We assume that the calendar Cal_n is the smallest calendar that contains the enabling intervals of all roles of the given GTRBAC policy. Therefore, I_n corresponds to the smallest interval that covers the enabling interval of all roles in one calendar period. We denote the set of all intervals of a periodic expression PE that are contained in I_n by $\Gamma(I_n, PE)$. For example, for I_n = [0, 1440 minutes (1 day)] and $PE = \{all.days, \{9, 15, 23\}.Hours, \{20, 50\}.Minutes \triangleright 15.Minutes\}$, $\Gamma(I_n, PE)$ = {[09:20, 09:35], [09:50, 10:05], [15:20, 15:35], [15:50, 16:05], [23:20, 23:35], [23:50, 23:59], [00:00, 00:05]}. For computing $\Gamma(I_n, PE)$, we divide any interval $I = [a , b]$ of a PE that overlaps with I_n but is not fully contained in I_n, into two intervals $I_1 = [a, duration(Cal_n/Cal_1)]$ and $I_2 = [0, b - duration(Cal_n/Cal_1)]$.

Periodic intervals can be specified for various constraints such as, role enabling, role assignment, and role activation. However, for simplicity we consider periodic intervals for role enabling events only. Given a role set R and an interval I_n, the enabling intervals of all roles in R that are contained in I_n is denoted by the set EI_R.

$$EI_R = \bigcup_{r \in R} \Gamma(I_n, PE_r) = \bigcup_{r \in R} \{[a_r, b_r]\},$$

where, PE_r is the periodic expression for enabling of role r. We assume that each role has only one enabling interval in EI_R. If a role r has multiple enabling intervals, say

m, then we create roles $r_1, r_2, \ldots, r_m$, one for each of the m intervals. Each role r_j is similar to r in user-role assignment, role permission assignment, SoD, and trigger constraints.

4.3.2. GTRBAC policy specification

In this section, we discuss the syntax and semantics of various GTRBAC constraints used to specify the time dependent access control policies of domains. As discussed in Chapter 2, the GTRBAC constraints can be divided in to six types: i) user-role assignments and role-permission assignments, ii) periodicity constraints on role enabling, iii) role activation constraints, iv) run-time events, v) triggers, and vi) separation of duty (SoD) constraints. For specification of domains policies, we consider a restricted set of GTRBAC constraints. These constraints are listed in Table 4.1. The user-role and role-permission assignment expressions, listed in Table 4.1, specify the authorizations of users over the GTRBAC roles and the underlying resources. For simplicity, we do not consider any temporal and periodicity constraints on user-role and role permission assignments. Omission of these constraints does not restrict the expressiveness of the GTRBAC model considered in this dissertation, as the temporal constraints on user-role and role permission assignments can be specified using role enabling constraints [79]. The periodicity constraints on role enabling/disabling are specified using periodic expressions as Table 4.1, specifies a lower and upper bound on the activation duration of a given role by any user. Accordingly, the activation duration of a role in any session must be greater than or equal to $d_r^{\min}$ and less than or equal to $d_r^{\max}$, where $d_r^{\max} \geq d_r^{\min} > 0$. The original GTRBAC model does not constrain the activation of a role to a minimum duration. However for state-based analysis of a GTRBAC policy, we require that a role be activated for a finite number of times in any finite time interval. To satisfy this requirement, we have introduced the minimum activation duration constraint for each role.

Table 4.1
GTRBAC Constraints considered for specification of domains' policies

$r \in$ R, $u \in$ U, $p \in$ P, tg $\in$ TRG R is a set of roles, U is a set of users, P is a set of permissions, and TRG is a set of Triggers			
Constraints		Expression	Semantics
User-role assignment		(assign$_U$ r to u)	Role r is assigned to user u
Role-Permission assignment		(assign$_U$ p to r)	Permission p is assigned to role r
Periodicity constraint on role enabling		(PE$_r$ enable r)	Role r is periodically enabled during the intervals contained in the periodic expression PE$_r$
Duration constraint on role activation		([$d_r^{\min}$, $d_r^{\max}$] active r)	The activation duration of role r in any session must be greater than or equal to $d_r^{\min}$, and less than or equal to $d_r^{\max}$.
Run-time request		(activate/deactivate r for u)	User's/administrator's request for role activation or deactivation.
Trigger		ev, sp$_1$,…sp$_k$ $\rightarrow$ ev'	An event ev must be immediately followed by ev' provided that all status predicates sp_1,…sp_k hold at the time of the occurrence of ev.
SoD	Role-specific	$\forall$ r, r' $\in$ R-SoD(u), u-active(u, r) $\Rightarrow$ $\neg$u-active(u, r')	R-SoD(u) is a set of conflicting roles for user u, i.e., u can activate at most one role in R-SoD(u) at any given time.
	User-Specific	$\forall$ u, u' $\in$ U-SoD(r), u-active(u, r) $\Rightarrow$ $\neg$u-active(u', r)	U-SoD(r) is a set of conflicting users for role r, i.e., r can be activated by at most one user in U-SoD(r) at any given time.

Table 4.2.
Event and status predicates used in the restricted GTRBAC Model

R ∈ R, *u* ∈ U, *p* ∈ P, tg ∈ TRG R is a set of roles, U is a set of users, P is a set of permissions, and TRG is a set of Triggers			
Simple Event		Status Predicate	Semantics
enable *r*		ur-assigned(*u*, *r*)	*u* is assigned to *r*
Disable *r*		pr-assigned(*p*, *r*)	*p* is assigned to *r*
activate *r* for *u*		r-enabled(*r*)	*r* is enabled
de-activate *r* for *u*		r-active(*r*)	r is active in at least one user's session
		u-active(*u*, *r*)	*r* is active in *u*'s session
		trg-enabled(*tg*)	Trigger *tg* is enabled, i.e., the event *ev* defined in the body of the trigger *tg* has occurred and the status predicates hold.

Table 4.3
GTRBAC Policies of CTO and CCO domains

County Treasurer Office (ID_{CTO})			
User-role assignment	1	assign$_U$ u_1 to {TEP, TPP}; assign$_U$ u_2 to TPP; assign$_U$ u_3 to TRP	
Role-permission assignment	2	assign$_p$ p_1(Tax Exemption Records, {read,write,approve}) to TEP; assign$_p$ p_2(Tax Refund Records, {read,write,approve}) to TRP; assign$_p$ p_3(Tax Payment Records, {read,write,approve}) to TPP; assign$_p$ p_4(Tax Billing Records, {read,write,approve}) to TRP and TEP; assign$_p$ p_5(Tax Billing Records, {read) to TPP and TRP	
Periodicity constraints on role enabling	3	PE_{TEP}: all.Days+ 10.Hours+1.Minutes$\triangleright$ 420.Minutes; $\Gamma(I_n, PE_{TEP})$ = [09:00, 16:00] PE_{TPP}: all.Days+ 11.Hours+1.Minutes$\triangleright$ 240.Minutes; $\Gamma(I_n, PE_{TEP})$ = [10:00, 14:00] PE_{TRP}: all.Days+ 11.Hours+1.Minutes$\triangleright$ 240.Minutes; $\Gamma(I_n, PE_{TEP})$ = [10:00, 14:00]	(PE_{TEP}, enable TEP) (PE_{TPP}, enable TPP) (PE_{TRP}, enable TRP)
Role-activation constraint	4	([120min, 420min] active TEP); ([240min, 240min] active TPP); ([240min, 240min] active TRP)	
Separation of Duty	5	U-SoD(TPP) = {u_1, u_2}	
County Clerk Office (ID_{CCO})			
User-role assignment	6	assign$_U$ u_4 to {TAP_1, TAP_2}; assign$_U$ u_5 to {DTP_1, DTP_2}	
Role-permission assignment	7	assign$_p$ p_6(Tax Delinquency Records, {read}) to {TAP_1, TAP_2}; assign$_p$ p_7(Property ownership Records, {read}) to {TAP_1, TAP_2}; assign$_p$ p_8(Tax Exemption Records, {read}) to {DTP_1, DTP_2}; assign$_p$ p_9(Tax Sale Records, {read}) to {DTP_1, DTP_2}; assign$_p$ p_{10}(Redemption Invoice, {read,write,approve}) to {DTP_1, DTP_2}	
Periodicity constraints on role enabling	8	PE_{TAP1}: all.Days+ 9.Hours+1.Minutes$\triangleright$ 240.Minutes; $\Gamma(I_n, PE_{TAP1})$ = [08:00, 12:00] PE_{TAP2}: all.Days+ 15.Hours+1.Minutes$\triangleright$ 180.Minutes; $\Gamma(I_n, PE_{TAP2})$ = [14:00, 17:00] PE_{DTP1}: all.Days+ 9.Hours+1.Minutes$\triangleright$ 240.Minutes; $\Gamma(I_n, PE_{DTP1})$ = [08:00, 12:00] PE_{DTP2}: all.Days+ 15.Hours+1.Minutes$\triangleright$ 180.Minutes; $\Gamma(I_n, PE_{DTP2})$ = [14:00, 17:00]	(PE_{TAP1}, enable TAP_1) (PE_{TAP2}, enable TAP_2) (PE_{DTP1}, enable DTP_1) (PE_{DTP2}, enable DTP_2)
Role-activation constraint	9	([240min, 240min] active TAP_1); ([240min, 240min] active DTP_1); ([180min, 180min] active TAP_2); ([180min, 180min] active DTP_2)	

The run-time events allow an administrator or a user to request the activation or deactivation of a role. GTRBAC triggers are used to specify the dependence relationship among events. The expression for a trigger considered in this chapter has the following form: ev, sp1,…spk $\rightarrow$ ev', where ev is a simple event expression and spis are GTRBAC status predicates listed in Table 4.2. The event ev in the body of the trigger is called the triggering event and ev' is called the triggered event. We consider the triggered event ev' to be a role deactivation event. Note that the original GTRBAC model allows ev' to be role enabling or disabling event [78]. However, we assume that roles are automatically enabled during the specified time intervals. Therefore, defining triggers for role enabling or disabling event will violate this assumption.

Separation of duty (SoD) constraints, listed in Table 4.1, are used to prevent conflicting users from accessing same role concurrently or to prohibit conflicting roles from being accessed by same user at the same time. Although SoD constraints can be specified for user-role assignment, role enabling, and role activation, we consider SoDs that are specific to role activations only. We assume that the user-role assignment remains fixed throughout the policy life time and no periodic or temporal constraint is defined on such assignments, therefore, assignment-specific SoDs are not considered.

Example 4.2: Table 4.3 shows the GTRBAC policies of domains ID_{CTO} and ID_{CCO} collaborating for tax collection and payment processing. The roles of the ID_{CTO} include Tax Exemption Processor (TEP), Tax Refund Processor (TRP), and Tax Payment Processor (TPP). The user role assignments of the GTRBAC policy of ID_{CTO} are as follows: u_1 is assigned all three roles, u_2 is assigned the TPP role, and u_3 is assigned the TRP role. TEP is authorized for accessing the tax exemption records for verification and approval of *exemptions* claimed by property owners. TRP performs tax *refund adjustment* in the tax bills due for payment. For this purpose, TEP has appropriate authorization for accessing tax billing and refund records. TPP *processes payments* of adjusted tax bills by property owners. For processing such payments, TPP is assigned a read permission on tax billing records and read/write permission on tax payment records as shown in Table 4.3 TEP is enabled daily from 9:00am to 4:00 pm, while TRP and TPP are enabled from 10:00am to 2:00pm every day. For security reasons, ID_{CTO} does not allow a single user to perform both *exemption processing* and *payment processing* for the same property index. This constraint is defined as a role-specific separation of duty (SoD) constraint between TEP and TRP roles, prohibiting any user (in this case u_1) to activate both roles TEP and TRP simultaneously.

The GTRBAC policy of the domain ID_{CCO} includes four roles, namely: Tax Assessment Processor (TAP) and Delinquent Tax Processor (DTP). TAP is authorized to access tax delinquency and property ownership records for performing an *initial assessment* of tax redemption cost. DTP is responsible for preparing the *final estimate* for tax redemption. In addition, DTP also performs *clearance processing*. Both TAP and DTP can be enabled from 8:00am to 12:00pm and from 2:00pm to 5:00pm. Since we have assumed that a role can have only one enabling interval in a single calendar period, therefore both TAP and DTP are split into two roles, namely: TAP_1, TAP_2, DTP_1, and DTP_2. TAP_1 and DTP_1 are assigned the enabling interval of [8:00am, 12:00pm], whereas, TAP_2 and DTP_2 are assigned the enabling interval [2:00pm, 5:00pm].

4.3.3. State-based representation of GTRBAC policy

We model the GTRBAC policy of a domain as a timed graph introduced by Alur *et. al.*[4, 5]. Timed graphs have been widely used to characterize behavior of real-time systems having finite number of states. A timed graph is a directed graph consisting of a finite set of nodes, a finite set of edges, and a finite set of real-valued clocks. The following definition characterizes the state space of the FSM of GTRBAC policy.

Definition 4.5 [GTRBAC Timed graph]: *A GTRBAC timed graph is represented by a tuple* $TG = <S, SP, \mu, s_0, E, C, c_0, b_{max}, \gamma, \delta>$, *where*

- *S is a finite set of nodes representing GTRBAC states.*
- *SP denote the set of GTRBAC status predicates.* $SP=\{r\text{-}enabled(r) \mid r \in R\} \cup \{u\text{-}active(u, r) \mid u \in U, r \in R, \text{ and } u\text{-}assigned(u, r)\} \cup \{trg\text{-}enabled(tg) \mid tg \text{ is a trigger in GTRBAC policy}\}$.

- *μ: $S \rightarrow \mathbf{A} \subseteq 2^{SP}$ is a labeling function assigning to each state the set of status predicates that are true in that state. Where, $\mathbf{A}$ is the maximal subset of 2^{SP} such that predicate assignment $a \in \mathbf{A}$ satisfies all the GTRBAC constraints listed in Table 4.4.*
- *$E \subseteq S$ x S is a set of edges. The edges represent the events causing the domain to move from one GTRBAC state to another.*
- *$s_0 \in S$ is the initial state and $s_{reset} \in S$ is the calendar clock reset state. In state s_0 and s_{reset} all roles are disabled. For all state $s \in S - \{s_0, s_{reset}\}$, $(s, s_{reset}) \notin E$.*
- *C is a finite set of clocks.*
- *c_0 is a calendar clock which is reset with the occurrence of clock reset event represented by the edge from s_0 to s_{reset}.*
- *$b_{max} = \max\limits_{r \in \mathrm{R}} \{b_r\}$, where b_r is the end point of the interval during which role r is enabled.*
- *γ is a function labeling each edge with an enabling condition of the form $(d_1 \leq \mathrm{c}_0 \leq d_2) \bigwedge\limits_{x \in C'} (d_{1x} \leq x \leq d_{2x})$, where $C' \subseteq C$ and $d_1, d_2, d_{1x}, d_{2x} \in Z^+$ with $d_1 \leq d_2 \leq b_{max}$ and $d_{1x} \leq d_{2x} < b_{max}$. For the edge e_{reset} from s_0 to s_{reset}, $\gamma(e_{reset}) = b_{max} \leq c_0 \leq b_{max}$, and for the edge e_0 from s_{reset} to s_0, $\gamma(e_0) = 0 \leq c_0 \leq 0$.*
- *δ: $E \rightarrow 2^C$ is a function mapping an edge to a set (possibly an empty set) of clocks that are reset with the edge. The function δ maps the edge e_{reset} from s_0 to s_{reset} to c_0, i.e., $\delta(e_{reset}) = c_0$.*

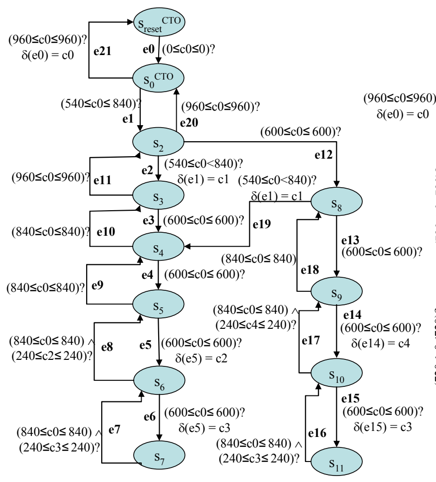
A node in a GTRBAC timed graph models the access control state of a domain characterized by the status predicates true in that state. All states in *S* satisfy the GTRBAC policy constraints including separation of duty constraints, dependence constraint between role enabling and role activation, dependence constraint between role assignment and activation, and trigger enabling constraint. These constraints are listed Table 4.4. Edges in the GTRBAC timed graph represent the state transition events, which are listed in Table 4.3. Each edge is labeled with an enabling condition defined using clock values. At any point in time, the domain can make a transition from its current state s_i to a next state s_j, if the enabling condition associated with the edge (s_i, s_j) is satisfied by the current values of clock. A clock can be reset with any state transition. At any instant, the value of a clock is equal to the time elapsed since the last time the clock was reset. Each edge in the GTRBAC timed graph is mapped to a set (possibly an empty set) of clocks that are reset when the corresponding state transition event occurs. In states s_0 and s_{reset} all roles are disabled. The state s_{reset} is visited when no role can be enabled during the current calendar period. By visiting state s_{reset}, the calendar clock c_0 of a domain is reset/initialized to the starting point of next calendar period in which the enabling and activation of roles follow the pattern of previous calendar periods.

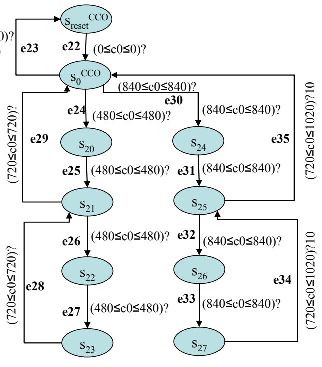
The procedure for generating the FSM of a GTRBAC policy in a timed graph representation is depicted in Fig. 4.3. This procedure first generates the state space of the given GTRBAC policy by considering all valid status predicate assignments that satisfy the GTRBAC constraints listed in Table 4.4. After generating the state space, the state transitions in the GTRBAC timed graph are defined by creating the edge set *E*. For all pairs of GTRBAC states s_i and s_j, an edge is created from s_i to s_j only if there exists a GTRBAC event ev_{ij} such that s_i satisfies all the precondition of ev_{ij} and s_j satisfies the post conditions of ev_{ij}. Next the edges in the set *E* are labeled with appropriate enabling conditions and clock reset function.

The timed graph of the GTRBAC policy of the ID_{CTO} generated by this procedure is shown in Fig. 4.4(a). The initial state of this timed graph is s_0^{CTO} and the calendar clock reset state is s_{reset}^{CTO}. The status predicates that are true in the GTRBAC states of Fig. 4.4(a) are tabulated in Fig. 4.4(c). The events corresponding to the edges of Fig. 4.4(a) and Fig. 4.4 (b) are listed in Fig. 4.4(e). Each edge is labeled with an enabling condition defining the timing constraints for the corresponding GTRBAC event. For instance, the edge e2, representing the event *activate* TEP for u_1, is labeled with the enabling condition '$540 \leq c_0 \leq 840$'. This enabling condition implies that the TEP role can be activated by u_1 from the state s_2 within an interval of [540, 840] minutes. This interval is defined with respect to the calendar clock c_0 which is initialized (reset) in state s_{reset}^{CTO}.

```
GTRBAC-FSM
INPUT: GTRBAC Policy Instance
OUTPUT: Timed graph
1. Let SP={r-enabled($r$)| $r \in$ R}∪{u-active($u, r$)| $u \in$ U, $r \in$ R, and u-assigned($u$,
   $r$)}∪{trg-enabled($tg$)| $tg$ is a trigger in GTRBAC policy}
2. Let A be a maximal subset of $2^{SP}$ such that each $a \in$ A satisfies all the GTRBAC constraints listed
   in Table 4.
3. Generate a set of states S such that |S| = |A| + 1.
4. Create a one-to-one mapping $\mu$ from states in the set S-{$s_{reset}$} to predicate assignments in the set A.
   Name the state that is mapped to an assignment in which all roles are disabled as $s_0$. Let $\mu(s_{reset})$ =
   $\mu(s_0)$.
5. For each pair of states $s_i$ and $s_j$, if there is a GTRBAC event $ev$ that can cause state transition from
   $s_i$ and $s_j$, then create an edge $e_{ij}$ from $s_i$ and $s_j$.
      a. If $ev$ is an enabling event of a role $r$ then define the following mapping:
         $\gamma(e_{ij}) = a_r \le c_0 \le a_r$
      b. If $ev$ is a disabling event of role $r$ then define the following mapping:
         $\gamma(e_{ij}) = (b_r \le c_0 \le b_r)$
      c. If $ev$ an activation event of role $r$ by user $u$ then define the following mappings:
         $\gamma(e_{ij}) = a_r \le c_0 \le (b_r - d_r^{min})$
         $\delta(e_{ij}) = c_{ur}^{act}$, where, $c_{ur}^{act} \in C$ is a clock that measure the time elapsed since activation of
         role $r$ by $u$.
      d. If $ev$ is a deactivation event of role $r$ by user $u$ then define the following mapping:
         $\gamma(e_{ij}) = (a_r + d_r^{min} \le c_0 \le b_r) \wedge (d_r^{min} \le c_{ur}^{act} \le d_r^{max})$
      e. For each GTRBAC trigger $tg$: $ev_1, sp_1, \ldots sp_k \rightarrow$ deactivate $r$ for $u$, such that t-
         enabled($tg$) $\in \mu(s_i)$, t-enabled($tg$) $\notin \mu(s_j)$, and ($ev$ == deactivate $r$ for $u$),
         then $\gamma(e_{ij}) = et(s_i)$.
6. Delete all edges $e_{ij}$ corresponding to some event $ev$, if there exists a GTRBAC trigger $tg$: $ev_1$,
   $sp_1, \ldots sp_k \rightarrow$ deactivate $r$ for $u$, such that one of the following holds:
      a. t-enabled($tg$) $\in \mu(s_i)$ and t-enabled($tg$) $\in \mu(s_j)$.
      b. t-enabled(tg) $\in \mu(s_i)$ and ($ev \ne$ deactivate $r$ for $u$).
      c. ($ev \ne ev_1$) and t-enabled(tg) $\in \mu(s_j)$.
7. Create an edge $e_0$ from $s_{reset}$ to $s_0$ and define the following mapping:
   $\gamma(e_0) = 0 \le c_0 \le 0$
8. Create an edge $e_{reset}$ from $s_0$ and $s_{reset}$ and define the following mapping:
   $\gamma(e_{reset}) = b_{max} \le c_0 \le b_{max}$
   $\delta(e_{reset}) = c_0$
```

Fig. 4.3 Procedure for generating the timed graph of a GTRBAC policy





States	*enable*(TEP)	*enable*(TPP)	*enable*(TRP)	*u-activee*(u_1,TEP)	*u-activee*(u_1,TPP)	*u-activee*(u_2,TPP)	*u-activee*(u_3,TRP)
s_0^{CTO}, s_{reset}^{CTO}	0	0	0	0	0	0	0
s_2	1	0	0	0	0	0	0
s_3	1	0	0	1	0	0	0
s_4	1	1	0	1	0	0	0
s_5	1	1	1	1	0	0	0
s_6	1	1	1	1	0	1	0
s_7	1	1	1	1	0	1	1
s_8	1	1	0	0	0	0	0
s_9	1	1	1	0	0	0	0
s_{10}	1	1	1	0	1	0	0
s_{11}	1	1	1	0	1	0	1

States	*enable*(TAP_1)	*enable*(DTP_1)	*enable*(TAP_2)	*enable*(DTP_2)	*u-activee*(u_4,TAP_1)	*u-activee*(u_5,DTP_1)	*u-activee*(u_4,TAP_2)	*u-activee*(u_5,DTP_2)
s_0^{CCO}, s_{reset}^{CCO}	0	0	0	0	0	0	0	0
s_{20}	1	0	0	0	0	0	0	0
s_{21}	1	0	0	0	1	0	0	0
s_{22}	1	1	0	0	1	0	0	0
s_{23}	1	1	0	0	1	1	0	0
s_{24}	0	0	1	0	0	0	0	0
s_{25}	0	0	1	0	0	0	1	0
s_{26}	0	0	1	1	0	0	1	0
s_{27}	0	0	1	1	0	0	1	1

Edges	**Event**	**Edges**	**Event**	**Edges**	**Events**
e0	New Period	e10	*disable* TPP	e26	*enable* DTP_1
e1,	*enable* TEP	e11	*deactivate* TEP for u_1	e27	*activate* DTP_1 for u_5
e2,e19	*activate* TEP for u_1	e14	*activate* TPP for u_1	e28	*disable* DTP_1
e3,e12	*enable* TPP	e17	*deactivate* TPP for u_1	e29	*disable* TAP_1
e4,e13	*enable* TRP	e20	*disable* TEP	e30	*enable* TAP_2
e5	*activate* TPP for u_2	e21	Reset event for c0 in CTO	e31	*activate* TAP_2 for u_4
e6,e15	*activate* TRP for u_3	e22	New Period	e32	*enable* DTP_2
e7,e16	*deactivate* TRP for u_3	e23	Reset event for c0 in CTO	e33	*activate* DTP_2 for u_5
e8	*deactivate* TPP for u_2	e24	*enable* TAP_1	e34	*disable* DTP_2
e9,e18	*disable* TRP	e25	*activate* TAP_1 for u_4	e35	*disable* TAP_2

Fig. 4.4. (a) Timed graph of the GTRBAC policy of IDCTO. (b) Timed graph of the GTRBAC policy of IDCCO. (c) GTRBAC status predicate assignment to the states of Fig. 4.4(a). (d) GTRBAC status predicate assignment to the states of Fig. 4.4(b). (e) Events associated with the edges of Fig. 4.4(a) and 4(b).

Table 4.4
Constraints on all valid status predicate assignment to GTRBAC states

SP={r-enabled(r)| $r \in$ R}∪{u-active(u, r)| $u \in$U, $r \in$R, and u-assigned(u, r)}∪{trg-enabled(tg)| tg is a trigger in GTRBAC policy} $\forall a \in$ A such that A is a maximal subset of 2^{SP}, the following constraints must be satisfied:		
	Constraints	Meaning
1	u-active(u, r) $\in a \Rightarrow$ r-enabled(r) $\in a$	A disabled role cannot be activated by any user in any GTRBAC state.
2	$\forall r$, $r' \in$ R such that $\Gamma(I_n, \text{PE}_r) \cap \Gamma(I_n, \text{PE}_{r'}) = \phi$, r-enabled($r$) $\in a \Rightarrow$ r-enabled(r') $\notin a$	Two roles with disjoint enabling intervals cannot be enabled simultaneously in any GTRBAC state.
3	$\forall r$, $r' \in$ R such that $\Gamma(I_n, \text{PE}_r)=[a_r, b_r]$, $\Gamma(I_n, \text{PE}_{r'}) = [a_{r'}, b_{r'}]$, and $a_{r'} \le a_r \le b_r \le b_{r'}$, r-enabled($r$) $\in a \Rightarrow$ r-enabled(r') $\in a$	If the enabling interval of r is contained in the enabling interval of r', then in any GTRBAC state in which r is enabled, r' is also enabled.
4	$\forall$ r, $r' \in$ R-SoD(u), u_active(u, r) $\in a$ $\Rightarrow$ u_active(u, r') $\notin a$	In any GTRBAC state, a user u can activate at most one role in R-SoD(u).
5	$\forall$ u, $u' \in$ U-SoD(r), u_active(u, r) $\in a$ $\Rightarrow$ u_active(u', r) $\notin a$	In any GTRBAC state, at most one user in the set U-SoD(r) can activate r.
For any trigger tg: ev_1, sp_1,...sp_k → deactivate r for u		
6	trg-enabled(tg) $\in a \Rightarrow \bigwedge_{i=1}^{k} sp_i \in a \wedge$ u-active(u, r) $\in a$ $\wedge$[(u-active(u', r') $\in a$ if ev_1==activate r' for u') $\vee$ (u-active(u', r') $\notin a$ if ev_1==deactivate r' for u') $\vee$ (r-enable(r') $\in a$ if ev_1==enable r') $\vee$ (r-enable(r') $\notin a$ if ev_1==disable r')]	Trigger tg is enabled only if the event ev_1 occurs and at the time of occurrence of ev_1, all status predicates defined in the triggers body hold. After enabling of trigger tg, the only event that can occur is deactivate r for u.

4.3.4. Definition of state path and timing constraints

In this section, we define state path and the state timing constraints. These timing constraints are used to determine the composibility of a given distributed workflow with respect to the FSM of domains as discussed in Section 4.4.

Definition 4.6 (State Path): *A state path π is a sequence $s_1.e_1.s_2.e_2\ldots\ldots,e_{n-1}.s_n$, $n > 0$, such that the symbol s_i $(1 \le i \le n)$ in path π denotes a GTRBAC state, and the* *symbol e_j ($1 \le j \le n-1$) denotes an edge in the timed graph of GTRBAC policy. The edge e_j represents the event that causes a transition in the GTRBAC system from state s_j to s_{j+1}.*

Definition 4.7 (State Entry Time): *The time instant at which a GTRBAC state, say s_j, can be visited is called the entry time of state s_j and is denoted by $et(s_j)$. The entry time of a state is measured relative to the domain's calendar clock c_0, which is initialized and reset in state s_{reset} only.*

For computing $et(s_j)$, we need to determine the enabling and activation times of all roles that are enabled and active in state s_j. Let R_j^{en} and R_j^{act}, respectively, denote the set of roles that are enabled and active in state s_j. The following constraint defines an upper and lower bound on value of $et(s_j)$.

$$\max_{r \in R_i^{en}} \{a_r\} \le et(s_j) \le \max_{r'' \in R_i^{act}} \{b_{r''} - d_{r''}^{\min}\},$$

where, $[a_r, b_r]$ is the enabling interval of role r, and d_r^{min} is the minimum duration for which r can be activated by any user.

Definition 4.8 (State Residence Time): *The time a domain stays in a particular GTRBAC state, say s_j, in a state path, say π, is called the residence time of state s_j in π.*

Let t^{π}_{sj} denote the residence time of state s_j in path π: $s_1.e_1\ldots\ldots e_{j-1}s_j.e_js_{j+1}\ldots,e_{n-1}.s_n$. Suppose $\gamma(e_j) = \bigwedge_{x \in C'} d_{1x} \le x \le d_{2x}$ is the enabling condition for the event represented by edge e_j, where $C' \subseteq C$ and $c_0 \in C'$. For a clock $x \in C'$, let e_{j-kx} be an edge in π such that $x \in \delta(e_{j-kx})$, (i.e., clock x is reset at the edge e_{j-kx}) and there is no other edge e_p between e_{j-kx} and e_j in π for which $x \in \delta(e_p)$. The following inequalities provides a bound on the residence time t^{π}_{sj} with respect to the residence time of the predecessor states of s_j in path π.

$$\forall x \in C',\ d_{1x} \le \sum_{p=0}^{kx} t^{\pi}_{sj-p} \le d_{2x}$$

We refer to the above inequality as residence time constraint. Note that in the GTRBAC timed graph definition, the enabling condition for each event is defined with respect to the calendar clock c_0 of the domain, which is reset when the domain make a transition from state s_0 to s_{reset} by traversing the edge e_{reset}. However, the edge e_{reset} may not be included in the state path π. To ensure that a valid residence time constraint can be defined for each state in π, we concatenate a dummy path π_d: $s_{d1}.e_{d1}.s_{d2}.e_{d2}$ to the beginning of π, where $\delta(e_{d1}) = c_0$ and $\gamma(e_{d2}) = et(s_1)$. It can be easily proved that the entry time of all states in π remains unchanged with the concatenation of path π_d. The main reason for this concatenation is that the calendar clock c_0 is initialized just before the first state of π is visited, therefore, the residence time constraint can be defined for all states in π.

Definition 4.9 (Traversal time of a state path): *The traversal time of a state path π is defined as the sum of the residence times of all states included in π.*

Given a state path π: $s_1.e_1......e_{j-1}s_j.e_js_{j+1}....,e_{n-1}.s_n$, we can compute its minimal or maximal traversal time using the procedure given in Fig. 4.5. This minimal and maximal value for state path traversal is used to determine if the given state path π satisfies the duration and temporal constraints associated with the *component service*s as discussed in Section 4.4.

```
path-traversal-time(π)
1. π' ← π_d. π, where π_d is a dummy path s_d1.e_d1.s_d2.e_d2 with δ(e_d1) = c_0 and
   γ(e_d2) = et(s_1).
2. p ← index(first-state(π')) and q ← index(last-state(π'))
3. for i ← p to q
4.     do define the residence time constraints for state s_i and add it to the set of
   equations/inequalities for path π'.
5. Solve the system of residence time constraint generated in steps 2 and 3 for minimal
   or maximal value of $\sum_{i=p}^{q} t^{\pi'}_{s_i}$
```

Fig. 4.5 Procedure for computing the minimum or maximal residence time of states in a state path.

In Section 4.5.1, we present an algorithm for verifying the correctness of a PW with respect to the GTRBAC policy of a domain. This algorithm iteratively discovers all state paths with traversal time less than a given threshold value between a given pair of sates. To discover such paths, we need to have *a priori* information about the residence time of all the states in the corresponding domain's FSM. For this purpose, we define a minimum residence time graph (MRTG) which is generated from the GTRBAC timed graph.

Definition 10 (Minimum Residence Time Graph): *A minimum residence time graph (MRTG) of a domain is a tuple MRTG = <S, E, w>, where, S and E respectively denote the sets of states and edges defined in the GTRBAC timed graph, and w is a weight function that maps each edge in the set E to a non-negative real number. For an edge e_j from state s_j to s_{j+1}, $w(e_j)$ denotes the minimum time the domain stays in GTRBAC state s_j before moving to the next state s_{j+1}.*

For computing $w(e_j)$, we evaluate the minimum residence time of state s_j over all over all possible state paths passing through edge e_j. Fig. 4.6 shows a procedure for determining the weight *w* for each edge in MRTG.

```
MRTG-Edge-Weights
1.Set $w(e_j) \leftarrow \infty$ for all $e_j \in E$.
2.For each pair of states $s_p, s_q \in S$ $(p \neq q)$, find a set of all simple paths $\prod_{pq}$ from $s_p$
  to $s_q$.
3.For each $\pi$ in $\prod_{pq}$ compute the minimum traversal time of $\pi$. Let $t^{\pi}_{ej}$ denote the
  residence time of state $s_j$ such that $s_j$ is connected to its successor state $s_{j+1}$ in $\pi$
  by the edge $e_j$.
    if $t^{\pi}_{ej} < w(e_j)$, then  $w(e_j) \leftarrow t^{\pi}_{ej}$
4.Repeat step 3 for all state pairs $s_p, s_q \in S$
```

Fig. 4.6 Procedure for determining the edge weights in a MRTG

Note that the length of any path in the minimum residence time graph defines a lower bound on the traversal time of the corresponding path in the GTRBAC timed graph. Therefore, the set of all MRTG paths between state nodes s_i and s_j that are shorter than a given threshold value T includes all state path from s_i to s_j with traversal time less than T.

4.4. Composibility Verification

In this section, we describe the proposed approach for verification of workflow composibility. For verifying secure composibility of a distributed workflow, the correctness of the workflow specification needs to be evaluated against the individual as well as the collective behavior of all collaborating domains. This requirement provides a general guideline for analyzing the composibility of a given distributed workflow. In Section 4.4.1, we translate this requirement into a set of *workflow composibility* conditions against which the correctness of the distributed workflow is evaluated.

We use a two-step approach for verifying secure workflow composibility. In the first step, the distributed workflow specifications are analyzed for conformance with the security and access control policy of each collaborating domain. In the next step, the *cross-domain dependencies* amongst the *component services* of the workflow are verified. We use the term *cross-domain dependency* to refer to the precedence relationship between *component services* of the workflow that are provided by different domains. For instance, in Fig. 4.2 there is a cross-domain dependency between the *final estimate* preparation service provided by ID_{CCO} and the *redemption payment* processing service provided by ID_{CTO}. For a given distributed workflow, the set CS_{dep} defined in Table 4.5 captures all the cross-domain dependencies of the workflow.

Table 4.5
Symbols and notations used in defining workflow composibility conditions

Symbol/Notation	Description
PW_i	Projected workflow assigned to domain ID_i
CS_i	Set of *component service*s provided by domain ID_i
CS_{dep}	Set of all cross-domain *component service*s that have a precedence relationship. $CS_{dep} = \{(c_q , c_r) \mid c_q \in CS_i, c_q \in CS_j \ (i \neq j)$, and c_q precedes c_r in the execution order of the distributed workflow$\}$
π_i	State path of domain ID_i that satisfies WC1, WC2, and WC3 for PW_i
$\Pi^{(i)}$	Set of all paths that satisfies conditions WC1, WC2, and WC3 for the projected workflow PW_i
$T_{\min}^{q,q+1} \, (T_{\max}^{q,q+1})$	Minimum (maximum) time between completion of intra domain *component service*s c_q and c_{q+1}
$\phi_q^{\pi I} \ (\theta_q^{\pi i})$	Initiation (completion) time of *component service*s c_q in path π_i
$[\min(\phi_q^{\pi i}), \max(\phi_q^{\pi i})]$	Time interval during which *component service*s c_q can be initiated in path π_i
$[\min(\theta_q^{\pi i}), \max(\theta_q^{\pi i})]$	Time interval during which *component service*s c_q can be completed in path π_i
Δ_i	Duration of the smallest calendar period that contains the enabling intervals of all role of ID_i
$CS^{(i)}_{init}$	Time interval during which each *component service* of domain ID_i can be initiated. $CS^{(i)}_{init}=\{[\min(\phi_q^{\pi}), \max(\phi_q^{\pi})] \mid c_q \in CS_j$ and $\pi \in \Pi^{(i)}\}$
$CS^{(i)}_{end}$	Time interval during which each *component service* of domain ID_i can be completed. $CS^{(i)}_{end}=\{[\min(\theta_q^{\pi}), \max(\theta_q^{\pi})] \mid c_q \in CS_j$ and $\pi \in \Pi^{(i)}\}$

The overall process of workflow composibility verification is depicted in Fig. 4.1. In this process, first a projected workflow (PW) specification is generated from the distributed workflow specification for all domains. The PW specification of a domain is represented in form of a task graph as discussed in Section 4.2. Next a mapping is established between each task of the PW and the user-role activation required for execution of the corresponding task. After establishing the semantic mapping, a state-based representation of a PW is generated by mapping the GTRBAC based specification of the PW to all valid state paths that satisfy all the constraints included in the PW specification. The procedure for state mapping is given in Section 4.5.1. Mapping of a PW to valid state paths verifies the consistency of the distributed workflow with respect to the access control policy of the corresponding domain. However, this PW to state path mapping does not imply that the domain can satisfy the cross-domain dependency constraints amongst the *component service*s of the distributed workflow. For this purpose, all combinations of valid state paths from all domains are analyzed for satisfaction of cross-domain dependencies. In Sections 4.4.2 and 4.2.2, we discuss how the state paths from different domains are verified for preservation of cross-domain dependencies among the *component services.*

4.4.1. Workflow composibility conditions

In this section, we specify the criteria for workflow composibility verification in a formal manner. In particular, we define a set of conditions against which a distributed workflow specification is evaluated. We classify these conditions as *intra-domain* and *inter-domain workflow composibility conditions*.

4.4.1.1. Intra-domain workflow composibility conditions

The intra-domain workflow composibility conditions are used to verify domain-specific projected workflow for conformance with the local GTRBAC policy of the domain. Let TG_A denote the timed graph of the GTRBAC policy of domain ID_A, and PW_A be the task graph corresponding to the projected workflow of ID_A. For a task $\tau_i \in PW_A$, let $\phi_i^{\pi'}$ denotes the time instant at which the processing of τ_i is initiated. We say PW_A is consistent with respect to TG_A if there exist a state path $\pi' = (\pi_d).(\pi) = (s_{d1}e_{d1}s_{d2}e_{d2}).(s_1e_1s_2\ldots.e_{n-1}s_n)$, such that $\delta(e_{d1}) = c_0$, $\gamma(e_{d2}) = et(s_1)$, for all $k < n$, $(s_k, s_{k+1}) \in E$ and the following conditions hold:

WC1. For each task $\tau_i \in PW_A$, there exists a sub-path, $\pi_i = s^i_1e^i_1s^i_2\ldots.\ e^i_{m-1}s^i_m$, of π that satisfies the following properties:

a. $index(s^i_{k+1}) = index(s^i_k) + 1$ for $0 < k < m$, where the function $index(s)$ returns the index of state s in the sequence π.

b. There exists $(u, r) \in DRM(\tau_i)$ such that role r is active for user u in all the states of the sub-sequence π_i. As discussed in Section 4.3.1, the function $DRM(\tau_i)$ maps τ_i to a set of user role activation pair (u', r') such that each r' has the required permissions for processing task τ_i and u' is authorized for r'.

$$t_{s\mathrm{d}2} + \sum_{k=1}^{p-1} t_{sk} \le \phi_i^{\pi'} \le \sum_{k=1}^{q} t_{sk} - duration(\tau_i)$$

Where $p = index(\mathrm{s}^i{}_1)$, $q = index(\mathrm{s}^i{}_m)$, and t_{sk} is the residence time of state s_k in path π and $ts_{\mathrm{d}2}$ is the residence time of state $s_{\mathrm{d}2}$ in π' (see Section 4.3.3).

WC2. For any pair of tasks τ_i and τ_j of $\mathrm{PW_A}$ such that τ_i precedes τ_j in the task execution order, the time associated with the completion of task τ_i is less than or equal to the time associated with the initiation of task τ_j. Formally:

$$\phi_i^{\pi'} + duration(\tau_i) \le \phi_j^{\pi'},$$

WC3. For any pair of tasks τ_i and τ_j of $\mathrm{PW_A}$ such that τ_i precedes τ_j in the task execution order with the temporal constraint requiring the delay between the completion of τ_i and initiation of τ_j to be bounded by the interval $[\mathrm{T_{min}}, \mathrm{T_{max}}]$, the following inequalities must hold:

$$\phi_i^{\pi'} + duration(\tau_i) \le \phi_j^{\pi'}$$

$$\mathrm{T_{min}} \le \phi_j^{\pi'} - (\phi_i^{\pi'} + duration(\tau_i)) \le \mathrm{T_{max}}.$$

Intuitively, the first condition (WC1) implies that for a PW to conform with the corresponding domain's policy, at least one state path must exist that satisfies duration constraints of each task of the PW. The workflow composibility conditions WC2 and WC3 imply that such state path must also satisfy the temporal constraints between all task pairs of the PW. These constraints include the precedence relationship and the timing constraints between task pairs of a PW as discussed in Section 4.2.2.

4.4.2. Inter-Domain workflow composibility condition

The *inter-domain workflow composibility condition* defines the criterion for evaluating the correctness of workflow specification with respect to the collective behavior of all collaborating domains. In particular, this condition stipulates that the cross-domain dependencies among the *component services* in a distributed workflow specification needs to be satisfied. For this purpose, the state paths of all collaborating domains that satisfy the intra-domain workflow composibility conditions (WC1, WC2, and WC3) need to be analyzed for satisfaction of cross-domain dependencies. This analysis requires comparing the earliest and latest initiation/completion time of inter-domain *component services* or *tasks* that are involved in cross-domain dependencies. As mentioned in Section 4.2, a *component service* may itself be a workflow process comprising multiple tasks. The initiation time of a *component service* corresponds to the initiation time of the first task of the workflow associated with the *component service*. Similarly, the completion time of a *component service* is the completion time of the last/final task of the *component service* workflow.

With reference to the state path $\pi' = (\pi_d).(\pi) = (s_{d1}e_{d1}s_{d2}e_{d2}).(s_1e_1s_2 \ldots e_{n-1}s_n)$ considered in Section 4.4.1.1, The expression $t_{sd2} + \sum_{k=1}^{p-1} t_{sk} \leq \phi_i^{\pi'} \leq \sum_{k=1}^{q} t_{sk} - duration(\tau_i)$ defines a range of values for the initiation time of each task τ_i in π' with respect to the calendar clock of the domain performing this task. We assume that the calendar clocks of all collaborating domains are synchronized at the time their policy instances are created. Note that this assumption does not restrict domains to have different periods for resetting of their calendar clocks. For instance the calendar clock of one domain may reset on a daily basis, whereas the calendar clock of another may reset on a weekly basis. With this assumption, the calendar clock values of all domains can be compared and so the cross-domain dependencies amongst the *component service*s can be verified based on the timing information provided by the domains. This timing information includes the earliest and latest time for initiation/completion of tasks that are involved in cross-domain dependencies. Fig. 4.7 depicts the procedure for computing earliest and latest initiation times for each task of the PW. The earliest and latest completion time of any task τ can be easily computed by adding *duration*(τ) to the task initiation time values returned by the procedure.

```
task-initiation-time
INPUT: $\pi' = (\pi_d).(\pi) = (s_{d1}e_{d1}s_{d2}e_{d2}).(s_1e_1s_2....e_{n-1}s_n)$, such that $\delta(e_{d1}) = c_0$, $\gamma(e_{d2}) = et(s_1)$
       PW
OUTPUT: $\min(\phi_i^{\pi'})$ for each $\tau_i \in$ PW
        $\max(\phi_i^{\pi'})$ for each $\tau_i \in$ PW
1  Generate a system of linear inequalities by adding:
   a  residence time constraints for each state included in $\pi'$.
   b  task initiation time constraint (composibility condition WC1) for each $\tau_i \in$ PW
   c  precedence constraint between all pairs of tasks $\tau_i, \tau_j \in$ PW such that $\tau_i$
      precedes $\tau_j$ in the task execution order with the temporal constraint
      requiring the delay between the completion of $\tau_i$ and initiation of $\tau_j$ to be
      bounded by the interval $[T_{min}, T_{max}]$ (composibility conditions WC2 and WC3).
2  Solve the system of constraints generated in step 1 with the objective of
   minimizing $\Sigma_i\phi_i$. The value assigned to each $\phi_i$ equals $\min(\phi_i^{\pi'})$.
3  Solve the system of constraints generated in step 1 with the objective of
   maximizing $\Sigma_i\phi_i$. The value assigned to each $\phi_i$ equals $\max(\phi_i^{\pi'})$.
```

Fig. 4.7. Procedure for computing the earliest and latest initiation time of each task in a projected workflow.

Proposition 4.1: Given a task $\tau_i \in$ PW, the time $\min(\phi_i^{\pi'})$ ($\max(\phi_i^{\pi'})$) computed using the task initiation time procedure is the earliest (latest) time at which the task τ_i can be initiated in state path π' that satisfies intra-domain workflow composibility conditions WC1, WC2, and WC3 for PW.

Proof of this proposition is provided in the Appendix B.

The cross-domain dependencies amongst the *component service*s can be represented in an algebraic form based on the task initiation and completion information provided by collaborating domains. The notations and symbols used in this representation of cross-domain dependencies are listed in Table 4.5.

For any pair of cross-domain *component service*s c_q and c_r such that $(c_q, c_r) \in CS_{dep}$, the following set of inequalities captures both the intra-domain and cross-domain dependency constraints between c_q and c_r.

$$v\Delta_i + \theta_q^{\pi i} \le w\Delta_j + \phi_y^{\pi j} \qquad (v, w \in Z^+) \qquad \text{(I)}$$

$$\min(\theta_q^{\pi i}) \le \theta_q^{\pi i} \le \max(\theta_q^{\pi i}) \qquad \text{(II)}$$

$$\min(\phi_q^{\pi i}) \le \phi_q^{\pi i} \le \max(\phi_q^{\pi i}) \qquad \text{(III)}$$

$$\min(\theta_r^{\pi j}) \le \theta_r^{\pi j} \le \max(\theta_r^{\pi j}) \qquad \text{(IV)}$$

$$\min(\phi_r^{\pi j}) \le \phi_r^{\pi j} \le \max(\phi_r^{\pi j}) \qquad \text{(V)}$$

$$T_{\min}^{q,q+1} \le \phi_q^{\pi i} - \theta_q^{\pi} i \le T_{\max}^{q,q+1} \qquad \text{(VI)}$$

$$T_{\min}^{r-1,r} \le \phi_r^{\pi j} - \theta_r^{\pi j} \le T_{\max}^{r-1,r} \qquad \text{(VIII)}$$

Constraint (I) implies that the *component service c_q* must be completed before c_r is initiated in any calendar period. The variables $\theta_q^{\pi i}$and $\phi_r^{\pi j}$ denote the completion and initiation times of *component service*s c_q and c_r in state paths π_i and π_j respectively. The bounds on these two variables are specified in constraints (II) - (V) given above. Constraints (VI) and (VII) specify timing constraints between the intra-domain *component service*s. These timing constraints are computed while generating PW specification as discussed in Section 4.2.

If the solution set of the above system of inequalities generated for all $(c_q , c_r) \in CS_{\text{dep}}$ is non-empty then the state paths π_i and π_j jointly satisfy all the cross-domain dependencies between PW$_i$ and PW$_j$. Based on this implication, the following condition for verifying the workflow composibility with respect to temporal dependencies among the *component service*s can be defined.

WC4. Two state paths π_i and π_j, respectively satisfying conditions WC1, WC2, and WC3 for the projected workflows assigned to ID$_i$ and ID$_j$, are consistent if they satisfy all the cross-domain dependencies included in the set CS$_{\text{dep}}$.

4.4.3. Overall criteria for workflow composibility

Based on the *intra-domain* and *inter-domain workflow composibility* conditions, we provide the following overall criteria for workflow composibility.

Given a distributed workflow **S**, a set of **S**'s projected workflows **PW** = {PW$_1$,…,PW$_n$}, a set of cross-domain dependencies among *component service*s **CS$_{\text{dep}}$** =

$\{(c_i^q, c_j^r) | c_i^q$ precedes c_j^r and $1 \leq i,j \leq n$ and $i \neq j\}$, and a set **F** of FSMs, modeling domains' GTRBAC policies. Let $\Pi^{(i)}$ denote the set of state paths of ID$_i$ such that each path in $\Pi^{(i)}$ satisfies workflow composibility conditions WC1, WC2, and WC3 for PW$_i$. We say that **S** is composable with respect to **F** if the following hold:

- For any ID$_i$, the set of paths $\Pi^{(i)}$ is non-empty.
- There exists a tuple $(\pi_1, \pi_2, \ldots, \pi_n) \in \Pi^{(1)} \times \Pi^{(2)} \times \ldots \times \Pi^{(n)}$ such that $(\pi_1, \pi_2, \ldots, \pi_n)$ satisfy all cross-domain dependencies $(c_i^q, c_j^r) \in$ **CS$_{dep}$**, where $1 \leq i,j \leq$ n.

Table 4.6
Constraints on the initiation and completion times of component services imposed by state paths π_1 and π_2.

Δ_{CTO}	1440
$[\min(\phi_{EP}^{\pi1}), \min(\phi_{EP}^{\pi1})]$	[540, 740]
$[\min(\phi_{RA}^{\pi1}), \min(\phi_{RA}^{\pi1})]$	[620, 800]
$[\min(\theta_{EP}^{\pi1}), \min(\theta_{EP}^{\pi1})]$	[560, 760]
$[\min(\theta_{PP}^{\pi1}), \min(\theta_{PP}^{\pi1})]$	[680, 840]
$\phi_{RA}^{\pi1}$-$\theta_{EP}^{\pi1}$	≥ 60 and ≤235

Δ_{CCO}	1440
$[\min(\phi_{IA}^{\pi2}), \min(\phi_{IA}^{\pi2})]$	[480, 585]
$[\min(\phi_{FE}^{\pi2}), \min(\phi_{FE}^{\pi2})]$	[505, 610]
$[\min(\phi_{CP}^{\pi2}), \min(\phi_{CP}^{\pi2})]$	[605, 710]
$[\min(\theta_{IA}^{\pi2}), \min(\theta_{IA}^{\pi2})]$	[485, 590]
$[\min(\theta_{FE}^{\pi2}), \min(\theta_{FE}^{\pi2})]$	[565, 670]
$[\min(\theta_{CP}^{\pi2}), \min(\theta_{CP}^{\pi2})]$	[615, 720]
$\phi_{FE}^{\pi2}$-$\theta_{IA}^{\pi2}$	≥ 20 and ≤55
$\phi_{CP}^{\pi2}$-$\theta_{FE}^{\pi2}$	≥ 40 and ≤170

Example 4.3: Consider the projected workflows PW$_{CTO}$ and PW$_{CCO}$ assigned to ID$_{CTO}$ and ID$_{CCO}$ as shown in Fig. 4.2. The GTRBAC policies of these domains are listed in Table 4.3 and the corresponding FSMs (F$_{CTO}$ and F$_{CCO}$) are shown in Fig. 4.4. For ID$_{CTO}$, we consider the state transition path $\pi_1 = s_3.e_3.s_4.e_4.s_5.e_5.s_6.e_6.s_7.e_7.s_6$ of F$_{CTO}$ that satisfies the composibility conditions WC1, WC2, and WC3 for PW$_{CTO}$. In the path π_1, all the states s_3, s_4, s_5, s_6, and s_7 support execution of *exemption processing* (EP) task that requires activation of the role TEP by an authorized user (in this case u_1). The task of *payment processing* (PP) can be performed in states s_6 or s_7 in which the role TPP is active for user u_2. Finally, in state s_7 the *refund adjustment* (RA) *task* can be processed by u_3 assuming the role TRP. For the projected workflow PW$_{CCO}$, the state path $\pi_2 = s_{21}.e_{26}.s_{22}.e_{27}.s_{23}.e_{28}.s_{21}$ of F$_{CCO}$ satisfies the composibility conditions WC1, WC2, and WC3. In the path π_2, the *initial assessment* (IA) task can be processed in all states

included in π_2. The tasks of preparing *final estimate* (FE) and *clearance processing* (CP) can only be performed in state s_{23}. The constraints on the initiation and completion times of the *component service*s imposed by state paths π_1 and π_2 are listed in Table 4.6.

To verify whether the state paths π_1 and π_2 satisfy all the cross-domain dependencies between ID_{CTO} and ID_{CCO}, the following system of constraints is generated and solved for a feasible solution.

(a1) $v\Delta_{CCO} + \theta_{IA}^{\pi 2} \le w\Delta_{CTO} + \phi_{EP}^{\pi 1}$; (a2) $w\Delta_{CTO} + \theta_{EP}^{\pi 1} \le v\Delta_{CCO} + \phi_{FE}^{\pi 2}$;
(a3) $v\Delta_{CCO} + \theta_{FE}^{\pi 2} \le w\Delta_{CTO} + \phi_{RA}^{\pi 1}$; (a4) $v\Delta_{CCO} + \theta_{PP}^{\pi 1} \le w\Delta_{CCO} + \phi_{CP}^{\pi 2}$;
(a5) $480 \le \phi_{IA}^{\pi 2} \le 585$; (a6) $505 \le \phi_{FE}^{\pi 2} \le 610$; (a7) $605 \le \phi_{CP}^{\pi 2} \le 710$;
(a8) $485 \le \theta_{IA}^{\pi 2} \le 590$; (a9) $565 \le \theta_{FE}^{\pi 2} \le 670$; (a10) $615 \le \theta_{CP}^{\pi 2} \le 720$;
(a11) $540 \le \phi_{EP}^{\pi 1} \le 740$; (a12) $620 \le \phi_{RA}^{\pi 1} \le 800$; (a13) $560 \le \theta_{EP}^{\pi 1} \le 760$; (a14) $680 \le \theta_{PP}^{\pi 1} \le 840$;
(a15) $20 \le \phi_{FE}^{\pi 2} - \theta_{IA}^{\pi 2} \le 55$; (a16) $40 \le \phi_{CP}^{\pi 2} - \theta_{FE}^{\pi 2} \le 170$; (a17) $60 \le \phi_{RA}^{\pi 1} - \theta_{EP}^{\pi 1} \le 235$;
(a18) $v \ge 0$ and integer; $v \ge 0$ and integer

One of the feasible solutions to the above system of inequalities have the following assignment: $v = w = 1$, $\phi_{IA}{}^{\pi 2} = 580$, $\theta_{IA}{}^{\pi 2} = 590$, $\phi_{EP}{}^{\pi 1} = 590$, $\theta_{EP}{}^{\pi 1} = 610$, $\phi_{FE}{}^{\pi 2} = 610$, $\theta_{FE}{}^{\pi 2} = 670$, $\phi_{RA}{}^{\pi 1} = 670$, $\theta_{PP}{}^{\pi 1} = 710$, $\phi_{CP}{}^{\pi 2} = 710$, $\theta_{CP}{}^{\pi 2} = 720$. With this assignment the initiation time of the *tax redemption workflow* of Fig. 4.2 is 580 minutes with respect to the calendar clock of ID_{CCO}, i.e., the *tax redemption workflow* can be initiated at 9:40 am. Note that the above system of inequalities is satisfied if we add the term *km*' to both sides of constraints a1 - a4, listed above. Where *m'* is the *lowest common multiple* of $v\Delta_{CTO}$ and $w\Delta_{CCO}$, and *k* is any non-negative integer. For the above assignment, *m'* equals 1440. With reference to the FSM of Fig. 4.4, *m'* corresponds to a periodic time instant at which ID_{CTO} returns to state s_3 (first state of π_1) and ID_{CCO} returns to state s_{21} (first state of π_2). This means that the *tax redemption workflow* can be repeatedly initiated after every 1440 minutes (24 hours) since its previous initiation. In other words, the workflow can be initiated every day at 9:40 am.

4.5. Composibility Verification Algorithm

In this section, we present two algorithms for verifying the composibility of a given distributed workflow. The first algorithm, presented in Section 4.5.1, verifies the correctness of a domain specific projected workflow by finding all valid state paths that satisfy the intra-domain workflow composibility conditions. We use the term valid state path to refer to a path that satisfies composibility conditions WC1, WC2, and WC3 for the given projected workflow. The second algorithm, presented in Section 4.5.2, analyzes all inter-domain path combinations for satisfaction of the inter-domain workflow composibility condition.

4.5.1. PW consistency verification

For discovering all valid state paths for a given PW, we use functions *smap* and *emap* iteratively. The function *smap* maps a given task of a PW to a set of GTRBAC states that have the appropriate user-role activation required for task execution. We refer to such states as the entry states of the task. Formally: $smap(\tau) = \{s \mid s \in S \textit{ and } \exists (u, r) \in DRM(\tau) \textit{ such that role } r \textit{ is active for } u \textit{ in state } s\}$. The function *emap* maps an edge (τ_i, τ_j), representing successive tasks in a PW graph, to a set of state paths Π_{ij} that satisfy the composibility conditions WC1, WC2, and WC3 for tasks τ_i and τ_j.

Fig. 4.9 shows the pseudo-code for the edge-mapping procedure. This procedure first discovers all state paths with traversal time less than a threshold value between all entry states of tasks τ_i and τ_j. The threshold value corresponds to the maximum time allowed for completion of tasks τ_i and τ_j. The discovered paths are then analyzed for satisfaction of composibility conditions WC1, WC2, and WC3 for tasks pair (τ_i, τ_j). We use the minimum residence time graph (MRTG) defined in Section 4.3.4 to discover these state paths. As mentioned in Section 4.3.4, the set all MRTG paths between state nodes s_1 and s_2 that are shorter than a given threshold value T includes all state paths from s_1 to s_2 with traversal time less than T.

```
WPS
INPUT: PW (project workflow graph of a domain)
  TG (GTRBAC timed graph of domain)
        MRTG (Minimum residence time graph of a domain)
OUTPUT: $\Pi_{1n}$ (set of all state paths that satisfy all the intra-domain
compositibility conditions for PW)
1    for $\tau_i \in$ PW
2      do color[$\tau_i$] $\leftarrow$ white
3        for each $\tau_j \in$ PW
4          do $\Pi_{ij} \leftarrow \phi$
5    color[$\tau_1$] $\leftarrow$ dark gray
6    $Q \leftarrow \tau_1$
7    while $Q \neq \phi$
8      do $\tau_i \leftarrow$ dequeue($Q$)
9        for each $\tau_j$ such that $(\tau_i, \tau_j) \in$ E[PW]
10       do if color[$\tau_j$] = white  or color[$\tau_j$] = light gray
11            then $\Pi_{ij} \leftarrow$ edge-mapping(TG, MRTG, $\tau_i$, $\tau_j$)
12               if $\tau_i \neq \tau_1$
13                 then if color[$\tau_j$] = light gray
14                      then $\Pi_{1j} \leftarrow$ path-extend($\Pi_{1i}$, $\Pi_{ij}$) $\cap \Pi_{1j}$
15                      else  $\Pi_{1j} \leftarrow$ path-extend($\Pi_{1i}$, $\Pi_{ij}$)
16               if there exists $\tau_k$ such that $(\tau_k, \tau_j) \in$ E[PW]
17                 and $\tau_k \neq \tau_i$ and color[$\tau_k$] $\neq$ black
18                 then color[$\tau_j$] $\leftarrow$ light gray
19                 else color[$\tau_j$] $\leftarrow$ gray
20                      $Q \leftarrow Q \cup \tau_j$
21       color[$\tau_i$] $\leftarrow$ black
```

Fig. 4.8 Algorithm for discovering valid state paths of a given PW.

```
edge-mapping(TG, MRTG, $\tau_i$, $\tau_j$)
1. for each $p \in smap(\tau_i)$ and $q \in smap(\tau_j)$
2.    do $\Pi \leftarrow$ find-all-paths(p, q, MRTG)
3.       for each $\pi' \in \Pi$
4.          do if $\pi'$ does not satisfy any of the composibility conditions
   WC1, WC2, and WC3 for $\tau_i$ and $\tau_j$
5.             then $\Pi \leftarrow \Pi - \pi'$
6. return $\Pi$

path-extend(PW, $\Pi 1i$, $\Pi ij$)
1. $\Pi'_{1j} \leftarrow \phi$
2. for each $\pi'$, $\pi$ such that $\pi' \in \Pi_{1i}$ and $\pi \in \Pi_{ij}$
3. do $\pi_i \leftarrow$ maximal sub-path of both $\pi'$ and $\pi$
4.       such that finish($\pi_i$, $\pi'$) and start($\pi_i$, $\pi$)
5.       if $\pi_i = \pi$ then $\pi_{1j} \leftarrow \pi'$
              else $\pi_{1j} \leftarrow concat(\pi'/\pi_i, \pi)$
6.       if $\pi_{1j}$ satisfies the conditions WC1, WC2, and WC3 for all task
   pairs $\tau_p$, $\tau_q$ such that $(\tau_p, \tau_q) \in E[PW]$ and color[$\tau_p$] $\neq$ white and ($\tau_q$
   $= \tau_j$ or color[$\tau_q$] $\neq$ white)
7. then $\Pi'_{1j} \leftarrow \Pi'_{1j} \cup \pi_{1j}$
8. return $\Pi_{1j}$
```

Fig. 4.9 Functions used by the WPS algorithm

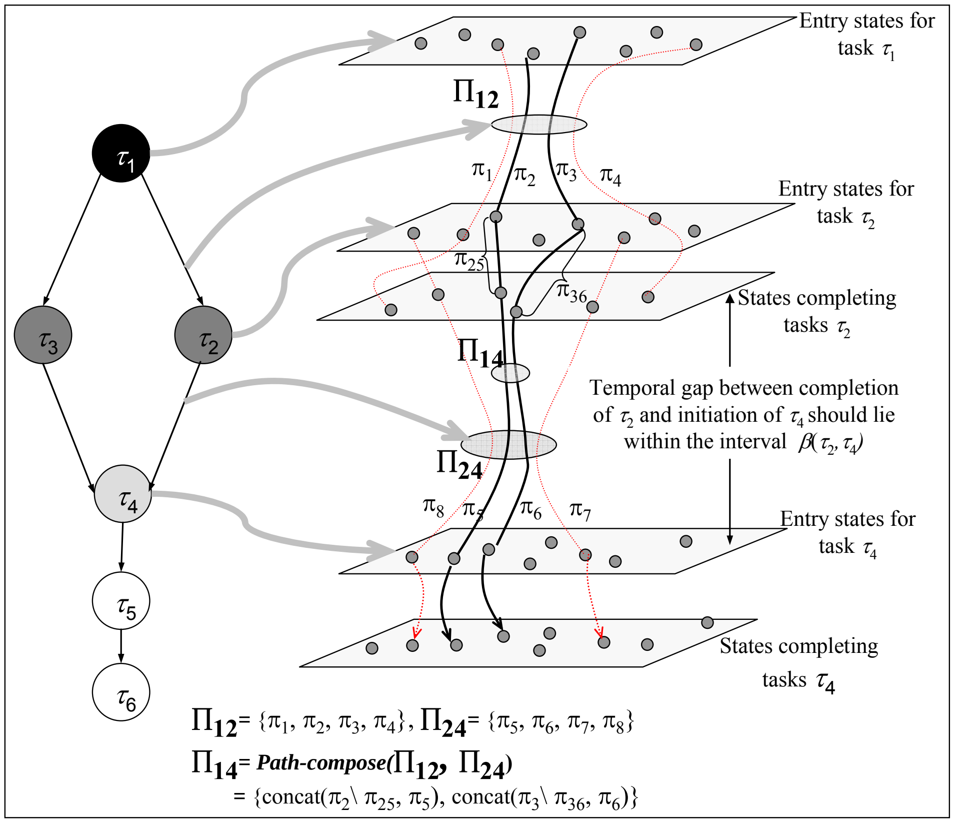
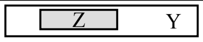
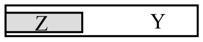
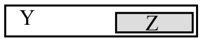
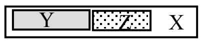

Entry states for task τ_1
Π_{12}
τ_1
π_1 π_2 π_3 π_4
Entry states for task τ_2
π_{25}
π_{36}
States completing tasks τ_2
τ_3
τ_2
Π_{14}
Temporal gap between completion of τ_2 and initiation of τ_4 should lie within the interval $\beta(\tau_2, \tau_4)$
Π_{24}
τ_4
π_8 π_5 π_6 π_7
Entry states for task τ_4
τ_5
States completing tasks τ_4
τ_6
Π_{12}= {π_1, π_2, π_3, π_4}, Π_{24}= {π_5, π_6, π_7, π_8}
Π_{14}= ***Path-compose***(Π_{12}, Π_{24})
= {concat(π_2\ π_{25}, π_5), concat(π_3\ π_{36}, π_6)}

Fig. 4.10 Mapping between PW graph and state paths of a domain's FSM

Table 4.7
Path relations

Let X = <x_0=(s_0,e_0), ….. x_{m-1}=(s_{m-1},e_{m-1})>, Y = <y_0=(s_0',e_0'),… .. y_{n-1}=(s_{n-1}',e_{n-1}') >, and Z = <z_0=(s_0'',e_0''),…., z_{k-11}=(s_{k-1}'',e_{k-1}'') > be state paths.		
Relation Predicate	**Definition**	**Illustration**
during(Z, Y)	There exists an index *i* of Y, such that for all *j* = 0, 1, ..., *k*-1, $z_j = y_{i+j}$.	
Start(Z, Y)	For all *j* = 0, 1, ..., *k*-1, $z_j = y_j$	
finish(Z, Y)	For all *j* = 0, 1, ..., *k*-1, $z_j = y_{(n-1)-(k-1)+j}$, where $y_{(n-1)}$ is the last element of the state transition path Y.	
Meet(Z, Y, X)	There exists an index *p* of path X, such that for all *i* = 0, 1, ..., *n*-1, and for all *j* = 0, 1, ..., *k*-1, $y_i = x_{p+i}$, $z_j = x_{p+n+j}$.	

The complete state path of a PW can be composed by incrementally extending the state paths corresponding to successive edge mappings as shown in Fig. 4.10. In this figure, $\pi_2 \in \Pi_{12}$ represents a state path from the initiation of task τ_1 to the completion of task τ_2, and $\pi_5 \in \Pi_{24}$ is a state path from the initiation of task τ_2 to the completion of task τ_4. These two state paths overlap for the execution duration of task τ_2 and therefore can be combined to compose a state path from τ_1 to τ_4. Let π_{25} be the overlapping sub-path of π_2 and π_5 such that the following path relations, defined in Table 4.6, hold: *finish*(π_{25}, π_2) and *start*(π_{25}, π_5). Moreover, for all states *s* in π_{25}, $s \in$ *smap*(τ_2). The state path π_2 can be written in a concatenated form as $\pi_2 = \pi'.\pi_{25}$, where π' is a sub-path of π_2 such that *meet*(π', π_{25}, π_2) is true. A state path π_{14} can be composed by concatenating π_5 to the end of π'. The path π_{14} need to be checked for satisfaction of composibility conditions WC1, WC2, and WC3 for both task pairs (τ_1 ,τ_2) and (τ_2 ,τ_4).

Fig. 4.8 shows an algorithm *workflow path search* (WPS) for discovering all valid state paths for a given PW. The algorithm takes input the task graph of a PW, the FSM of a domain's GTRBAC policy represented as timed graph TG, and the minimum residence time graph of TG. During the path search, the PW graph is traversed in a breadth first manner to explore all the state paths from the source node τ_1 to all other nodes of the PW that satisfy the composibility conditions WC1, WC2, and WC3 for all successive pairs of tasks in the PW. The path set $\prod$ is indexed by the indices of source and destination nodes of the PW graph. For a given source $\tau_1 \in$ PW and destination $\tau_i \in$ PW, $\prod_{1i}$ denotes the set of all valid state paths that satisfy all the temporal ordering and duration constraints for successful completion of a PW with τ_1 as a source node and τ_i as a terminal node in the PW graph.

To keep track of the tasks whose state paths from the source node have been discovered, the algorithm *WPS* colors each vertex in the PW graph as white, dark gray, light gray, or black. All task vertices in the PW graph start out as white. A task vertex τ_i becomes dark gray after the discovery of all the valid state paths from the source vertex τ_1 to τ_i. Incase a task has multiple adjacent vertices that precede τ_i in the execution order, τ_i becomes light gray after its first discovery and remains light gray until the consistency of the path set $\prod_{1i}$ has been verified for all the adjacent vertices preceding τ_i in the workflow execution order. After this consistency verification, vertex τ_i is colored from light gray to dark gray. A dark gray vertex τ_i becomes black after all the valid state paths from τ_1 to all the adjacent vertices of τ_i have been discovered. The algorithm terminates when all the vertices of the PW have been colored black. At this point all the valid state paths from τ_1 to the final task have been discovered.

Correctness of the algorithm: To verify that a PW conforms to the GTRBAC policy of the designated domain, we need to find at least one state path that satisfies all the intra-domain workflow composibility conditions WC1, WC2, and WC3 described in Section 4.4.1. The set of all state paths returned by the *WPS* procedure meet this requirement for PW verification. It can also be noted that the set of state paths returned by *WPS* are exhaustive, i.e., if any path satisfies the intra-domain composibility conditions for a given PW, then it is included in the set of paths discovered by *WPS*.

Theorem 4.1: Let G_X be a graph representing the specification of a PW assigned to domain ID_X. Suppose τ_1 is a distinguished source vertex of G_X that initiates the PW. Let τ_j ($\neq \tau_1$) be any task vertex in G_X. In the *WPS* procedure, after the task vertex τ_j is colored dark gray, following properties hold for each state transition path π in the path set $\prod_{1j}$.

- Compozibility condition WC1 is satisfied for task τ_j and all tasks τ_i preceding τ_j in task execution order.
- Composibility conditions WC2 and WC3 are satisfied for all task pairs (τ_i, τ_j) such that (τ_i, τ_j) $\in E[G_X]$. In addition, WC2 and WC3 are also satisfied for all task pairs (τ_p, τ_q) such that (τ_p, τ_q) $\in E[G_X]$ and τ_p, τ_q precedes τ_j in the task execution order.

Proof given in Appendix B.

4.5.2. Cross-domain dependency verification algorithm and complexity

In this section, we present a simple algorithm, shown in Fig. 4.11, for verification of distributed workflow with respect to cross-domain dependencies. The symbols and notations used in this procedure are described in Table 4.5. The cross-domain dependency verification is performed by a central site. Given $\Pi^{(i)}$, $CS^{(i)}_{init}$, and $CS^{(i)}_{end}$ ($1 \leq i \leq n$) for all collaborating domains, and the set CS_{dep}, the algorithm analyzes all cross-domain path combinations for satisfaction of the precedence relationship between *component service*s specified in CS_{dep}. In this analysis, a system of inequalities defining precedence relationship among the *component service*s is generated and solved for each *n*-ary tuple $y = (\pi_1, \pi_2, \ldots, \pi_n)$. A feasible solution to this system of inequalities implies the following:

- The state path combination $(\pi_1, \pi_2, \ldots, \pi_n)$ corresponding to the tuple y, satisfies all the cross-domain dependency relationships specified in the distributed workflow specification.
- For the above path combination, the projected workflow in ID_i can be supported at any time included in the solution space of the system of inequalities generated for y.

If no feasible solution exists for any tuple $y \in (\Pi^{(1)} \times \Pi^{(2)} \times \ldots \times \Pi^{(n)})$, then the verification procedure returns **No**. In this case, the given distributed workflow cannot be supported. This is stated in the following theorem.

Theorem 4.2: Given $\Pi^{(i)}$, $CS^{(i)}_{init}$, and $CS^{(i)}_{end}$ for each domain ID_i ($1 \leq i \leq n$) and the set CS_{dep}, if the cross domain verification procedure fails, then the cross domain dependencies in the given set CS_{dep} cannot be satisfied, Accordingly, the corresponding distributed workflow cannot be supported.

Proof of this theorem is given in Appendix B.

The proposed workflow composibility verification approach has a high computational complexity as observed in many other similar problems [91, 127, 32, 31, 30]. This high complexity is mainly due to the exhaustive state path searches performed in the projected workflow verification step. In this step, all state paths of length less than a given threshold value are discovered between the entry states of successive tasks of the projected workflow. The problem of finding all length limited paths between any two nodes in a graph is at least as difficult as solving the S-T PATH problem, which is defined as finding all simple paths from a node *s* to another node *t* in a graph. The S-T PATH problem is proved to be #P-Complete [125].

The complexity of the composibility verification problem discussed in this chapter can be significantly reduced, if instead of discovering all valid state paths, only m-shortest paths are discovered, where $m > 1$. However, this heuristic will consider a sub-set of all valid state paths for workflow verification and in the worst case may declare a correct workflow specification as incomposable.

```
cross-domain-dependency-verification
INPUT: $\Pi^{(i)}$ for all IDs ($1 \leq i \leq n$)
       $CS^{(i)}_{init}$ for all IDs ($1 \leq i \leq n$)
          $CS^{(i)}_{end}$ for all IDs ($1 \leq i \leq n$)
          $CS_{dep}$
OUTPUT: {Yes, No}
Y ← $\Pi^{(1)} \times \Pi^{(2)} \times \ldots \times \Pi^{(n)}$
  1. for each $y \in Y$
  2.    do for each (($c_q$, $c_r$) $\in CS_{dep}$ define cross-domain dependency constraints
        between $c_q$ and $c_r$ end for /*end inner for loop of line 3 */
  3.       if the solution space to the system of inequalities generated for $y \in Y$ is
           non empty then return Yes
  4. end for /*end outer for loop of line 2 */
  5. return No
```

Fig. 4.11 Algorithm for verifying distributed workflow with respect to cross-domain dependencies among component services.

4.6. Conclusions

In this chapter, we have proposed an approach for verifying the secure composibility of distributed workflows in a collaborative environment comprising autonomous domains. The objective of workflow composibility verification is to ensure that all the users or processes executing the designated workflow tasks have proper authorization and their activities within the context of workflow specification cannot cause security breaches in any domain. The proposed approach achieves this objective by verifying the distributed workflow specifications against the access control policies of all domains collaborating for workflow execution. A key challenge in this verification process is posed by the time-dependent access control policies of collaborating domains which are specified using GTRBAC model. The GTRBAC policy of a domain contributes to its non-reentrant behavior which is modeled as a time augmented FSM. The proposed approach verifies workflow composibility by exploring the FSM of each domain to find state paths that satisfy the given workflow specifications. This workflow composibility verification is performed without creating a unified global FSM which is required for model checking-based approaches for composibility verification. Therefore, the proposed approach is unique and does not compromise the autonomy and privacy of collaborating domains.

5. A FRAMEWORK FOR COMPARISON OF POLICY-BASED DISTRIBUTED SYSTEMS

This chapter provides a comparative analysis of the policy composition approach, described in Chapter 3, and the workflow composibility verification approach described in Chapter 4. The trade-off between these approaches is analyzed with respect to the four metrics, including, degree of interoperation (DOI), degree of autonomy (DOA), degree of privacy (DOP), and verification complexity. These metrics are defined in Section 5.1.1 of this chapter. In addition, this chapter presents a comparative analysis between the policy composition and verification approaches proposed in this dissertation and the existing approaches for verification of distributed systems and services.

5.1. Comparative Analysis of the Proposed Approaches for Secure Composition of Collaborative Applications

In this section, we discuss the trade-off between the global meta-policy based approach and the distributed multi-policy based approach for composibility verification of time-dependent collaborative applications. We refer to verification approach described in Chapter 4 as multi-policy based approach as it performs verification of the distributed workflow applications with respect to the policies of multiple collaborating domains and does not require a global meta-policy that mediates cross-domain accesses. The comparison between the two approaches is performed with respect to various aspects of the collaborative environment and distributed applications. These include, degree of coupling among domains, level of information and resource sharing, privacy preferences of domains for disclosure of their policies, overhead related to policy composition, and complexity associated with verification of distributed collaborative applications. We use a set of metrics to formally analyze the trade-off between the two approaches based on the above-mentioned criteria. In the following, we first describe these metrics and then discuss the trade-off between the two verification approaches.

5.1.1. Metrics

For measuring the effectiveness of policies and mechanisms employed in multi-domain system for development of secure and consistent information sharing and collaborative applications, we consider the following set of metrics: i) degree of interoperation (DOI), ii) autonomy loss (AL), iii) degree of privacy (DOP), and iv) verification complexity (VC). The first two metrics (DOI and AL) are evaluated based on the number of cross-domain accesses and local accesses. As discussed earlier, a cross-domain access corresponds to acquisition of privileges on the local objects of a domain by an agent or process running on behalf of a remote user, i.e., the user is not affiliated with the domain owning the object being accessed. A local access corresponds to the acquisition of privileges over a domain's local objects by a local user or process running on behalf of local user. The objects of a domain can be characterized as data objects (database relations, tuples, views, documents, files etc.), compute and storage resources CPU cycles, disk space, printers), and services (credit checking service, tax filing service). We assume that each object is a separate entity and do not consider any hierarchical or object-oriented model for defining relationship between different objects. For instance, a database table, say T, and a view, say V, defined on some columns of T are considered as two separate objects, and accessing the table T is considered as a single access even though the view V is derived from T. Similarly, we do not make a distinction between materialized and non-materialized (e.g., views) objects.

5.1.1.1. Degree of Interoperation

The *degree of interoperation* (DOI) indicates the level of information sharing a domain allows in a multi-domain environment. This information sharing can be determined in terms of the cross-domain accesses provided by the domain to its local objects. Let O_i^S denote the set of the shareable objects of domain D_i that can be accessed remotely by cross-domain users, and O_i^L denote the set of objects that can be accessed locally within domain D_i. The DOI provided by domain D_i can be quantified as:

$$DOI(D_i) = \frac{|O_i^S|}{|O_i^S \cup O_i^L|},$$

Where, |O| denote the cardinality of the set O. The $DOI(D_i)$ assumes a value of one, if all the object owned by domain D_i are shareable and can be accessed remotely, i.e., $O_i^L \subseteq O_i^S$. On the other hand, if none of the objects owned by D_i can be accessed then the DOI offered by D_i is zero.

The overall DOI for the multi-domain environment comprising n domains can be evaluated by taking the average value of DOI of individual collaborating domains.

$$DOI\left(\bigcup_{i=1}^{n} D_i\right) = \frac{1}{n}\sum_{i=1}^{n} DOI(D_i)$$

Note that the degree of interoperation is not a fixed value and may change with time as the information sharing policies of domains may get evolved or domains may disassociate themselves from the collaboration. Generally, the degree of interoperation remains stable in tightly-coupled or federated systems in which collaboration among domains is on a long term basis and the domains have a high degree of trust towards each other.

5.1.1.2. Autonomy

Autonomy refers to the ability of a domain to carryout its local operations and activities without any interference from cross-domain accesses or services provided to remote users. In a multi-domain collaborative environment, autonomy of a domain is measured in terms of its users' authorizations over the domain's local objects or resources [61]. A collaborative environment is considered to be autonomy preserving if all the local authorizations of domains remain unaffected by the cross-domain accesses. However, there is a trade-off between seeking interoperability and preserving autonomy. In order to provide cross-domain information accessibility, a domain may be forced to restrict its local accesses. This is considered as autonomy loss (AL) which can be quantified as:

$$AL(D_i) = \frac{\begin{pmatrix}\text{Total number of local accesses} \\ \text{of } D_i \text{ prior to interoperation}\end{pmatrix} - \begin{pmatrix}\text{Total number of local accesses of } D_i \\ \text{after establishing interoperation}\end{pmatrix}}{\begin{pmatrix}\text{Total number of local accesses} \\ \text{of } D_i \text{ prior to interoperation}\end{pmatrix}}$$

5.1.1.3. Degree of Privacy

The *degree of privacy* (DOP) specifies how much information a domain is willing to disclose about its internal policies, local constraint, and non-shareable objects and metadata. Generally, the access control policy of any domain is considered as a protective object as it contains information about the domain's organizational structure, business strategies, security mechanisms, and other protective resources [133, 134, 128, 129, 89, 116]. Therefore, disclosing the contents of domain's access control policy may leak sensitive information which can be misused by adversaries.

The privacy metric has been quantitatively defined in the data-mining literature as a measure of how closely the original value of a modified, obfuscated, or distorted attribute can be estimated. From the policy disclosure perspective, we consider the privacy as a measure of how much information about a domain's non-disclosed policy attributes, non-shareable objects or meta-data can be inferred by untrusted domains. Since this information is inferred from the policy attributes that are voluntarily disclosed by the domain for establishing interoperation, therefore, we measure privacy in terms of how much information a domain provides about its policy. Let PA denote the set of all policy attributes of domain D_i and PA^{S} ($\subseteq$ PA) denote the set of all policy attributes that are disclosed to the collaborating domains for facilitating secure information and resource sharing. We quantify the DOP of a domain D_i as:

$$DOP(D_i) = \frac{\sum_{pa_j \in PA^S} w_j \mid pa_j \mid}{\sum_{pa_k \in PA} w_k \mid pa_k \mid}$$

Where, pa_j denote the set of related attributes in the domain's policy, $\mid pa_j \mid$ denote the cardinality of the set pa_j, and w_j specifies the relative importance or weight of the attributes in the set pa_j with respect to other attribute types. Disclosure of different policy attributes will have a different impact on the privacy of the domain. For example disclosing the assignment of non-shareable objects to roles may result in a lower degree of privacy as opposed to the disclosure of assignment of shareable objects. This impact of attribute disclosure on the degree of privacy is characterized by the weight of the corresponding attribute. In the context of RBAC, the policy attributes may include the set of roles, the role hierarchy relation, user-role assignment relation, role-permission assignment relation, SoD constraints, and role cardinality constraints.

5.1.1.4. Verification Complexity

Verification complexity (VC) characterizes the overhead associated with verifying the correctness of distributed applications. This overhead can be evaluated in terms of the algorithmic complexity of the verification approach. In addition, we also consider the overhead associated with structuring, organization, and management of data and policies for computing VC. For instance, verifying the specification of a distributed application for conformance against a single meta-policy is much easier than verifying the same specifications against the multiple-policies. However, generation of a global meta-policy may have a significantly high complexity. Moreover, the meta-policy may get evolved due to changes in the policies of domains, joining of new domains, or disassociation of a domain from the collaborative environment. Any change in the meta-policy may warrant re-verification of distributed applications verified with respect to the previous meta-policy. Therefore, the overhead associated with the composition and management of meta-policy and re-verification of previously verified applications need to be incorporated in the complexity metric.

5.1.2. Global meta-policy vs. distributed multi-policy

In this section, we compare the global meta-policy and distributed multi-policy based approaches for secure composition of distributed workflow based collaborative applications. Our comparison is based on the metrics described in the above section. For uniformity in comparison, we assume that the policies of all collaborating domains have the same level of expressiveness and can support specification of temporal constraints as discussed in Chapter 4. Moreover, we assume that the global meta-policy composed from the local policies of collaborating domains preserve all the temporal dependencies specified in local policies and also allows specification of temporal constraints on cross-domain accesses. The meta-policy composition approach, described in Chapter 3, is designed for composition of RBAC policies which do not include temporal constraints. However, the policy composition approach can be easily extended to incorporate temporal constraints and some preliminary work in this direction has been described in [24]. We also assume that all the time dependent policies including the global meta-policy and the local policies of domains can be represented using finite state models (FSMs). In Chapter 4, we have described the procedure for transforming a GTRBAC policy specification into an FSM-based representation.

Table 5.1 shows the comparison between the global meta-policy and distributed multi-policy based approaches for secure composition of collaborative environments using the aforementioned metrics. In the following, we elaborate on this comparison for each individual metric.

Degree of coupling. As discussed in Chapter 3, a global meta-policy is composed to support collaboration in a tightly-coupled or collaborative environment in which the domains have a high degree of mutual trust and establish a long-term relationship. Therefore, the meta-policy based approach is suitable for designing tightly integrated business-to-business (B2B) processes that require close interactions among a pre-selected set of collaborating domains on a long-term basis. The main advantage of this approach is that it can guide the development of secure and consistent collaborative applications based on a single policy without requiring excessive mediations among collaborating domains. In other words, ensuring that a collaborative application satisfies all the constraints and requirements of a global meta-policy is sufficient to verify its consistency with respect to the local policies of all collaborating domains. A distributed meta-policy based approach can also support collaborative applications that require long-term interactions among domains in both federated and loosely-coupled environment. A major drawback of this approach is that whenever a new collaborative application is composed or an existing application is modified, it needs to be verified against the policies of all collaborating domains and may require extensive mediation among these domains.

Degree of interoperation. Generally collaborative applications in a federated environment require a high degree of interoperation. As discussed in Chapter 3, a high level of information and resource sharing among domains may introduce conflicts due to the interplay of various constraints in the policies of collaborating domains. These conflicts if remain undetected and unresolved expose the domains to numerous security vulnerabilities and may cause serious security breaches such as unauthorized accesses, privilege escalation, and SoD violation. Detection and resolution of these conflicts involves extensive mediation among collaborating domains and requires a global view of the authorizations of all local and cross-domain accesses. This global view is provided by a global meta-policy, which can also facilitate in establishing an optimal level of information and resource sharing without violating the security constraints of collaborating domains. In this regard, a meta-policy based approach can be categorized as a preventive approach that prevents security breaches due to diverse security policies of collaborating domains. A multi-policy based approach can also detect potential accesses that may violate the security constraints of individual domains, however, such an approach may not resolve the policy conflicts in a consistent, deterministic, and optimal manner.

Autonomy. In a federated environment operated under a global meta-policy, all the cross-domain accesses in a collaborative application that conforms to the global meta-policy must be supported by each collaborating domain even though such accesses may affect the local operations of the domain. Therefore, a meta-policy based approach provides little or no autonomy to collaborating domains to reject a cross-domain resource access request whose authorization is established by the meta-policy. Generally domains exercise their autonomy during the mediation phase for generation of the meta-policy as discussed in Chapter 3. However, once a meta-policy is generated it guides the development and composition of collaborative applications without considering the local policies of domains.

As discussed in Chapter 4, a multi-policy based approach can also be used for developing collaborative applications that require recurrent interactions among a pre-selected set of domains on a long-term basis. This approach provides a high degree of autonomy during the design of such collaborative applications as domains may choose not to participate in any collaborative activity that may cause violation of their local policy constraints or restrict the accessibility of their local users. A domain may also specify its terms and conditions under which it can provide access to it resource/services to support a collaborative application. However, once a domain agrees to support a collaborative application, it cannot reject any cross-domain access that satisfies all its terms and conditions.

Degree of privacy. The degree of privacy is the most important metric that distinguishes the two approaches. In order to compose a secure and consistent global meta-policy, a full disclosure of domains access control policies is needed. However in a loosely-coupled multi-domain environment, a domain may only provide information specific to its shareable resources and may not disclose its policy completely due to security or privacy reasons. Therefore, a global meta-policy based approach cannot be applied in a loosely-coupled collaborative environment.

Verification complexity. In a federated environment operating under a global meta-policy, a model checking based approach can be used to verify the composiblity and correctness of distributed collaborative applications. As discussed above, the meta-policy is modeled using a time-augmented finite state machine (FSM) to capture the temporal constraints and time-dependent authorizations.

The meta-policy is generated from the access control policies of all collaborating domains. In Chapter 3, we have discussed how a global meta-policy can be composed from the RBAC policies of domains. The main complexity of meta-policy composition lies in the resolution of policy conflicts that may occur due to the interplay of various policy constraints in the multi-domain environment. In Chapter 3, we have presented an integer programming (IP)-based approach that resolves such conflicts and generates a meta-policy that allows maximum interoperation. The problem of generating a secure and optimal meta-policy is proved to be NP hard [61] and the proposed IP-based technique follows the same complexity result.

The proposed IP-based technique can be extended to compose a global meta-policy from the time-dependent policies of all collaborating domains. These time dependent policies are specified using the GTRBAC model (restricted version) as discussed in Chapter 4. In order to generate a secure and optimal meta-policy from the GTRBAC policies of collaborating domains, the IP-based conflict resolution procedure needs to be invoked every time a periodic temporal constraint becomes active. In the GTRBAC model discussed in Chapter 4, we assume that periodicity constraint can be defined for role enabling events only. With this assumption, the IP-based conflict resolution procedure needs to be called at most |R| times, where |R| denotes the total number of roles in the multi-domain environment. Therefore the time complexity of composing a global meta-policy from the GTRBAC policies of collaborating domains is **O(|R|2^{m})**, where *m* denotes the number of role mapping links. Another important parameter which is used to compute the verification complexity is the size of the state-space of the global meta-policy. The total number of states in the FSM of the global meta-policy is bounded by $\prod_{i=1}^{n} |S_i|$, where $|S_i|$ denote the number of states in the FSM of the GTRBAC policy of domain *i* and *n* denotes the total number of domains in the multi-domain environment. Similarly, the number of clocks in the FSM of the global meta-policy is bounded by $\sum_{i=1}^{n} |C_i|$, where $|C_i|$ denote the number of clocks in the FSM of the GTRBAC policy of domain *i*

For verifying the correctness of the distributed applications with respect to the global meta-policy, we can use a scenario matching approach proposed by Braberman *et. al.* in [30]. This approach uses a Time Computation Tree logic (TCTL)-based model checking technique to verify whether a given scenario can be supported by a timed automaton. The time complexity of TCTL-based model checking is linear in the number of states in the timed automaton modeling the system behavior and exponential in the number of clocks modeling the temporal constraints [4, 5]. With reference to the composibility verification of distributed applications, the application specification corresponds to the scenario that needs to be matched with the timed automaton corresponding to the global meta-policy. Therefore, given a global meta-policy in a timed-automaton representation and scenario corresponding to the application specification, the complexity of verifying whether the application can be supported by the meta-policy is given by $O\left(\prod_{i=1}^{n} |S_i|.S_A + 2^{\sum_{j=1}^{n}|C_j|}\right)$, where $|S_i|$ and $|C_j|$ are defined above and S_A denotes the number of states in the automaton generated from the scenario corresponding to the application specifications.

For the multi-policy based approach, the problem of verifying the specifications of a given distributed application is #P-complete as discussed in Chapter 4.

Table 5.1
Global meta-policy vs. distributed multi-policy

Metrics	Global Meta-Policy	Distributed Multi-policy
Degree of Coupling	Tightly-coupled federated multi-domain environment	Both federated and loosely-coupled multi-domain systems
Degree of Interoperation	High	Low - medium
Autonomy	Low	Allow high degree of autonomy in loosely-coupled multi-domain environment
Degree of Privacy	Low	High in loosely-coupled collaborative environment.
Verification Complexity	Linear in the number of states and exponential in the number of temporal constraints in the global meta-policy	#P-Complete

5.2. Policy-based Composition and Verification of Distributed Collaborative Applications

In this section, we present a comparison between the proposed policy composition and verification framework with the existing approaches for verification of distributed systems and services. For comparing these approaches, we will use the term component to refer to an individual system, domain, or a peer interacting with other components for supporting the collaborative applications. This comparison is based on the following factors: i) model used for specifying the policy-driven behavior of component systems; ii) model used for specifying interactions among different components; iii) behavioral properties of components being modeled such as real-time or bounded response time response or non-reentrant characteristics; iv) disclosure of components' policies driving their behavior, generation of a global meta-policy; v) verification criteria; and vi) verification complexity. All the approaches considered for the comparative analysis use formal models (such as state machines, timed automata, and Petri nets) to characterize the policy driven behavior of component systems.

A key factor that distinguishes the different approaches is whether they are able to verify the conformance of the interactions or interoperation requirements of distributed applications with the time-dependent non-reentrant behavior of the component systems. As discussed in Chapter 4, a component is characterized as reentrant or non-reentrant based on the software system implementing the component's functionality. A reentrant component can be invoked multiple times and all the invocations of such component are considered independent of each other. On the other hand a non-reentrant component does not allow its multiple simultaneous, interleaved, or nested invocations. At any time a non-reentrant component is managed by only one policy instance which determine its response to various events based on its current state. As a result, a non-reentrant component may interact differently with same peer components at different times. Consequently, a distributed application requiring interaction among different time-dependent and non-reentrant components may not be supported at any arbitrary time, even though the interactions specified in the application design are complete, consistent, unambiguous, and conform to the interface specifications of each individual component. A key issue related to verification of distributed applications requiring interoperation among component systems is to ensure that such applications can eventually be executed despite the components' time dependent non-reentrant behavior. This requires identifying all possible time instants during which the components can support the required interoperations and verifying whether such schedule satisfy the application requirements.

The verification approaches, discussed in this section, can also be differentiated based on how much information about component local policies or behavioral specifications need to be disclosed to other components. As discussed above, disclosure of local policies may help in generation of a global meta-policy, which may significantly reduce the verification complexity for some collaborative applications. Some distributed protocol verification approaches that do not rely on meta-policy generation, require that the interacting components should be able to access or inquire about each others state information [31]. However, disclosure of domain's policies in distributed enterprise environment or business-to-business based interactions may not be allowed due to security and privacy concerns of domains.

The verification criteria and verification complexity are the two most important metrics for comparing the different approaches. The approaches discussed in this section, differ based on the properties of the distributed applications they verify. The selection of these approaches based on the difference in their verification criteria is intentional to cover a broad range of representative work in the areas of distributed systems, protocol verification, and service composition. In Table 5.2, Table 5.3, and Table 5.4, we provide a comparison of the different verification approaches using the features discussed above. In the following, we briefly discuss these approaches in the context of their application category.

5.2.1. Web-service composition and verification

Berardi *et. al.* in [17] have used a model checking based approach for automatic composition of e-services. They have proposed an algorithm that takes the target service specifications and a set of available component services as input and synthesize a composite service that uses only the available component services and fully captures the target service. In this approach, the behavior of a target service is specified as an execution tree that captures all possible execution of the target service. The nodes in the execution tree represent the service state and the edges between the nodes denote the operations or actions. The number of states in the execution tree is assumed to be finite and a deterministic finite sate machine (FSM) is used to model the execution tree. The behavior of each component service is also modeled using a separate FSM. The authors have shown that the problem of checking whether the FSMs of the component services can synthesize the execution tree of the target service is equivalent to determining the satisfiability of a deterministic propositional dynamic logic (DPDL) formula. This formula is generated by a central entity, called composer, from the FSM of the target service and the FSMs of all component services. The composer is assumed to have full knowledge about the policies and the FSM of each component service. The model of the satisfiable DPDL formula corresponds to the execution tree of the target service. The complexity of determining the satisfiability of the DPDL formula corresponding to the target service and the component service specifications is exponential in the size of the FSM of all component services and the target service. This approach provides a formal framework for verifying the synthesis of a target service without considering any temporal dependencies and real-time constraints. Moreover, the component services are assumed to be reentrant and can be invoked/accessed at any time.

Betin-Can *et. al.* [32] have proposed a design methodology for reliable composition of Web services. In this methodology, the reliability of a composite Web service is verified by analyzing the interactions among the component Web-services, called peers, for satisfaction of safety and liveness properties. Interaction among the peers is established via asynchronous messages. Each peer is assumed to have a fist in first out (FIFO) queue which stores all the incoming messages to the corresponding peer. The peer processes all messages in the order they are received. Each peer advertises its interface specifications which describe the messages the peer can receive (input messages), the messages the peer can send (output messages), and the response of the peer with respect to different input messages. This response is specified in terms of the output messages the peer generates when it processes a given input message, or the messages the peer expects after sending an output message. The interface specification of each peer is described modeled as an FSM. The composite Web service is also modeled as a state transition system, which specifies the desired interactions among the peers in form of message exchanges. Each state in the state transition system corresponding to the composite Web service, provides information about the local state of all peers and the configuration of their message queues. Bertin-can *et. al.* use a modular approach for verification of composite Web services. The verification modules include interface verification and behavior verification. Interface verification involves checking whether the internal policy and/or the implementation of a peer conform to its interface specifications. The interface verification is performed by analyzing all possible execution paths of the peer's internal policy for violation of its interface specifications. The complexity of interface verification depends on the number of paths that can be generated from a given policy. Behavior verification involves analyzing the safety and liveness properties of the composite Web service. For behavior verification, the peers are assumed to conform to their interface specifications. Based on this assumption, the safety and liveness properties of a composite Web service can be verified by using only the peer interfaces to characterize their behavior without considering their internal policies. The safety and liveness properties can be represented as linear temporal logic (LTL) formulae which can be verified against the state transition system corresponding to the composite service specification. The complexity of verifying an LTL formula is linear in the size of the number of the states of the corresponding state transition system and exponential in the size of the LTL formula [32]. The verification methodology described in [32], is similar to the composibility verification approach discussed in Chapter 4 in the following ways: i) both approaches do not allow disclosure of the internal policies of the components, and ii) both approaches require verification of the internal policies of the components for conformance with their interfaces. However, [32] does not consider any real-time constraints or non-reentrant behavior of components during interface or behavior verification.

Chun *et. al.* in [36] have addressed the issue of policy based composition of web services in an open web-based collaborative environment. They consider different policies for service composition. These policies include service provisioning policies, service flow policies, and user-specific policies. Service provisioning policies are defined by the domains offering their services. These policies specify the terms and conditions that need to be met before the corresponding component service can be accessed or invoked. The service flow policies specify various constraints related to ordering of component services, component service selection criteria, and the semantic description of the desired/requested component services. The user policies specify the constraints and preferences of the end user in selecting the particular services. The service flow policy combined with the user policy defines the overall service composition requirements. In the service composition process, first all the component services that satisfy the semantic properties of the composite service are discovered. In this process similar component services from different domains may be selected. Next an instance of a composite service is generated by selecting one component service from each pool of similar/related services. This selection is based on the syntactic and semantic compatibility of the component services. The composite service instance is then evaluated for conformance of the overall service composition requirements with the service provisioning policies of the domains providing the selected component services. If the composite service instance satisfies all the composition requirements then it is considered as a valid service composition and the verification process stops. Otherwise, a new composite service instance is composed from the candidate pool of component services and is evaluated for satisfaction of the service flow and user-specific policies. No formal model for service specification or technique for verifying service composibility is provided in [36].

5.2.2. Distributed protocol verification

One of the earliest works on verification of communication protocols in distributed computing environment is by Brand and Zafiropulo [31]. They consider each protocol as a communicating process, which is modeled as a finite state machine (FSM). Each pair of communicating processes is assumed to be connected by a full-duplex, error free, FIFO channel via which the processes exchange messages. Brand and Zafiropulo do not consider any specific interactions among the communicating processes against which the function and behavior of each protocol needs to be verified. Rather, they consider certain properties of interest to all protocols independent of their intended functions. These properties include *executable reception* and *stable N-tuple*. *Executable reception* implies that a protocol must be able to process any received message from its current state. In other words, a protocol must never reach a state in which it is unable to send a message to other protocol processes or retrieves a received message that lies at the head of its FIFO channel. *Stable N-tuple* property implies that the communication among the protocols can lead to a global reachable state with all channels empty. A *stable N-tuple* may correspond to a deadlock situation. Brand and Zafiropulo have proved that given a set of communicating protocols, the problem of verifying *executable reception* and *stable N-tuple* reachability is undecidable in the general case. They have also proved that the problem is decidable if the following two conditions hold:

1. The channel size of each protocol is bounded.
2. Each communicating protocol can be transformed into a tree protocol.

With the above assumptions, they have provided an approach that verifies the *executable reception* and *stable N-tuples* properties for a given set of protocols. The approach verifies these properties separately for each protocol and takes exponential time in the size of protocol specifications. However, the approach is limited to verification of communicating processes and protocols that do not have any real-time constraints.

Fu *et. al.* in [51] have studied the problem of realizing a given conversation protocol. The conversation protocol describes all possible interactions or message exchanges that can occur among collaborating component peers. The conversation protocol in [51] is represented using a non-deterministic Buchi automaton. Fu *et. al.* have proved that given a set of peer components with their behavior specified using non-deterministic Buchi automata, the problem of verifying whether the given component peers conform to the conversation protocol is undecidable. Therefore, they have considered a top-down approach for determining the realizability of conversation protocols. They define the realizability problem as, given a Buchi conversation protocol, is it possible to obtain a composition of components which produces exactly the same set of conversations as specified by the global protocol. In this top-down approach, the implementation or behavior of each component is synthesized from the given conversation protocol via projection. For ensuring the realizability of a given conversation protocol, they have defined three necessary and sufficient conditions which the protocols must satisfy. These include lossless join condition, synchronous compatible condition, and autonomous condition. Verification of given conversation protocol, represented as a non-deterministic Buchi automaton, with respect to the three realizability conditions can be performed in EXPTIME in the size of the automaton.

5.2.3. Distributed real-time systems verification

Braberman *et. al.* in [29, 30] have addressed the issue of verifying the properties of distributed real-time systems using a scenario matching approach. In this approach, the behavior of each component system is described using a separate timed-automaton which is similar to the automaton we have considered for modeling GTRBAC policy. The properties that need to be verified are specified using visual timed event scenarios (VTS) that can specify both real-time and event-based constraints. For verifying whether the component systems satisfy a given scenario, a global automaton is composed from the timed automata of all component systems. The number of states in the global automaton can be of the order of $\prod_{i=1}^{n} |S_i|$, where $|S_i|$ denotes the number of states in the automaton of the ith component, and n denotes the number of the interacting components. Similarly, the number of clocks modeling real-time constraints in the global automaton can be of the order of $\sum_{i=1}^{n} |C_i|$, where, $|C_i|$ denote the set of clocks in the automaton of the ith component. Generation of a global automaton from the component automata is analogous to composition of a global meta-policy in the multi-domain environment. Verification of the timed scenario involves finding at least one timed trace in the global automaton that matches with the given scenario. For determining such matching, the scenario is transformed into a timed automaton and a parallel (product) composition of the global automaton and the automaton corresponding to the scenario is generated. The resulting automaton, called the composite automaton, is then analyzed for satisfaction of a timed computation tree logic TCTL formula, which states whether the accepting state of the composite automaton can be reached from its initial state. The satisfiability of such formula implies that the given scenario can be supported by the component systems. The complexity of verifying scenario matching using TCTL based model checking is linear in the number of states and exponential in the number of clocks of the composite automaton [4, 5].

Table 5.2
Comparative analysis of approaches verifying service composition

Approach	Model used for specifying component system policies/behavior	Real-time/ Non real-time	Reentrant/ Non - reentrant behavior	Disclos-ure of policies	Global Meta-policy Generation	Interaction Modeling	Verification Criteria	Verification Complexity
Berardi et. al. [17]	The behavior of each component is modeled using a separate FSM	Non real-time	Re-entrant	Yes	A mealy finite state machine MFSM is generated from the FSM of component system. Complexity: Exponential in the size of the FSM of all component systems/services	The behavior target service/system is also specified using FSM	Checking whether the execution tree of the MFSM is equivalent to the execution tree generated by the FSM of target system/service	Exponential in the size of the FSM of all component systems/services
Betin-Can et. al. [32]	The interface of each component system/service is modeled using FSM	Non real-time	Re-entrant	Internal policy or behavior of a compon-ent is not disclosed. The beha-vior of a component is assumed to be con-sistent with its interface policy	No	A composite service is modeled as a state transition system specifying the message exchange between the component service interfaces.	**Interface Verification**: The internal policy or behavior of component conforms to its interface specifications. **Behavior Verification**: Safety and liveness properties of the composite system/service Assumption: The behavior of a component con-sistent with its interface policy	**Interface Verification:** need to traverse all possible execution paths that can be generated from the internal policy/program of the component. **Behavior Verification:** The safety and liveness properties are specified using LTL formula. The complexity of verifying an LTL formula is linear in the size of the number of states of the composite service and exponential in the size of the LTL formula [115].
Proposed approach for workflow composibility verification	Timed automaton	Real-time	Non-reentrant	No	No	Distributed workflow is specified using UML-based sequence diag.	Finding time traces in the FSM of component domains that can support the distributed workflow.	#P-Complete

Table 5.3
Comparative analysis of distributed protocol verification approaches

Approach	Model used for specifying component system policies/behavior	Real-time/ Non real-time	Reentrant/ Non - reentrant behavior	Disclos-ure of policies	Global Meta-policy Generation	Interaction Modeling	Verification Criteria	Verification Complexity
Brand *et. al.* [31]	The behavior of each component is described using a protocol represented as FSM. The state transitions in the FSM of a component protocol occur due to reception or transmission of messages from or to other peer components.	Non real-time	Re-entrant (multiple instances of a protocol can be created).	A receiver upon receiving a message should be able to identify the current state of the sender peer.	No	Only the messages that can be exchanged between the peers are described. No global protocol or scenario is specified.	**Executable reception**: A component peer should be able to process any received message from its current state. **Stable N-tuple**: Can a global state be reached in which the input queues of all peers are empty. The existence of such a state may imply a deadlock.	Decidable provided the following conditions hold: 1. The protocol of each component can be transformed into a tree protocol. 2. The queue size of each component is bounded. **Verification complexity:** exponential in the size of the FSM of each component.
Fu. *et. al.* [51**]**	Each component is specified in terms of the messages it can receive or transmit.	Non real-time	Re-entrant (A new instance of a component service can be created for a new session).			A non deterministic Buchi automaton is used to specify the desired set of conversation (conversation protocol), among the components).	Given the set of input and output alphabets of each component service and a conversation protocol, verify whether the conversation protocol can be realized. Alternatively, can we synthesize a set of finite state peers such that the synthesize components conform to the conversation protocol.	Fu *et. al*. have defined necessary and sufficient conditions for realizability of conversation protocols: lossless-join condition, synchronous compatible condition, and autonomous condition. Verification complexity is EXPTIME in the size of the automaton modeling the conversation protocol.
Proposed approach for workflow composibility verification	Timed automaton	Real-time	Non-reentrant	No	No	Distributed workflow is specified using UML-based sequence diag.	Finding time traces in the FSM of component domains that can support the distributed workflow.	#P-Complete

Table 5.4
Approach for verification of distributed real-time system

Approach	Model used for specifying component system policies/behavior	Real-time/ Non real-time	Reentrant/ Non - reentrant behavior	Disclosure of policies	Global Meta-policy Generation	Interaction Modeling	Verification Criteria	Verification Complexity
Braberman *et. al*. [30, 29]	The behavior of each component is modeled using a separate timed automaton	Real-time	Both	Yes	A product/ composite automaton is generated from the component automata and the scenario to be verified.	Uses visual timed event scenarios to specify the desirable or undesirable properties of the integrated system	Verifies whether here exists a timed trace in the composite automaton that matches with the given scenario.	Uses a TCTL based model checking technique. **Complexity of TCTL Model Checking**: linear in the number of states in the composite automaton and exponential in the number of clocks of the composite automaton [4, 5].
Proposed approach for workflow composibility verification	Timed automaton	Real-time	Non-reentrant	No	No	Distributed workflow is specified using UML-based sequence diag.	Finding time traces in the FSM of component domains that can support the distributed workflow.	#P-Complete

6. CONCLUSION AND FUTURE WORK

In this chapter, we summarize the contributions of this dissertation and discuss future research directions

6.1. Research Contributions

In this dissertation, we have focused on policy-based access management and secure interoperation in distributed collaborative systems. In particular, we have developed a policy-based framework that allows secure information and resource sharing in multi-domain environments with varying degree of coupling among the collaborating domains. The framework, proposed in this dissertation, provides efficient solution and strategies for ensuring secure interoperation in both federated and loosely-coupled multi-domain environments based on the degree of interoperation, the level of trust among domains, and the security, autonomy, and privacy requirements of collaborating domains.

For establishing secure interoperation in a federated multi-domain environment, we have proposed a policy composition approach that generates a global meta-policy from the local access control policies of collaborating domains. This approach is designed for multi-domain systems employing RBAC policies. The global meta-policy is generated from the RBAC policies of the collaborating domains by defining role mappings across domains. Such mappings enable inter-domain information and resource sharing via mapped roles. The RBAC policies of domains may have conflicting security and access control requirements which may cause serious security implications in terms of unauthorized accesses and erroneous system behavior. To resolve such inconsistencies and conflicts in the meta-policy, we have proposed a systematic approach for policy synthesis and conflict resolution with various optimality measures, including, maximizing overall information accessibility, maximizing prioritized accesses, and minimizing constraint relaxation. Conflict resolution may require strong mediation among domains' policies, and may trigger policy transformations to support secure collaboration. Such transformations in policies, although increase interoperation among collaborating domains, may result in a loss of their autonomy. A key requirement for developing the global meta-policy is to allow maximum autonomy. Although, violations of domain's security policy are generally not permissible, some domains may concede their autonomy for allowing an increased level of interoperability. In the proposed approach, the problem of secure interoperation is formulated as an optimization problem with an objective of maximizing interoperability with minimum autonomy losses and without causing any security violations of collaborating domains. This optimization problem is solved using 0-1 integer programming based technique with the given optimality measure.

We have also addressed the issue of developing distributed service or workflow based applications requiring secure information and resource sharing among autonomous domains in a loosely-coupled multi-domain environment. For verifying the correctness and composibility of such distributed workflows/services, we have proposed a verification approach that analyzes the workflow/service specifications for conformance with the policies of all collaborating domains. A key challenge in this verification is posed by the time-dependent policies of collaborating domains which are specified using GTRBAC model. The GTRBAC policy of a domain contributes to its non-reentrant behavior which is modeled as a time augmented FSM. The proposed approach verifies workflow/service composibility by exploring the FSM of each domain to find state paths that satisfy the given workflow/service specifications. This workflow composibility verification is performed without creating a unified global FSM which is required for model checking-based approaches for composibility verification. Therefore, the proposed approach is unique and does not compromise the autonomy and privacy of collaborating domains.

We believe that the verification approach presented in this dissertation is generic and can be applied to many distributed applications involving collaborations among non-reentrant and autonomous components. Examples of such applications include process control systems [73], mission planning and control in military systems [38], real-time speech recognition systems [44], and workflow-based production systems [95]. The underlying verification problem in such applications is to determine whether or not a given configuration of non-reentrant components can support the functionality required by distributed applications.

We have analyzed the trade-offs between the global meta-policy and distributed multi-policy based approaches for establishing secure interoperation. The trade-off between these two approaches is analyzed with respect to various metrics. We have also presented a comparison between the proposed policy composition and verification framework with the existing approaches for verification of distributed systems and services.

6.2. Future Work

The research work reported in this dissertation provides a foundation to explore several research avenues in the area of information and system security. Below, we summarize several directions in which our work can be pursued.

6.2.1. Policy verification of individual domains

The underlying assumption, while designing secure collaborative applications, is that the policy of each collaborating domain is consistent, and conflict-free. In case the policy of any of the collaborating domain is inconsistent or has security flaws, then the entire process of policy composition and verification of distributed information sharing applications described in this dissertation fails. Therefore, the policies of domains need to be verified before a global meta-policy is generated or distributed applications are designed.

Security policy verification in general is an undecidable problem [65]. However, much work has been done to determine reasonable models and limitations under which safety is decidable and tractable [6, 7, 8, 122, 70 82]. Verification of a domain's policy entails various challenges, including: i) specifying policy using a formal model, ii) identifying the safety requirements, and iii) determining if a given policy conform to the safety requirements. Generally the safety requirements are specified in the form of constraints. These constraints can be 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.

For verifying the correctness of policies, we plan to use the model checking and scenario matching based techniques. Deciding the correctness or consistency of a policy is one aspect of the verification problem. The other aspect is to guide the policy designers or administrators to resolve conflicts from an inconsistent policy. Conflicts in a given policy 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, or minimum deviation from original policy design etc. We believe that the Integer Programming based approach discussed in the context of policy integration can be used to resolve the conflicts present in a domain's policy.

6.2.2. Policy partitioning for enterprise splitting

In an 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.

6.2.3. Software testing of access control mechanisms

Testing of the software systems implementing the security and access control mechanisms is indispensable even if their policy specifications are formally verified. The formal verification techniques only ensure the correctness of the specifications or design under certain assumptions and cannot guarantee correct implementation of such specifications or design. A mathematical proof about the conformance of an implementation with the policy specification is usually not feasible because it would require complicated formal semantics of the language in which the implementation is coded and the environment in which it runs (operating system and hardware).

We are interested in developing a toolkit for testing the implementation of the access control systems. In this regard, we plan to investigate the efficacy of model-based techniques from software engineering area to identify the errors and flaws in the software components enforcing the access control policies. A significant advantage of using model-based testing techniques is that the policy models developed in the specification and verification phase can be directly used to generate the test cases.

6.2.4. Digital identity and privacy management

Internet-based business transactions often involve exchange of confidential and sensitive information including personal data, financial details, and business data among the interacting parties. Particularly interesting is the case where such transactions span across multiple parties due to sub-contracting, outsourcing, and integration of services supplied by multiple providers. In such scenarios, disclosure of personal identity and profile information can be used to simplify users' experience and enable single sign-on to reduce the overhead associated with the repeated exchange of information at different authentication points. However, managing multiple versions of users' identities across several service provider domains is a key challenge for ensuring information security and privacy in multiparty transactions. Another challenging aspect of the single sign-on access comes from the diverse or possibly contradictory access control policies of the domains involved in multiparty transactions. In this respect, there is a close synergy between identity management in a multiparty transaction and policy composition and verification problem addressed in this dissertation. We plan to expand our work on access management in collaborative systems to address these issues. In addition, we are interested in developing mechanisms for identity provisioning and lifecycle management, identity interoperability and extensibility, and cross-domain communication and mobility.

LIST OF REFERENCES

LIST OF REFERENCES

[1] N. R. Adam, V. Athluri, W. Huang. "Modeling and Analysis of Workflows Using Petri Nets." *Journal of Intelligent Information Systems*, Vol. 10, pp. 131-158, 1998.

[2] J. R. Agre, S. K. Tripathi, "Modeling Reentrant and Nonreentrant Software." *Proc. ACM SIGMETRICS Conference on Measurement and modeling of computer systems*, 12(3), pp.163-178, 1982.

[3] G. Ahn, R. Sandhu, "Role-Based Authorization Constraints Specification," *ACM Transactions on Information and System Security*, Vol. 3 No. 4, November 2000.

[4] R. Alur, C. Courcoubetis, and D. Dill, "Model Checking in Dense Real-Time," *Information and Computation*, Vol. 104, No. 1, 1993, pp. 2-34.

[5] R. Alur and D. Dill, "A Theory of Timed Automata," *Theoretical Computer Science*, Vol. 126, No. 2, 1994, pp. 183-235.

[6] 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

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

[8] 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.

[9] A. I. Antón and J. B. Earp. "Strategies for Developing Policies and Requirements for Secure E-Commerce Systems." *Recent Advances in E-Commerce Security and Privacy*, Kluwer Academic Publishers, pp. 29-46, 2001.

[10] 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.

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

[12] J. Barkley, A. Cincotta, D. Ferraiolo, S. Gavrila, and D.R. Kuhn, "Role Based Access Control for the World Wide Web," in *proc. of the 20th National Information System Security Conference*, NIST/NSA, 1997.

[13] 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.

[14] A. Belokosztolszki and K. Moody, "Meta-Policies for Distributed Role-Based Access Control Systems," *proc. of the IEEE International Workshop on Policies for Distributed Systems and Networks*, 2002.

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

[16] D. E. Bell and L. J. LaPadula, "Secure Computer System: Unified Exposition and Multics Interpretation," MTR-2997, MITRE Corp., Bedford, MA, March, 1976. Available as NTIS AD A023 588.

[17] D. Bedrardi, D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Mecella, "Automatic Composition of E-Services that Export Their Behavior," *Proc. of International Symposium on Service Oriented Computing*, 2003, pp. 43-58.

[18] D. Bedrardi, D. Calvanese, G. D. Giacomo, R. Hull, and M. Mecella, "Automatic Composition of Transition Based Semantic Web services with Messaging," *proc. of the the 31st VLDB Conference*, Trondheim, Norway, 2005, pp. 613-623.

[19] 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*, Vol.2, No. 1, pp. 65-104. 1999.

[20] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati, "An Access Control Model Supporting Periodicity Constraints and Temporal Reasoning," *ACM Transactions on Database Systems*, Vol. 23, No. 3, pp. 231-285.

[21] 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, pp. 65-104.

[22] E. Bertino, F. Buccafurri, E. Ferrari, and P. Rullo, "A Logical Framework for Reasoning on Data Access Control Policies," in *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, 1999, pp. 175-189.

[23] 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, pp. 206-214.

[24] R. Bhatti, B. Shafiq, J. Joshi, E. Bertino, and A. Ghafoor, "XGTRBAC Admin: A Decentralized Administration Model for Enterprise-Wide Access Control ," *ACM Transactions on Information and System Security*, Vol. 8, Issue 4, Nov. 2005, pp. 388-423.

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

[26] P. Bonatti, S.D.C. Vimercati, and P. Samarati, "An Algebra for Composing Access Control Policies," *ACM Transactions on Information and System Security*, Vol. 5, No. 1, February 2002.

[27] Y. Bontemps, P. Heymans, and P-Y Schobbens, "From Live Sequence Charts to State Machines and Back: A guided Tour," *IEEE Transactions on Software Engineering*, Vol. 31, No. 12, pp. 999-1013.

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

[29] V. Braberman, D. Garbervetsky, A. Olivero, "Improving the Verification of Timed Systems Using Influence Information," *Proc. of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, *Lecture Notes In Computer Science*, Vol. 2280, 2002, pp. 21 – 36.

[30] V. Braberman, N. Kicilof, and A. Olivero, "A Scenario-Matching Approach to the Description and Model Checking of Real-Time Properties," *IEEE Transactions on Software Engineering*, Vol. 31, No. 12, pp. 1028-1041.

[31] D. Brand and P. Zafiropulo, "On Communicating Finite-State Machines," *Journal of the ACM*, Vol. 30, No. 2, April 1983, pp. 323.342.

[32] A. B-Can, T. Bultan, and X. Fu, "Design for Verification for Asynchronously Communicating Web Services," *Proc. of the Fourteenth International World Wide Web Conference (WWW 2005)*, pp. 750-759.

[33] J. Cao, S.A.Jarvis, S. Saini and G. R. Nudd, "GridFlow: Workflow Management for Grid Computing," *proc. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid03)*, pp.198 – 205. 2003.

[34] C. Canright, "Will Real-Time Systems Spell the End for Batch Processing," *Bank Admin*., Vol. 64, No. 9, September 1988, pp. 42-46.

[36] S. A. Chun, V. Atluri, and N. R. Adam, "Using Semantics for Policy-Based Web Service Composition," *Distributed and Parallel Databases*, Vol. 18, 2005, pp. 37-64.

[37] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic Verification of Concurrent Systems Using Temporal Logic Specifications," *ACM Transactions on Programming languages and Systems*, Vol. 8, No. 2, pp. 244-263.

[38] 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.

[39] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, S. Weerawarana, "Unravelling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI," *IEEE Internet Computing*, Vol 6, No. 2, March 2002, pp. 86-93.

[40] S. Dawson, S. Qian, and P. Samarati, "Providing Security and Interoperation of Heterogeneous Systems," *Distributed and Parallel Databases*, Vol. 8, August 2000, pp. 119 -145.

[41] Y. Deng, J. Wang, J.J.P Tsai and K. Beznosov, "An Approach for Modeling and Analysis of Security System Architectures," *IEEE Transactions on Knowledge and Data Engineering*, 15(5), pp.1099 - 1119, 2003.

[42] J. Eder, E. Panagos, Michael Robinovich, "Time Constraints in Workflow Systems," *Proc. of 11th Int. Conf. on Adv. Inf. Systems Engineering (CAiSE 99)*, Heidelberg, Germany, 1999.

[43] 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.

[44] L. D. Erman, F. H.-Roth, V. R. Lesser, and D. R. Reddy, "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty," *ACM Computing Surveys*, June 1980, pp. 213-252.

[45] R. Eshuis, R. Wieringa, "Tool Support for Verifying UML Activity Diagrams," *IEEE Transactions on Software Engineering*, 30(7), pp. 437 – 447, 2004.

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

[47] 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.

[48] 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, pp. 107-116.

[49] H. Foster, S. Uchitel, J. Magee, and J. Kramer, “Model-based Verification of Web Service Compositions,” *proc. of the IEEE International Conference on Automated Software Engineering*, 2003.

[50] X. Fu, T. Bultan, and J. Su, “Formal Verification of e-Services and Workflows,” *proc. of Workshop on Web Services, E-Business, and the Semantic Web*, May 2002, pp. 188-202.

[51] X. Fu, T. Bultan and J. Su. "Conversation Protocols: A Formalism for Specification and Verification of Reactive Electronic Services." *Theoretical Computer Science (TCS)*, Vol. 328, No. 1-2, November 2004, pp. 19-37.

[52] D. Gabbay, A. Pnuelli, S. Shelah, and J. Stavi, “On the Temporal Analysis of Fairness,” *proc. of Seventh ACM Symposium on Principles of Programming Languages*, 1980, pp. 163-173.

[53] C. Gacek, “Detecting Architectural Mismatches during Systems Composition,” PhD. Thesis, University of Southern California, 1998.

[54] S. Garfinkel, E. H. Spafford, “Practical UNIX & Internet Security,” O'Reilly & Associates, Inc., 2nd Edition, April 1996.

[55] S. Garfinkel, E. H. Spafford, “Web Security & Commerce,” O'Reilly & Associates, Inc., Sebastapol, CA, 1997.

[56] S. I. Gavrila , J. F. Barkley, “Formal Specification for Role Based Access Control User/role and Role/role Relationship Management,” in *Proc. of the ACM Workshop on Role-Based Access Control*, Fairfax, Virginia, United States, pp. 81-90, October 1998.

[57] S. Ghosh, "NOVADIB: a Novel Architecture for Asynchronous, Distributed, Real-Time Banking Modeled on Loosely-coupled Parallel Processors," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 23, No. 3, May 1993, pp. 917-927.

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

[59] L. Giuri. "Role-based Access Control: A Natural Approach," in *proc. of the 1st ACM Workshop on Role-Based Access Control*. ACM, 1997.

[60] J. V-Gomez, "Multidomain Security," Computer & Security, Vol. 13, 1994, pp. 161-184.

[61] L. Gong and X. Qian, "Computational Issues in Secure Interoperation," *IEEE Transaction on Software and Engineering,* Vol. 22, No. 1, pp.43-52, 1996.

[62] G. Graham and P. Denning, "Protection -- principles and practice," in *proc. Spring Joint Computer Conference*. AFIPS Press, 1972.

[63] 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.

[64] D. Harel, D. Kozen, and J. Tiuryn. Dynamic Logic. The MIT Press, 2000.

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

[66] Q. He and A.I. Antón, "Deriving Access Control Policies from Requirements Specifications and Database Designs," NCSU CSC Technical Report #TR-2004-24, 2004.

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

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

[69] ITU. Message Sequence Charts. Recommendation Z.120, International Telecomm. Union, Telecomm. Standardization Sector, 1996.

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

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

[72] S. Jajodia, P. Samarati, V. S. Subrahmanian, E. Bertino, "A Unified Framework for Enforcing Multiple Access Control Policies," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, May 1997, pages 474-485.

[73] M. S. Jaffe, N. G. Leveson, M. P. E. Heimdahl, and B. E. Melhart, "Software Requirements Analysis for Real-Time Process-Control Systems," *IEEE Transactions on Software Engineering*, Vo. 17, No. 3, pp. 241-258.

[74] 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, pp. 38-72.

[75] 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, pp. 66-72.

[76] 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.

[77] 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.

[78] J. Joshi, E. Bertino, U. Latif and A. Ghafoor. A Generalized Temporal Role Based Access Control Model for Developing Secure Systems. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17, No. 1, pp. 4-23, 2005.

[79] J. Joshi, E. Bertino, and A. Ghafoor, "Analysis of Expressiveness and Design Issues for a Temporal Role Based Access Control Model," *IEEE Transactions on Dependable and Secure Computing*, Vol. 2, No. 2, pp. 157-175.

[80] J. Jung, W. Hur, S. Kang and H. Kim, "Business Process Choreography for B2B Collaboration," *IEEE Internet Computing*, 8(1), pp. 37-45, 2004.

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

[82] 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.

[83] I. Krüger, “Service Specification with MSCs and Roles,” *proc. IASTED International Conference on Software Engineering*, Innsbruck, 2004.

[84] 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, pp. 177-228.

[85] B. Lampson, “Protection,” In the Princeton Symposium on Information Sciences and Systems, March 1971. Reprinted in *ACM Operating Systems Review*, 8(1) (1974).

[86] B. Lampson, “A note on the Confinement Problem,” Communications *of the ACM*, 16(10), October 1973, pages 613-615.

[87] B. W. Lampson, “Computer Security in the Real World,” *Annual Computer Security Applications Conference*, December 11-15, 2000.

[88] X. E. Landwehr, “Computer Security,” *International Journal of Information Security*, Vol. 1, No. 1, August 2001, pp. 3 – 13.

[89] Ninghui Li, John C. Mitchell, and William H. Winsborough, “Design of A Role-based Trust-management Framework, “ Proc. IEEE Symposium on Security and Privacy, May 2002, pp. 114-130.

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

[91] F. J. Lin, P. M. Chu, and M. T. Liu, “Protocol Verification Using Reachability Analysis: The State Space Explosion Problem and Relief Strategies,” *proc. of the ACM SIGCOMM’87 Workshop*, pp. 126-135.

[92] 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.

[93] E. Lupu and M. Sloman, "Conflicts in Policy-based Distributed Systems Management," *IEEE Transactions on Software Engineering*, Vol 25, No. 6, November 1999, pp.. 852-869.

[94] J. McLean, “Security Models and Information Flow,” *In Proceedings 1990 IEEE Symposium on Security and Privacy*, Oakland, CA, 1990, pages 180—187.

[95] M. Zur Muehlen. Workflow-based Process Controlling: Foundation, Design and Application of Workflow-Driven Process Information Systems. Logos, Berlin, 2004.

[96] T. M. Nguyen, A. M. Tjoa, G. Kickinger, and P. Brezany, “Towards Service Collaboration Model in Grid-Based Zero Latency Data Stream Warehouse (GZLDSWH),” *proc. of the IEEE International Conference on Service Computing*, 2004.

[97] M. Niezette and J. Stevenne, “An Efficient Symbolic Representation of Periodic Time,” *Proc. of First International Conference on Information and Knowledge Management*, November 2-5, 1992.

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

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

[100] OMG. Unified Modeling Language Specification: version 2.0. Object Management Group Inc. www.uml.org. 2003.

[101] 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.

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

[103] D. Pilone and N. Pitman. UML 2.0 in a Nutshell. O’Reilly Press, 2005.

[104] R. Pottinger and P. A. Bernstein, “Merging Models Based on Given Correspondences,” *VLDB 2003*, pp. 826-873.

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

[106] X. Qian and T. F. Lunt, “A MAC Policy Framework for Multilevel Relational Databases,” *IEEE Transactions on Knowledge and Data Engineering*, Vol. 8, No. 1, pp. 3 – 15, February 1996.

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

[108] R. Sandhu and P. Samarati, “Access Control: Principles and Practice,” *IEEE Communications Magazine*, Vol. 32, No. 9, September 1994, pp. 40-48.

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

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

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

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

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

[114] R. Sandhu, “Role Activation Hierarchies,” in *Proc.of the third ACM workshop on Role-based access control*, pp.33-40, October 22-23, 1998.

[115] P. Schnoebelen, “The Complexity of Temporal Logic Model Checking,” *Advances in Modal Logic*, Vol 4, 2002, pp. 1-44.

[116] K. E. Seamons, M. Winslett, and T. Yu, “Limiting The Disclosure of Access Control Policies during Automated Trust Negotiation,” *Proc. Workshop on Privacy Enhancing Technologies*, April 2002.

[117] B. Shafiq, J. Joshi, E. Bertino, and A. Ghafoor, “Secure Interoperation in a Multi-Domain Environment Employing RBAC Policies,” *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17, No. 11, pp. 1557-1577.

[118] A. P. Sheth and J. A. Larson, “Federated Database Systems for managing Distributed, Heterogeneous, and Autonomous Databases,” *ACM Computing Surveys*, Vol. 22, No. 3, September 1990, pp. 184-236.

[119] M. Shehab, E. Bertino, and A. Ghafoor, “Secure role mApping technique for decentralized secure interoperability,” *proc. of the 10th ACM Symposium on Access Control models and Technologies (SACMAT 05)*, June 2005.

[120] M. Shehab, E. Bertino, and A. Ghafoor, "Secure Collaboration in Mediator-Free Environments," *Proc. of the 12th ACM Conference on Computer and Communications Security (CCS)*, November 2005.

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

[122] 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.

[123] A. Tripathi, T. Ahmed, and R. Kumar, "Specification of Secure Distributed Collaboration Systems," in *proc of the IEEE International Symposium on Autonomous Distributed Systems (ISADS)*, pp. 149–156, 2003.

[124] A. Tripathi, T. Ahmed, D. Kulkarni, R. Kumar and K. Kashiramka, "Context-Based Secure Resource Access in Pervasive Computing Environments," *proc. of the 1st IEEE International Workshop on Pervasive Computing and Communications Security(IEEE PerSec'04)*, pp.159 – 163, 2004.

[125] L. G. Valiant, "The Complexity of Enumeration and Reliability Problems," *SIAM Journal on Computing*, Vo. 8, No. 3, August 1979, pp. 410-421.

[126] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman and S. Tuecke, "Security For Grid Services," in *proc. of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'12)*, pp. 48-57, 2003.

[127] C. H. West, "Protocol Validation in Complex Systems," in *proc. Symposium on Communications Architecture and Protocols*, Austin, Texas, 1989, pp. 303-312.

[128] W. Winsborough and N. Li, "Towards Practical Automated Trust Negotiation," in *proc. IEEE Workshop on Policies for Distributed Systems and Networks*, June 2002, pp. 92-103.

[129] W. Winsborough and N. Li. "Safety in Automated Trust Negotiation," in *proc. IEEE Symposium on Security and Privacy*, May 2004, pp. 147-160.

[130] D. Wodtke, J. Weissenfels, G. Weikum, and A. K. Dittrich, "The Mentor Project: Steps towards Enterprise-Wide Workflow Management," in *proc. of the 12th IEEE International Conference on Data Engineering*, 1996.

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

[132] G. Yan, W. K. Ng, E. Lim, "Product Schema Integration for Electronic Commerce - A Synonym Comparison Approach," *IEEE Transactions on Knowledge and data Engineering,* Vol. 14, No. 3, June 2002, pp. 583-598.

[133] T. Yu and M. Winslett, "A Unified Scheme for Resource Protection in Automated Trust Negotiation," *Proc. IEEE Symposium on Security and Privacy*, May 2003.

[134] T. Yu and M. Winslett, "Policy Migration for Sensitive Credentials in Trust Negotiation," *Proc. ACM Workshop on Privacy in the Electronic Society*, 2003, pp. 9-20.

[135] J. Yu. Dynamic "Web Service Invocation based on UDDI," proc of the IEEE International Conference on E-Commerce Technology for Dynamic E-Business, pp. 154 – 157, 2004.

APPENDICES

Appendix A.

Proofs of Lemmas and Theorems of Chapter 3

Proof of Lemma 3.1: The split function, given in Fig. 3.11, 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 3.2: 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 3.3

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 3.2 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. □

Proof of Theorem 3.2 (Associativity of *RBAC-integrate*): 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 [20]. 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 $pset_{\text{assign}}(r') = pset_{\text{assign}}(r)$

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).

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

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

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 $pset_{\text{assign}}(r_{BA}) = pset_{\text{assign}}(r)$ 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 $pset_{\text{assign}}(r_{BA}) \cap pset_{\text{assign}}(r_A)$. This means that there exists a role r' in P with $pset_{\text{assign}}(r') = pset_{\text{assign}}(r_{BA}) \cap pset_{\text{assign}}(r_A) = pset_{\text{assign}}\,(r_{BA}) = pset_{\text{assign}}\,(r)$. Also, when integrating P with G_C role r' is not split nor the permission in the set $pset_{\text{assign}}(r')$ 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 $pset_{\text{assign}}(r') = pset_{\text{assign}}(r_B) = pset_{\text{assign}}(r_{BA}) \cap pset_{\text{assign}}(r_A)$. This means that there exists a role r' in P with $pset_{\text{assign}}(r') = pset_{\text{assign}}\,(r_{BA}) = pset_{\text{assign}}\,(r)$. Also, when integrating P with G_C role r' is not split nor the permission in the set $pset_{\text{assign}}(r')$ 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 $pset_{\text{assign}}(r') = pset_{\text{assign}}(\underline{r})$, i.e., $\varphi(r') = r$. In a similar manner, we can prove the above for cases b and c as well. Hence, for all type (ii) roles $r \in$ Y, there exists a role $r' \in$ X such that $pset_{\text{assign}}(r') = pset_{\text{assign}}(r)$, 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 $pset_{\text{assign}}(r') = pset_{\text{assign}}(r)$. 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 $pset_{\text{assign}}(r) \subset pset_{\text{assign}}(r'')$. This and the element preservation property implies that there exists a role r' $\in$ X, such that $pset_{\text{assign}}(r') = pset_{\text{assign}}(r)$.

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 $pset_{\text{assign}}(r_j') = pset_{\text{assign}}(r_j)$.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 $pset_{\text{assign}}(r') = pset_{\text{assign}}(r)$. Suppose this is not the case and $pset_{\text{assign}}(r') \neq pset_{\text{assign}}(r)$. Both r_j and $r_j' \in$ *dom(A)*, which implies that $r' \in$ *dom(A)*. Suppose that $pset_{\text{assign}}(r') \supset pset_{\text{assign}}(r)$. 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 $pset_{\text{assign}}(r') \supset pset_{\text{assign}}(r)$, implying that either $pset_{\text{assign}}(r_j') \subset pset_{\text{assign}}(r_j)$ 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 $pset_{\text{assign}}(r_{j2}') = pset_{\text{assign}}(r_{j2})$. Nevertheless, r_{j2}' resulted from the split of role r'. This implies that all the permissions in the $pset_{\text{assign}}(r') \setminus pset_{\text{assign}}(r)$ are removed from r' and are assigned to r_{j2}'. Therefore, $pset_{\text{assign}}(r) \not\subset pset_{\text{assign}}(r')$

If we assume $pset_{\text{assign}}(r') \subset pset_{\text{assign}}(r)$ then, either $pset_{\text{assign}}(r_j') \supset pset_{\text{assign}}(r_j)$ 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 $pset_{\text{assign}}(r_{j2}') = pset_{\text{assign}}(r) \setminus pset_{\text{assign}}(r')$. 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 $pset_{\text{assign}}(r_{j2})$ = $pset_{\text{assign}}(r)$ $\cap$ $pset_{\text{assign}}(r'')$ =

$pset_{\text{assign}}(r_{j2}')$. This proves that $pset_{\text{assign}}(r') \not\subset pset_{\text{assign}}(r)$ provided r is split once or twice. Using induction we can prove that $pset_{\text{assign}}(r') \not\subset pset_{\text{assign}}(r)$ 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 $pset_{\text{assign}}(r') = pset_{\text{assign}}(r)$, 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 $pset_{\text{assign}}(r) = pset_{\text{assign}}(\varphi(r))$. Moreover, φ is a 1:1 morphism. This implies that for any permission *p*, *P-assign*(*r,p*) ∈ R_X if and only *P-assign*(φ(*r*),*p*) ∈ R_Y.

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

Proof of Theorem 3.3: We prove this theorem separately for *role assignment*, *role-specific SoD*, and *user-specific SoD* constraints.

Sub-proof 1: Any state S reachable from multi-domain RBAC graph G is secure with respect to the role-assignment constraint of all collaborating domains. We prove this claim by contradiction. Suppose that the above statement 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 $\left(u_{ir_l} = 1\right) \wedge \left(r_m \underset{I}{\geq}^* r_n \vee r_m = r_n\right) \wedge \left(r_p \underset{I}{\geq}^* r_j \vee r_p = r_j\right)$.

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: 1) $u_{ir_n} = 1$. 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). 2) $u_{ir_n} = 0$. 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.

Sub-proof 2: 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 four sub-cases corresponding to case 1, These sub-cases are shown in Figs. A.1(a-d).

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 Fig. A.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 Fig. A.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 Fig. A.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 $r_i \in conf\text{-}rset(r_t)$. 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 $r_u \in \mathrm{R_k}$ 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 $r_o \notin \mathrm{R_k}$ 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.

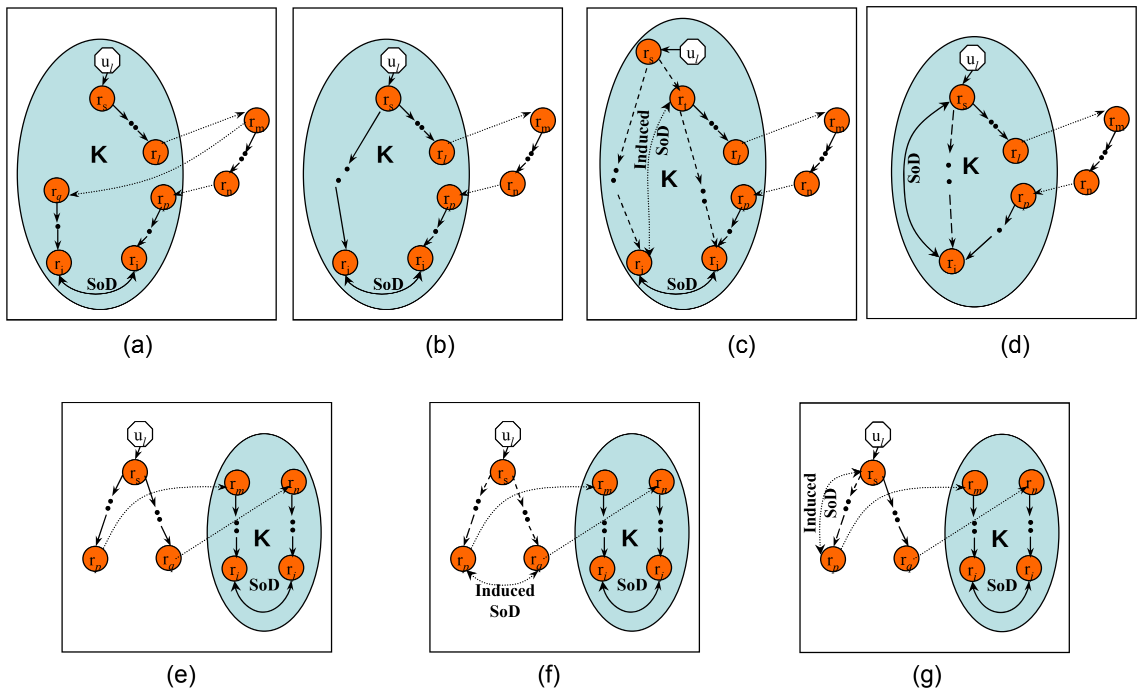


Fig. A.1 Cases of role-specific *SoD* violations involving cross-domain paths

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 Fig. A.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 $\in$ R$_k$ and r_m, r_n $\notin$ 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 A.1(e), A.1(f) and A.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 Fig. A.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 Fig. A.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 Fig. A.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 Fig. A.1(g). These cross-domain relationship $r_p \underset{I}{\geq}^* r_i$ and $r_s \underset{I}{\geq}^* r_j$ induces an SoD constraint between r_p and r_s as shown in Fig. A.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.

Sub-proof 3: 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 Fig. A.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 Fig. A.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, $r_n \notin R_k$, 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 Fig. A.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 Fig. A.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:

1. $u_{ir_n} = 1$. 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).
2. $u_{ir_n} = 0$. 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).

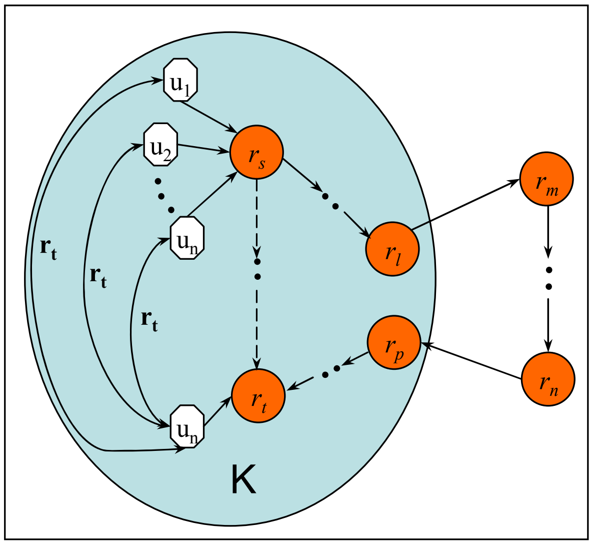


Fig. A.2 User-specific SoD violation through a cross-domain path

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.

This concludes the proof of Theorem 3.3. □

Appendix B.

Proofs of Theorems of Chapter 4.

Proof of Lemma 4.1: The workflow composibility conditions WC2 and WC3 establish a partial ordering among all tasks of the PW. For any task pairs τ_i and $\tau_j \in$ PW such that τ_i precedes τ_j in task execution order $\phi_j^{\pi'} \geq \phi_i^{\pi'} + \mathrm{D}$, where $0 < \mathrm{D} < \infty$. Moreover, if the delay between completion of τ_i and τ_j is bounded by the interval [$\mathrm{T}_{\min}$, $\mathrm{T}_{\max}$] then

$\mathrm{D}_1 + \phi_i^{\pi'} \leq \phi_j^{\pi'} \leq \mathrm{D}_2 + \phi_i^{\pi'}$ Where, $\mathrm{D}_1 = \mathrm{T}_{\min} + \mathrm{D}$, $\mathrm{D}_2 = \mathrm{T}_{\min} + \mathrm{D}$, and $\mathrm{D}_2 > \mathrm{D}_1 > 0$.

Note that the system of linear inequalities generated in step 1 of the task initiation time procedure does not have any constraint that bounds the sum of the initiation times of two or more tasks to a constant value. That is the system of linear inequalities generated in step 1 of the task initiation time procedure does not have any constraint of the form $\mathrm{M}_1 \leq \sum_i \phi_i^{\pi'} \leq \mathrm{M}_2$.

Therefore, minimizing the sum $\sum_i \phi_i^{\pi'}$ subject to the state residence time constraints and intra-domain workflow composibility constraints (WC1, WC2, and WC3) is equivalent to minimizing each individual $\phi_i^{\pi'}$. In other words, min($\phi_i^{\pi'}$) is the earliest time for initiation of task τ_i in state path π' such that the intra-domain workflow composibility conditions (WC1, WC2, and WC3) are satisfied. By the same argument, max($\phi_i^{\pi'}$) is the latest time for initiation of task τ_i. □

Proof of Theorem 4.1: We will first prove this theorem for the case when τ_j has only one parent node and then we will prove the multi-parent case. In both cases we will use inductive reasoning.

Single Parent case (τ_j has only one parent task node in G_X):

Base Case: The task τ_1 (first task of the PW) is the parent of τ_j. The edge mapping function ensures that all the state transition paths in the path set $\prod_{1j}$ satisfies intra-domain workflow composibility condition WC1, WC2, and WC3 for task pair τ_1 and τ_j.

Induction Step: Suppose τ_i ($\tau_i \neq \tau_1$) is the parent task node of τ_j i.e., $(\tau_i, \tau_j) \in E[G_X]$. Let $\prod_{1i}$ denotes the set of all valid state paths from τ_1 to τ_i that satisfy the intra-domain workflow composibility conditions WC1, WC2, and WC3 for task τ_i and all tasks that precede τ_i in the execution order of the PW. Let $\prod_{1j}$ be the set of all state paths from τ_1 to τ_j. The *for loop* in lines 2 -7 of the procedure *path-extend* called by *WPS* procedure ensures that for each $\pi_{1j} \in \prod_{1j}$, there exists a path $\pi_{1i} \in \prod_{1i}$ such that π_{1i} is a prefix sub-path of π_{1j}, i.e., the path relation *start*(π_{1i}, π_{1j}) holds.

The path set $\prod_{ij}$ returned by the edge-mapping procedure (in line 11 of *WPS* procedure), contains all paths that satisfy workflow composibility conditions WC1, WC2, and WC3 for the task pair τ_i and τ_j. Consider a path $\pi_{ij} \in \prod_{ij}$. Suppose π_i is a prefix path of π_{ij} such that the path relations *finish*(π_i, π_{1i}) and *start*(π_i, π_{ij}) hold, where $\pi_{1i} \in \prod_{1i}$. The path π_{ij} can be written as a concatenation of paths π_i and π' i.e, $\pi_{ij} = \pi_i . \pi'$. In lines 5 and 6 of the *path-extend* procedure the state path π_{1j} from τ_1 to τ_j is computed by concatenating π' to the end of π_{1i}. $\boldsymbol{\pi_{1j} = \pi_{1i} . \pi'}$.

Since π_{1j} include $\pi_{1i} \in \prod_{1i}$ as a sub-path, therefore all the intra-domain workflow composibility conditions that are true in π_{1i} are also true in π_{1j} for task τ_i and all tasks that precede τ_i in the execution order of the PW. π_{1j} also includes $\pi_{ij} \in \prod_{ij}$ and therefore, π_{1j} satisfies workflow composibility conditions WC1, WC2, and WC3 for the task pair τ_i and τ_j. Moreover, in line 7 of the *path-extend* procedure, π_{1j} is analyzed for satisfaction of workflow composibility WC1, WC2, and WC3 for all task pairs (τ_p, τ_q) such that $(\tau_p, \tau_q) \in E[G_X]$ and $\tau_q = \tau_j$ or τ_q precedes τ_j in the task execution order. This proves that path π_{1j} satisfies the two properties listed in the theorem statement.

Multiple parent case: We will prove the multi-parent case when τ_j is the first task node in G_X that has k parents with $k > 0$, as shown in Fig. B.1. The remaining multi-parent cases can be proved by similar reasoning.

Base case: k =1, the proof is similar to the single parent case.

Induction Step: Let $\prod_{1j}^{(k-1)}$ be the set of all state paths that satisfy: **(i)** workflow composibility condition WC1 for task τ_j and all predecessor tasks of τ_j that can reach τ_j via the task nodes $\tau_i^{(1)},\ldots,\ \tau_i^{(k-1)}$. **(ii)** workflow composibility conditions WC2 and WC3 between task pairs $(\tau_i^{(1)},\ \tau_j)$, ….$(\tau_i^{(k-1)},\ \tau_j)$ and all other tasks pairs $(\tau_p,\ \tau_q)$ such that $(\tau_p,\ \tau_q) \in E[G_X]$ and τ_q precedes $\tau_i^{(1)},\ldots,\ \tau_i^{(k-1)}$ in the task execution order.

Let $\prod_{1j}^{(k)}$ be the set of all state paths that satisfy: **(iii)** workflow composibility condition WC1 for task τ_j and all predecessor tasks of τ_j that can reach τ_j via the task nodes $\tau_i^{(k)}$. **(iv)** workflow composibility conditions WC2 and WC3 between task pairs $(\tau_i^{(k)},\ \tau_j)$ and all other tasks pairs $(\tau_p',\ \tau_q')$ such that $(\tau_p',\ \tau_q') \in E[G_X]$ and τ_q' precedes $\tau_i^{(k)}$ in the task execution order.

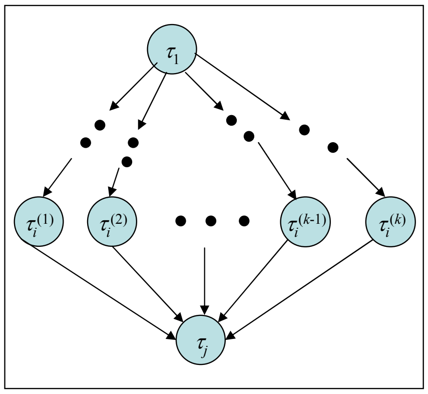


Fig. B.1

In the *WPS* procedure, the path set $\prod_{1j}$ is composed by taking an intersection of the path sets $\prod_{1j}^{(k-1)}$ and $\prod_{1j}^{(k)}$ (line 14). $\prod_{1j} = \prod_{1j}^{(k-1)} \cap \prod_{1j}^{(k)}$. Therefore, each path in the path set $\prod_{1j}$ satisfies (i), (ii), (iii), and (iv). Alternatively, each path $\pi_{1j} \in \prod_{1j}$ satisfies the following:

- Workflow composibility condition WC1 for task τ_j and all predecessor tasks of τ_j that can reach τ_j via its parent tasks.
- Workflow composibility conditions WC2 and WC3 between any task pair (τ_i, τ_j) such that $(\tau_i, \tau_j) \in E[G_X]$.
- Workflow composibility conditions WC2 and WC3 between any task pair (τ_p, τ_q) such that $(\tau_p, \tau_q) \in E[G_X]$ and τ_q precedes τ_j in the task execution order.

Proof of Theorem 4.2: The cross domain dependency verification procedure returns **No**, if there does not exist any path combination $(\pi_1, \pi_2, \ldots, \pi_n)$ (such that $\pi_k \in \Pi^{(k)}$ and $1 \le k \le n$), that satisfies the cross domain dependencies of the set CS_{dep}. Suppose on the contrary, that there exists a path combination $(\widehat{\pi_1}, \widehat{\pi_2}, \ldots, \widehat{\pi_n})$ that satisfies all the cross domain dependencies specified in the set CS_{dep} and each $\widehat{\pi_k}$ satisfies the composibility conditions WC1, WC2, and WC3 for the projected workflow PW_k assigned to ID_k. Since the verification procedure could not find the path combination $(\widehat{\pi_1}, \widehat{\pi_2}, \ldots, \widehat{\pi_n})$, the following two possibilities may occur:

1. At least one of the $\widehat{\pi_k}$ is not included in the path set $\Pi^{(k)}$, i.e, $\widehat{\pi_k} \notin \Pi^{(k)}$. As discussed in Section 4.5.1 of Chapter 4, the path set $\Pi^{(k)}$ returned by the procedure *WPS* is exhaustive and includes all the paths of ID_k that satisfy the composibility conditions WC1, WC2, and WC3 for PW_k. Therefore, $\widehat{\pi_k} \notin \Pi^{(k)}$ implies that $\widehat{\pi_k}$ cannot support PW_k. Hence, the path combination $(\widehat{\pi_1}, \widehat{\pi_2}, \ldots, \widehat{\pi_n})$ cannot support the distributed workflow.
2. $\forall\ k$, $\widehat{\pi_k} \in \Pi^{(k)}$. However, there exists at least one task $\tau_i \in PW_j$ such that $\phi_i^{\tilde{\pi}_j} \notin [\min(\phi_i^{\tilde{\pi}_j}), \max(\phi_i^{\tilde{\pi}_j})]$ or $\theta_i^{\tilde{\pi}_j} \notin [\min(\theta_i^{\tilde{\pi}_j}), \max(\theta_i^{\tilde{\pi}_j})]$, where $\phi_i^{\tilde{\pi}_j}$ and $\theta_i^{\tilde{\pi}_j}$ denote the initiation and completion times of τ_i in $\widehat{\pi_j} \in \Pi^{(j)}$. By Lemma 4.1, for any task $\tau_i \in PW_j$, $\min(\phi_i^{\pi})$, computed using the *task initiation time* procedure of Fig. 4.7, is the earliest time instant at which τ_i can be initiated in a state path π that satisfy

composibility conditions WC1, WC2, and WC3 for PW$_j$. Similarly, $\max(\phi_i^{\pi})$ corresponds to the latest initiation time of task τ_i. Since, $\widehat{\pi_j} \in \Pi^{(j)}$ and all paths in $\Pi^{(j)}$ satisfy WC1, WC2, and WC3 for PW$_j$, therefore, $\phi_i^{\tilde{\pi}_j} \in [\min(\phi_i^{\tilde{\pi}_j}), \max(\phi_i^{\tilde{\pi}_j})]$. Similarly, we can prove that $\theta_i^{\tilde{\pi}_j} \in [\min(\theta_i^{\tilde{\pi}_j}), \max(\theta_i^{\tilde{\pi}_j})]$.

With reference to the system of constraints (II) – (VIII) for cross-domain dependency verification, the above implies that all state paths in the combination $(\widehat{\pi_1}, \widehat{\pi_2}, \ldots., \widehat{\pi_n})$ satisfy constraints (II) – (VII). Since the verification procedure returned **No**, therefore, constraint (I) is not satisfied for at least one pair of cross-domain paths, say $\widehat{\pi_i}$ and $\widehat{\pi_j}$. This implies that $\widehat{\pi_i}$ and $\widehat{\pi_j}$ do not satisfy the cross-domain dependencies among the *component service*s of PW$_i$ and PW$_j$. Therefore, the path combination $(\widehat{\pi_1}, \widehat{\pi_2}, \ldots., \widehat{\pi_n})$ cannot support the distributed workflow. □

VITA

VITA

Basit Shafiq was born in Pakistan in 1975. He completed his Bachelors of Science in Electronics Engineering from Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, Topi, Pakistan, in 1999. He joined Purdue University in the fall of 1999 and obtained an M. S. in Electrical and Computer Engineering in December 2001. His research interests include information system security, distributed database systems, multimedia systems, and networking.