X-GTRBAC Admin: A Decentralized Administration Model for Enterprise-Wide Access Control

RAFAE BHATTI, BASIT SHAFIQ, ELISA BERTINO, and ARIF GHAFOOR
Purdue University
and
JAMES B. D. JOSHI
University of Pittsburgh

The modern enterprise spans several functional units or administrative domains with diverse authorization requirements. Access control policies in an enterprise environment typically express these requirements as authorization constraints. While desirable for access control, constraints can lead to conflicts in the overall policy in a multidomain environment. The administration problem for enterprise-wide access control, therefore, not only includes authorization management for users and resources within a single domain but also conflict resolution among heterogeneous access control policies of multiple domains to allow secure interoperation within the enterprise. This work presents design and implementation of X-GTRBAC Admin, an administration model that aims at enabling administration of role-based access control (RBAC) policies in the presence of constraints with support for conflict resolution in a multidomain environment. A key feature of the model is that it allows decentralization of policy administration tasks through the abstraction of administrative domains, which not only simplifies authorization management, but is also fundamental to the concept of decentralized conflict resolution presented. The paper also illustrates the applicability of the outlined administrative concepts in a realistic enterprise environment using an implementation prototype that facilitates policy administration in large enterprises.

Categories and Subject Descriptors: D.2.1 [**Software Engineering**]: Requirements/Specifications; D.4.6 [**Operating Systems**]: Security and Protection—*Access controls*; H.2.0 [**Database Management**]: General—*Security, integrity, and protection*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Algorithms, Management, Security.

Additional Key Words and Phrases: XML, role-based access control, secure interoperation, policy administration

The work reported in this paper has been partially supported by the National Science Foundation under grants no. IIS-0242419, IIS-0209111, and ITR-0428554, and by the sponsors of CERIAS.
Authors' addresses: Rafae Bhatti, Basit Shafiq, Elisa Bertino, and Arif Ghafoor; Department of Electrical and Computer Engineering and CERIAS, Purdue University, IN 47906; James B. D. Joshi, School of Information Sciences, University of Pittsburgh, PA 15260.

1. INTRODUCTION

Modern day enterprise spans several functional units or administrative domains. It is faced with the challenge of achieving efficient resource utilization to maintain a competitive edge and, simultaneously, ensuring secure interoperation across its constituent domains. The enterprise, thus, represents a dynamic environment where resource access policies include diverse authorization requirements expressed as authorization constraints. While desirable for access control, constraints can lead to conflicts in the overall policy in a multidomain environment. The administration of enterprise-wide access control, therefore, poses several challenges that range from authorization management of users and resources within individual domains to conflict resolution among heterogeneous access control policies of multiple domains to allow secure interoperation within the enterprise. Both of these issues are key aspects of the administration problem addressed by the work presented in this paper. Our work is based on the Role-Based Access Control (RBAC) model which is widely recognized as being helpful in simplifying authorization management in large enterprises [Sandhu et al. 1996]. The contributions of this paper are twofold: it presents (1) a formal specification of administrative concepts and constraints to facilitate the administration of advanced RBAC policies, and (2) a decentralized conflict resolution algorithm to allow secure interoperation in a multidomain environment.

Figure 1 illustrates a logical view of the policy administration aspects. An enterprise RBAC policy would typically consist of a set of users, roles, permissions, and the user-to-role and permission-to-role assignments. Figure 1a shows the RBAC policy administration tasks within a single domain. RBAC allows specification of constraints on policy administration tasks. The basic RBAC model includes separation of duty (SoD) and role-hierarchy related constraints, whereas advanced models (see Section 2) also allow specification of temporal and nontemporal contextual constraints. These constraints are essential to capture the access control requirements of the enterprise. Figure 1b shows the second aspect of policy administration concerning policy integration for interoperation in a multidomain environment. Policy integration, however, may introduce potential policy conflicts due to the presence of constraints. Administration of a multidomain policy, therefore, needs to be done in a manner that ensures the security of interoperation.

Although policy administration approaches for the basic RBAC model have been proposed (see Section 1.1), we believe that the basic RBAC model is not expressive enough to capture a variety of constraints needed to be enforced in an enterprise environment. The unique challenges motivating the use of more expressive policies for enterprise-wide access control have been highlighted in Bhatti et al. [2005] and Joshi et al. [2004], and an XML-based Generalized Temporal Role-Based Access Control (X-GTRBAC) framework has been proposed to address them. The X-GTRBAC specification language is based on Generalized Temporal Role-Based Access Control (GTRBAC) model [Joshi et al. 2005], which is a generalized temporal extension of the Role Based Access Control (RBAC) model proposed in the NIST RBAC standard [Ferraiolo et al. 2001]. X-GTRBAC

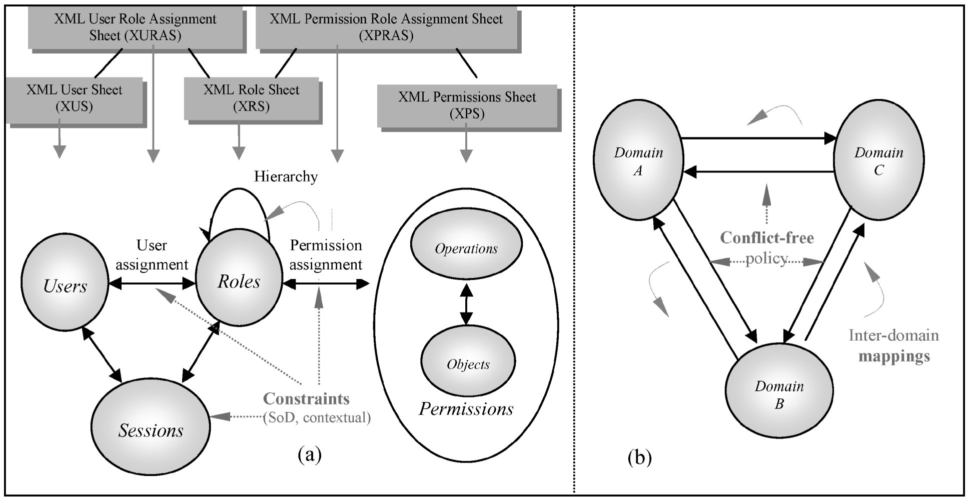


Fig. 1. Logical aspects of policy administration in a multidomain enterprise environment.

augments GTRBAC with XML to allow for supporting the policy enforcement in a heterogeneous, distributed environment.

While the X-GTRBAC framework is expressive enough for enterprise-wide access control (see Section 2.2), it lacks an administration model for authorization management, which may pose several challenges, since the task of managing a huge number of users and resources across the multiple administrative domains within the enterprise cannot realistically be centralized in a small team of security administrators. Hence, decentralizing the details of the access control scheme without losing central control over broad policy is a challenging goal [Sandhu and Munawer 1999]. Moreover, policy integration in a multidomain environment requires a conflict resolution strategy that can evaluate and resolve potential conflicts to allow secure interoperation. To enable solution to both these administration problems, we introduce in this paper X-GTRBAC Admin, the administration model for the X-GTRBAC framework. The primary focus of this paper is to elucidate these administrative concepts in the context of X-GTRBAC and outline the specifications of the proposed administration model.

The remainder of this paper is organized as follows. We begin with a comparison of our work with the earlier approaches for RBAC policy administration. We then discuss the background and motivation of our particular approach and outline the salient features of the X-GTRBAC specification language. We next present formal specifications of X-GTRBAC Admin, the administrative model for the X-GTRBAC framework for enterprise-wide access control. The discussion is divided into two sections. We first present the basic model addressing the policy administration tasks within a single domain and, thereafter, present the extended model in support of policy integration and secure interoperation in a multidomain environment. We consolidate the ideas presented in the paper with the discussion of a generic enterprise example and illustrate the applicability of our model using an implementation prototype that facilitates policy administration in large enterprises. To the best of our knowledge, no earlier work on policy administration has provided a comprehensive treatment of the two aspects of policy administration discussed in this paper. The paper concludes with a discussion on multidomain administration issues and a sketch of future research goals.

1.1 RELATED WORK

There has been a growing interest in administration models built on RBAC and related schemes. An administration model for RBAC (ARBAC99) has been proposed in Sandhu and Munawer [1999]. The model uses RBAC for role administration within an RBAC system and introduces the notion of an administrator role, with administrative permissions. It uses *can_assign* and *can_assignp* relations for role and permission assignments, respectively. These relations use the notions of (1) the "role range" that an administrator role has authority over, and (2) the "prerequisite role" (also called prerequisite condition) needed to exercise that authority. Both the *role range* and *prerequisite role* are derived from the role hierarchy. Certain weaknesses in the model have been highlighted in Oh and Sandhu [2002]. An ARBAC02 model has been presented in Oh and Sandhu [2002] to overcome these weaknesses and it uses the organization structure as the basis for prerequisite conditions, instead of prerequisite roles in a role hierarchy. While the ARBAC02 model is adequate for the tasks of role and permission assignments within an enterprise, there are certain issues left unaddressed. The original ARBAC97 model that ARBAC02 builds upon does not explicitly support specification of constraints during policy administration, as it assumes that "constraints will be enforced while carrying out administrative chores." The only kind of constraint explicitly supported by the model is a role-membership constraint. This leaves much to be desired, since an administration model should be able to express a variety of constraints and also include the management of constraints in the policy administration process. We provide formal specification of an extensive set of constraints, including those on role enabling and activation, and a set of administrative functions, relations, and operations that facilitates the process of policy administration in the presence of constraints. In addition, ARBAC02 does not address the issues related to policy administration in a multidomain environment. It does not formally define the notion of organizational structure, while we do so based on the semantics of role and domain hierarchies. This formalization is then used to address issues of secure interoperation and conflict resolution in a multidomain policy.

A scoped administration model for RBAC has been proposed in Crampton and Loizou [2002]. The model uses the notion of administrative scope to define administrative operations on role hierarchy. The primary focus of Crampton and Loizou [2002] is to observe and rectify any side effects of changes to a role hierarchy in a single domain. We maintain that we are tackling a related problem, but within a multidomain environment. We assume that the single-domain policies are initially consistent, with stable role hierarchies, and consider the effect of role assignments and interdomain role mappings on the consistency of these policies. The issues of role hierarchy management within a single domain are relevant to administration, but orthogonal to current focus of this paper. Like ARBAC02, the model in Crampton and Loizou [2002] also does not deal with management of constraints nor does it deal with issues of policy administration in a multidomain environment.

2. BACKGROUND AND MOTIVATION

In this section, we provide some background and motivation needed to discuss the administrative concepts related to the X-GTRBAC framework.

2.1 RBAC and GTRBAC

In order to discuss the salient features of the X-GTRBAC framework, and its administrative extension, we provide the formal definitions of the component models of our framework, namely, RBAC and GTRBAC.

Definition 2.1.1 **(RBAC Model)** [Ferraiolo et al. 2001] The RBAC model consists of the following components:

- Sets `Users`, `Roles`, `Permissions`, and `Sessions` representing the set of users, roles, permissions, and sessions, respectively;
- UA $\subseteq$ `Users` $\times$ `Roles`, the user assignment relation, that assigns users to roles;
- *assigned_users:* `Roles` $\rightarrow 2^{\texttt{Users}}$, the mapping of role r onto a set of users. Formally: $assigned\_users(r) = \{u | (u, r) \in \texttt{UA}\}$;
- `PA` $\subseteq$ `Roles` $\times$ `Permissions`, the permission assignment relation, that assigns permissions to roles;
- *assigned_permissions*: `Roles` $\rightarrow 2^{\texttt{Permissions}}$, the mapping of role r onto a set of permissions. Formally: $assigned\_permissions(r) = \{p | (p, r) \in PA\}$;
- `Sessions` $\subseteq$ `Users` $\times 2^{Roles}$;
- *user:* `Sessions` $\rightarrow$ `Users`, which maps each session to a single user;
- *role:* `Sessions` $\rightarrow 2^{Roles}$ that maps each session to a set of roles;
- `RH` $\subseteq$ `Roles` $\times$ `Roles`, a partially ordered role hierarchy (written $\geq$).
  A session $s_i \in$ `Sessions` has the permission of all roles r' junior to roles activated in the session, i.e.,
  $\{p | ((p, r) \in PA \ \vee (p, r') \in PA) \wedge r \in \text{role}(s_i) \wedge \text{r} \geq r' \}$

The *RH* relation is one of the most important aspects of RBAC for its use toward simplifying authorization management. 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 [Sandhu 1998]. An *inheritance-activation hierarchy* can be defined on roles by composing *inheritance* and *activation* hierarchies [Joshi et al. 2002].

In this paper, we do not concern ourselves with the advanced semantics of role hierarchies and use the $\geq$ relation defined in the RBAC standard.

The GTRBAC model [Joshi et al. 2005] incorporates a set of language constructs for the specification of various temporal constraints on roles, including constraints on role enabling, role activation, user-to-role assignments, and permission-to-role assignments. In particular, GTRBAC makes a clear distinction between role *enabling* and role *activation*. An *enabled* role indicates that a user can activate it, whereas an *activated* role indicates that at least one subject has activated a role in a session. The notion of separate activation conditions is particularly helpful in large enterprises, with several hundred users belonging to the same role, to selectively manage role activations at the individual user level.

The temporal framework in GTRBAC model allows the specification of the following constraints, events, and expressions:

1. *Temporal constraints on role enabling/disabling*: These constraints allow one to specify the time intervals during which a role is enabled. It is also possible to specify a role duration.
2. *Temporal constraints on user-to-role and permission-to-role assignments*: These are constructs to express either a specific interval or a duration in which a user or a permission is assigned to a role.
3. *Activation constraints*: These allow one to specify how a user should be restricted in activating 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.
4. *Run-time events*: A set of run-time events allows an administrator to dynamically initiate GTRBAC events or a user to issue activation requests.
5. *Constraint enabling expressions*: GTRBAC includes events that enable or disable duration constraints and role-activation constraints.
6. *Triggers*: Triggers allow one to express dependency among GTRBAC events as well as capture the past events and define future events on which they are based.
7. *Periodic time expression*: A periodic expression (`PTE`) is represented by pairs <`[begin,end]`,`P`>, where `P` is a periodic expression denoting an infinite set of periodic time instants, and `[begin, end]` is a time interval `I` denoting the lower and upper bounds that are imposed on instants in `P`. Formally, P is expressed as follows:

Definition 2.1.2 (**Periodic Expression**): [Joshi et al. 2005]: Given calendars $Cd, C1, \ldots, Cn$, and time occurrences $O_1, \ldots, O_n$, a periodic expression P is defined as:

$$\mathtt{P} = \sum_{i=1}^{n} O_i.C_i \triangleright x.C_d$$

where $O1 = \mathtt{all}$, $Oi \in 2^N \cup \{\mathtt{all}\}$, $Ci \sqsubseteq Ci\text{-}1$ for i = 2, .., n, $Cd = Cn$, and $x \in \mathbb{N}$.

Table I. Temporal Constraints and Event Expressions in GTRBAC

Constraint categories	Events	Expression
Enabling constraints	Role enabling	(I, P,D, `enable/disable` r)
Activation constraints	Role activation	<!–only occurs as a run-time event –>
Assignment constraint	User-to-role assignment	([I, P, D], $\texttt{assign}_{\text{U}}/\texttt{deassign}_{\text{U}} r$ to u)
	Permission-to-role assignment	([I, P, D], $\texttt{assign}_{\text{P}}/\texttt{deassign}_{\text{P}} p$ to r)
Trigger	<!–any triggering event –>	$E_1, \ldots, E_n$, $C_1, \ldots, C_k \rightarrow E$ `after` Δt
Run-time requests	Users' activation request	(s:(de)`activate` r `for` u `after` Δt))
	Administrator's run-time request	($\texttt{assign}_{\text{U}}/\texttt{de-assign}_{\text{U}} r$ `to` u `after` Δt)
		(`enable/disable` r `after` Δt)
		($\texttt{assign}_{\text{P}}/\texttt{de-assign}_{\text{P}} p$ `to` r `after` Δt)
		(`enable/disable` c `after` Δt)

The formalism for periodic expressions is based on the notion of calendars. A calendar is defined as a countable set of contiguous intervals, numbered by integers called indexes of the intervals. Symbol $\triangleright$ separates the first part of the periodic expression that identifies the set of starting points of the intervals it represents, from the specification of the duration D of each interval in terms of calendar *Cd*. For example, `all`. *Years* + {3, 7}. *Months* $\triangleright$ 2. *Months* represents the set of intervals starting at the same instant as the third and seventh month of every year and having a duration of 2 months. In practice, *Oi* is omitted when its value is `all`, whereas it is represented by its unique element when it is a singleton. $x.Cd$ is omitted when it is equal to $1.Cn$.

The temporal constraint expressions in GTRBAC are summarized in Table I.

2.2 X-GTRBAC

X-GTRBAC allows specification of all the elements of the GTRBAC model. These specifications are captured through a context-free grammar called X-Grammar, which follows the same notion of terminals and nonterminals as in Backus-Naur Form (BNF), but supports the tagging notation of XML that also allows expressing attributes within element tags. The use of attributes helps maintain compatibility with XML schema syntax, which serves as the type definition model for our language. Since it follows BNF convention, X-Grammar can be accepted by a well-defined automaton to allow automatic translation into XML schema documents. These XML documents ultimately constitute the enterprise access control policy base. Following is a brief description of the X-Grammar specifications that result in the policy base.

The X-Grammar for *user*, *role, and permission* called XML User Sheet (XUS), XML Role Sheet (XRS), and XML Permission Sheet (XPS), respectively, encode the respective elements of the RBAC model. In addition, they specify a list of attributes of a user, role, or permission, which may be relevant for the purposes of user-role and permission-role assignments. These assignments are captured, respectively, by XML User-to-Role Assignment Sheet (XURAS) and XML Permission-to-Role Assignment Sheet (XPRAS). A salient feature of X-GTRBAC is that it captures the temporal constraint expressions of the GTR-BAC model using X-Grammar specification. These expressions are defined in an XML Temporal Constraint Definition (XTempConstDef). The temporal constraints on user-to-role and permission-to-role assignments are then included, respectively, in the XURAS and XPRAS. Similarly, enabling and activation constraints may also be defined and included with the role definition in XRS.

In addition to temporal constraints, X-GTRBAC also supports nontemporal contextual constraints. For instance, an assignment constraint in XURAS may also involve the evaluation of the user attributes (as declared in XUS) to decide the eligibility of users for role assignments. Similarly, an enabling or activation constraint in XRS may involve the evaluation of the role attributes (as declared in XRS) to decide on the enabling or activation of a role. The evaluation of such contextual constraints allows a fine-grained mechanism for role assignment, enabling, or activation. Another kind of constraint that can be specified on roles is an SoD constraint. SoD constraints are defined using an XML separation-of-duty definition (XSoDDef) and included in XRS. The use of X-Grammar specification in policy administration process has been illustrated in Figure 1a. The X-Grammar for XUS, XRS, XPS, XURAS, and XPRAS is presented in Appendix A. The detailed specifications of X-GTRBAC framework can be found in [Bhatti et al. 2005].

We now introduce X-GTRBAC Admin, the administrative extensions to the GTRBAC model, and present the formal specification for its components.

3. X-GTRBAC ADMIN—BASIC MODEL

X-GTRBAC Admin is introduced to address the two aspects of the policy administration problem discussed in the paper. In this section, we present the basic concepts of the model designed primarily to address the policy administration tasks within a single domain (Figure 1a). The model is extended in Section 4 to support secure interoperation in a multidomain environment (Figure 1b).

In order to include the administrative concept in the X-GTRBAC framework, the specification language is extended to include the specification of an administrative domain (Admin Domain), an administrative role (Admin Role), and an administrative permission (Admin Permission).

Admin Domain is the most defining feature of the X-GTRBAC Admin model. It represents an administrative domain of authority within an enterprise. In X-GTRBAC Admin, all instances of regular and administrative roles and permissions are associated with an Admin Domain. The Admin Domains are related according to a partial order and this partially ordered set defines an administrative domain hierarchy. This hierarchy reflects the organizational structure of the enterprise. Each Admin Domain is assigned an Admin Role and an Admin Role may have authority over multiple Admin Domains by virtue of dominance relationship among domains as defined in the formal model.

An Admin Role is an administrative role authorized to manage policy administration tasks within a given Admin Domain. This authority is given by a set of associated Admin Permissions as defined in the model. A pool of administrative users is selected by the SSO for assignment to Admin Roles, where such assignment may be based on evaluation of user attributes and applicable constraints, as in the case of regular roles, to allow fine-grained policy assignments. The Admin Roles are related according to a partial order, which defines an administrative role hierarchy. As we will see later (Section 4.4), this hierarchy is induced by dominance relationship amongst domains.

An Admin Permission specifies an administrative permission that can be used by an Admin Role. Typically a set of available permissions belonging to various Admin Domains within the enterprise would be created by the respective SSOs. We use *assign*, *deassign*, *assignp*, *deassignp*, *enable*, *disable*, *map*, and *unmap* as the basic set of Admin Permissions. These permissions, however, are only a qualification and not an authorization. We define a set of authorization relations in the formal model that must also be satisfied by the Admin Role to carry out an administrative operation.

3.1 Formal Model

Based on these concepts, the formal definition of the X-GTRBAC Admin model is now presented.

Definition 3.1.1 **(Core Components):** The X-GTRBAC Admin model consists of the following core components:

- $\mathtt{AD}$, a set of Admin Domains;
- $\mathtt{AU}$, a set of administrative users, $\mathtt{AU} \subseteq \mathtt{Users}$;
- $\mathtt{RR}$, a set of regular roles, $\mathtt{RR} = \mathtt{Roles}$;
- $\mathtt{ER}$, a set of enabled regular roles, $\mathtt{ER} \subseteq \mathtt{RR}$;
- $\mathtt{RP}$, a set of regular permissions, $\mathtt{RP} = \mathtt{Permissions}$;
- $\mathtt{AR}$, a set of Admin Roles;
- $\mathtt{AP}$, a set of Admin Permissions;
- The association of regular roles with domains, called a *role instances*,[1] is defined as $\mathtt{RR}_\mathrm{D} \subseteq \mathtt{AD} \times \mathtt{RR} = \{(ad, rr) \mid ad \in \mathtt{AD}, rr \in \mathtt{RR}\}$;
- The association of regular permissions with domains, called *permission instances*, is defined as $\mathtt{RP}_\mathrm{D} \subseteq \mathtt{AD} \times \mathtt{RP} = \{(ad, rp) \mid ad \in \mathtt{AD}, rp \in \mathtt{RP}\}$;
- The association of an Admin Domain with an Admin Role is defined as $\mathtt{AR}_\mathrm{D} \subseteq \mathtt{AD} \times \mathtt{AR} = \{(ad, ar) \mid (ad \in \mathtt{AD}, ar \in \mathtt{AR})\ \}$;
- The association of Admin Permissions with Admin Domains is defined as $\mathtt{AP}_\mathrm{D} \subseteq \mathtt{AD} \times \mathtt{AP} = \{(ad, ap) \mid ad \in \mathtt{AD}, ap \in \mathtt{AP}\}$
- The set $\mathtt{ATTR}_x$ of attribute-value pairs for a regular user, role, or permission instance x; an attribute value pair $\mathtt{AVP}$ is defined as a tuple (*name, value*), where both *name* and *value* are constants.
- The set $\mathtt{SOD}$ of regular roles in Separation of Duty (SoD); a collection $\mathtt{SODS}$ of $\mathtt{SOD}$ sets may exist to define fine-grained SoD constraints.

[1]The terms role instance and permission instance are used to differentiate the instances of (possibly the same) role or permission in different domains. Therefore, the model allows the instances in different domains to share the same definition.

- The set CR of constraints defined according to Definition 3.1.2. CR may be an empty set.
- The set CONST is a collection of constants of simple data types (as string, integer, etc.).
- $\text{AUA} \subseteq \text{AU} \times \text{AR}$, the user-assignment relation, that assigns administrative users to Admin Roles;
- $\text{APA} \subseteq \text{AP} \times \text{AR}$, the permission-assignment relation, that assigns Admin Permissions to Admin Roles;
- $\text{RM} \subseteq \text{RR} \times \text{RR}$, the role-mapping relation, that maps a regular role from one domain to a regular role from another domain;
- A partially ordered regular role hierarchy $\text{RRH} \subseteq \text{RR} \times \text{RR}$; $\text{RRH} = \text{RH}$;
- A partially ordered Admin Role hierarchy, $\text{ARH} \subseteq \text{AR} \times \text{AR}$;
- A partially ordered Admin Domain hierarchy $\text{DH} \subseteq \text{AD} \times \text{AD}$; $(ad_x \geq ad_y) \in \text{DH}$ implies that domain ad_x dominates domain ad_y in the hierarchy. □

A significant feature of our model is the use of constraints directly in the policy administration process. The following definition formalizes the notion of a constraint expression used in our framework.

Definition 3.1.2 **(Constraints):** A constraint expression in X-GTRBAC Admin is one of the following two kinds:

- A periodic time expression PTE defined as per the GTRBAC model (see Section 2.1).
- A logical expression using the usual $\wedge$ and $\vee$ operators on three tuples of the form $(y, \omega, \delta(p1, \ldots, pn))$, where δ is a parameterized administrative function (as defined in Table II), p_i and y are a member of the set $(\text{RR} \cup \text{AR} \cup \text{RP} \cup \text{AP} \cup \text{AD} \cup \text{Users} \cup \text{CONST})$, and $\omega \in \{=, \neq, \geq, \leq, \in\}\}$

An administrative function *evaluate* is a predicate defined to validate a constraint expression, i.e., *evaluate:* $\text{CR} \rightarrow \{\texttt{true}, \texttt{false}\}$. A constraint evaluates to true in one of the following two ways: (1) It is a PTE and the associated interval and periodicity conditions are satisfied, or (2) it is a logical expression with clauses of the form $(y, \omega, \delta(p1, \ldots, pn))$, and the expression is satisfied over the set of clauses. A clause evaluates to true if y compares with the return value of the function δ according to the comparison operator ω. □

3.2 Administrative Features

We now present the salient features of X-GTRBAC Admin based on the formal model. The set of features supported by the model include administrative functions (used for review), administrative relations (used to determine authorizations), administrative operations (used to change the system state), and administrative restrictions (used to restrict membership in certain components of the model). Overall, the set of identified administrative features provide comprehensive coverage for all management tasks that are required to administer a multidomain RBAC system.

Table II. X-GTRBAC Admin Functions

Function	Description	Formal semantics
domain:(RR ∪ AR ∪ AP) → AD	Returns the domain of a role or permission instance	*domain(x)* = $\{d \mid (d, x) \in \mathrm{RR_D} \vee (d, x) \in \mathrm{AR_D} \vee (d, x) \in \mathrm{AP_D}\}$
has_authority_over: AR → 2^{AD}	Returns the set of all Admin Domains that an Admin Role has authority over. An Admin Role has authority over a domain if the latter is same as or dominated by the domain of the former	*has_authoriy_over(ar)* = $\{d \mid \mathrm{domain}(ar) = d \vee \mathrm{domain}(ar) \geq d\}$
administers: AR → 2^{RR}	Returns the set of all regular roles administered by an Admin Role. An Admin Role administers a regular role if both belong to the same domain	*administers(ar)* = $\{rr \mid \mathrm{domain}(ar) = \mathrm{domain}(rr)\}$
has_attribute_value: (Users ∪ RR ∪ RP) × CONST → CONST	For the user, role, or permission x, returns the value of the attribute identified by v	*has_attribute_value(x,v)* = $\{\mathtt{AVP.value} \mid \mathtt{AVP.name} = \mathtt{v} \wedge \mathtt{AVP} \in \mathtt{ATTR}_x\}$
in_separation_of_duty: RR → SODS	For the role *rr*, returns the SOD set	*in_separation_of_duty(rr)* = $\{\mathtt{SOD} \mid rr \in \mathtt{SOD}\}$
assigned_users: RR → 2^{Users}	For the role *rr*, returns the set of users assigned to the role	*assigned_users(rr* ∈ RR) = $\{u \mid (u, rr) \in \mathtt{UA}\}$[a]
assigned_permissions: RR → $2^{\mathrm{Permissions}}$	For the role *rr*, returns the set of permissions assigned to the role	*assigned_permissions(rr)* = $\{rp \mid (rr, rp) \in \mathtt{PA}\}$[a]
activated_users: RR → 2^{Users}	For the role *rr*, returns the set of users active in the role	*activated_users(rr)* = $\{u \mid rr \in \mathtt{ER} \wedge (\exists s_i \in \mathtt{Sessions} \mid rr \in \mathrm{role}(s_i) \wedge u \in \mathrm{user}(s_i))\}$[b]

[a]This function is modified to use qualified (i.e. regular) role instance. This definition supersedes that in Definition 2.1.1.
[b]Note that the roles that may have been activated belong to the enabled regular role set, i.e. only enabled roles can be activated.

Table II summarizes the administrative functions provided by X-GTRBAC Admin. These functions can be used by the SSO or the administrators to obtain information about different components of the system, such as association of a role or permission instance with a domain, authority of an Admin Role over Admin Domains, association of an Admin Role with a regular role, values of role attributes, roles in separation of duty, users, and permissions assigned to roles, and users who have activated certain roles. This information may be used as the basis of predicates in constraint expressions to restrict any administrative operation. An example of using review functions in constraints will be provided in Section 3.3.

The administrative relations provided by X-GTRBAC Admin are given in Table III. These relations are used to represent association between various components of the model, such as set of administrative users and Admin Roles, Admin Permissions and Admin Roles, and Admin Roles and regular roles. The relations involve the evaluation of a (possibly null) constraint expression that is used to determine the validity of the association for a particular administrative

Table III. X-GTRBAC Admin Relations

Relation	Description	Formal Semantics
can_assign_admin $\subseteq$ $\mathtt{AU} \times \mathtt{AR_D} \times \mathtt{CR}$	The user-assignment relation that authorizes an SSO to assign administrative users to Admin Roles subject to the satisfaction of the associated constraints	$\mathit{can\_assign\_admin}\ (au, ar, c) \leftrightarrow (c = \mathtt{null} \vee \mathit{evaluate(c)} = \mathtt{true})$
can_deassign_admin $\subseteq \mathtt{AU} \times \mathtt{AR_D} \times \mathtt{CR}$	analogous to *can_assign_admin*	
can_assignp_admin $\subseteq \mathtt{AR_D} \times \mathtt{AP_D} \times \mathtt{CR}$	The permission-assignment relation, that authorizes an SSO to assign Admin Permissions to Admin Roles if the permission and the role belong to the same domain, and subject to the satisfaction of the associated constraints	$\mathit{can\_assignp\_admin}\ (ar, ap, c) \leftrightarrow \mathit{domain(ar)} = \mathit{domain(ap)} \wedge (c = \text{null} \vee \mathit{evaluate(c)} = \text{true})$
can_deassignp_admin $\subseteq \mathtt{AR_D} \times \mathtt{AP_D} \times \mathtt{CR}$	analogous to *can_assignp_admin*	
can_assign $\subseteq \mathtt{AR_D} \times \mathtt{RR_D} \times \mathtt{CR}$	The user-assignment relation, that authorizes an Admin Role (or its senior) to assign a regular user to a regular role (or its junior) if the Admin Role has the *assign* Admin Permission, and subject to the satisfaction of the associated constraints	$\mathit{can\_assign(ar,rr,c)} \leftrightarrow (rr \in \mathit{administers(ar)} \wedge \mathit{(ar,assign)} \in \mathtt{APA} \wedge \mathit{evaluate(c)} = \text{true}) \vee (\exists\, ar' \in \mathtt{AR} \mid ar \geq ar' \wedge \mathit{can\_assign}(ar', rr, c)) \vee (\exists\, rr' \in \mathtt{RR} \mid rr' \geq rr \wedge \mathit{can\_assign}(ar, rr', c))$
can_deassign $\subseteq \mathtt{AR_D} \times \mathtt{RR_D} \times \mathtt{CR}$	analogous to *can_assign*	
can_enable $\subseteq \mathtt{AR_D} \times \mathtt{RR_D} \times \mathtt{CR}$	The role-enabling relation, that authorizes an Admin Role (or its senior) to enable a regular role (or its junior) if the Admin Role has the *enable* Admin Permission, and subject to the satisfaction of the associated constraints	$\mathit{can\_enable}(ar, rr, c) \leftrightarrow (rr \in \mathit{administers(ar)} \wedge \mathit{(ar,enable)} \in \mathtt{APA} \wedge \mathit{evaluate(c)} = \text{true}) \vee (\exists ar' \in \mathtt{AR} \mid ar \geq ar' \wedge \mathit{can\_enable}(ar', rr, c)) \vee (\exists\, rr' \in \mathtt{RR} \mid rr' \geq rr \wedge \mathit{can\_enable}(ar, rr', c))$
can_disable	analogous to *can_enable*	
can_assignp $\subseteq \mathtt{AR_D} \times \mathtt{RR_D} \times \mathtt{CR}$	The permission-assignment relation, that authorizes an Admin Role (or its senior) to assign a regular permission to a regular role (or its junior) if the Admin Role has the *assignp* Admin Permission, and subject to the satisfaction of the associated constraints	$\mathit{can\_assignp}(ar, rr, c) \leftrightarrow (rr \in \mathit{administers(ar)} \wedge \mathit{(ar,assignp)} \in \mathtt{APA} \wedge \mathit{evaluate(c)} = \text{true}) \vee (\exists\, ar' \in \mathtt{AR} \mid ar \geq ar' \wedge \mathit{can\_assignp}(ar', rr, c)) \vee (\exists\, rr' \in \mathtt{RR} \mid rr' \geq rr \wedge \mathit{can\_assignp}(ar, rr', c))$
can_deassignp	analogous to *can_assignp*	
can_map $\subseteq \mathtt{AR_D} \times \mathtt{RR_D} \times \mathtt{RR_D} \times \mathtt{CR}$	The role-mapping relation, that allows an Admin Role to map a regular role from one domain to a regular role from another domain if the Admin Role has the *map* Admin Permission, and subject to the satisfaction of the associated constraints	$\mathit{can\_map}(ar, r1, r2, c) \leftrightarrow (r1, r2 \in \mathit{administers(ar)} \wedge \mathit{domain(r1)} \neq \mathit{domain(r2)} \wedge \mathit{(ar,map)} \in \mathtt{APA} \wedge \mathit{evaluate(c)} = \text{true}) \vee (\exists\, ar' \in \mathtt{AR} \mid ar \geq ar' \wedge \mathit{can\_map}(ar', r1, r2, c)) \vee (\exists r1', r2' \in \mathtt{RR} \mid r1' \geq r1 \vee r2' \geq r2 \wedge \mathit{can\_map}(ar, r1', r2', c))$
can_unmap	analogous to *can_map*	

Table IV. X-GTRBAC Admin Operations[c]

Operation	Description	Formal semantics
assign_admin_role(au $\in$ AU, *ar* $\in$ AR, *c* $\in$ CR)	The user-assignment operation that adds a new tuple to the AUA relation	if *can_assign_admin*(au, ar, c) then $\mathtt{AUA} = \mathtt{AUA} \cup (au, ar)$
assign_admin_permission(ar $\in$ AR, *ap* $\in$ AP, *c* $\in$ CR)	The permission-assignment operation that adds a new tuple to the APA relation	if *can_assign_adminp*(ar, ap, c) then APA $= \mathtt{APA} \cup (ar, ap)$
assign_role(ar $\in$ AR, *rr* $\in$ RR, *u* $\in$ Users, *c* $\in$ CR)	The user-assignment operation that adds a new tuple to the UA relation	if *can_assign*(ar, rr, c) then $\mathtt{UA} = \mathtt{UA} \cup (u, rr)$
enable_role(ar $\in$ AR, *rr* $\in$ RR, *c* $\in$ CR)	The role-enabling operation that adds a new element to the ER set	if *can_enable*(ar, rr, c) then $\mathtt{ER} = \mathtt{ER} \cup (rr)$
assign_permission(ar $\in$ AR, *rr* $\in$ RR, *rp* $\in$ RP, *c* $\in$ CR)	The permission-assignment operation that adds a new tuple to the PA relation	if *can_assignp*(ar, rr, c) then $\mathtt{PA} = \mathtt{PA} \cup (rr, rp)$
map_role(ar $\in$ AR, *r*1 $\in$ RR, *r*2 $\in$ RR, *c* $\in$ CR)	The role-mapping operation that adds a new tuple to the RM relation	if *can_map*(ar, $r1$, $r2$, c) then $\mathtt{RM} = \mathtt{RM} \cup (r1, r2)$

[c]The reverse operations (such as *deassign_role*) are obtained by using the corresponding predicate (such as *can_deassign*) from Table III and replacing $\cup$ with -.

Table V. X-GTRBAC Admin Restrictions

Restriction	Description	Formal Semantics
RR-AR-Mutual Exclusion	No role can be included in both the regular and the administrative role sets	$\mathrm{RR} \cap \mathrm{AR} = \emptyset$
AR-AD Uniqueness	An Admin Role is associated uniquely with an Admin Domain, i.e., an Admin Domain cannot be assigned more than one Admin Role, and an Admin Role cannot be assigned to more than one Admin Domain	$\forall\, ar, ar', ad, ad' \in \mathtt{AR_D}, ar \neq ar', ad \neq ad', (ad, ar) \in \mathtt{AR_D} => (ad, ar') \notin \mathtt{AR_D} \wedge (ad', ar) \notin \mathtt{AR_D}$
AUA Role cardinality	The cardinality of Admin Roles associated with administrative users through the AUA relation is 1, i.e., only a single user can be assigned to an Admin Role at any given time	$\forall u, u' \in$ Users, $u \neq u'$, $(u, ar) \in \mathtt{AUA} => (u', ar) \notin \mathtt{AUA}$
AUA User cardinality	The cardinality of administrative users associated with Admin Roles through the AUA relation is 1, i.e., an administrative user can be assigned to only one Admin Role at any given time	$\forall\, ar, ar' \in \mathtt{AR}, ar \neq ar'$, $(u, ar) \in \mathtt{AUA} => (u, ar') \notin \mathtt{AUA}$

operation, such as role assignment, enabling, or interdomain role mapping. Note that role activation is not included as an administrative relation since it is not performed by an administrator, but is done at run time by users.

Table IV summarizes the administrative operations provided by X-GTRBAC Admin. These include operations for role assignment, enabling, and interdomain role mapping.

The set of administrative restrictions used in X-GTRBAC Admin are given in Table V. These restrictions are not a fundamental component of the model, and have the status of recommendations. The use of these restrictions is not only intuitive, but also keeps the administration concept simple in practice.

3.3 Administration Process

To better illustrate the administration model and the use of various administrative features, we outline the policy administration process in XGTRBAC-Admin.

3.3.1 *Creation of Core Component Sets.* The first step in the administration process is the one-time creation of the core component sets of the model by the SSO as defined in Definition 3.1.1. A key concept here is the creation of set of role (permission) instances, which allows different instances of the same role (permission) in different domains to adhere to the same role (permission) definition. This allows enforcement of uniform administration policies across the enterprise. Another significant feature is the introduction of partial order on the administrative role and administrative domain sets which captures the hierarchical organizational structure in the enterprise. We recommend the use of membership constraints defined in Table V on creation of $\mathtt{RR}$, $\mathtt{AD}$, and $\mathtt{AR}_\mathtt{D}$ sets. These constraints imply that the association of Admin Roles and Admin Domains should be unique and any authority over multiple Admin Domains can be transferred to an Admin Role only through semantics of the administrative role and domain hierarchies.

3.3.2 *Assignment of Administrative Users to Admin Roles, and That of Admin Permissions to Admin Roles.* This is done by the SSO using the *assign_admin_role* and *assign_admin_permission* operations defined in Table IV. We recommend the use of cardinality constraints defined in Table V on these assignments. These constraints are natural to impose given the distinction between administrative roles and regular roles; administrative tasks would typically not require multiple users to be assigned to them, and vice versa.

3.3.3 *Assignment of Regular Users to Regular Roles, and That of Regular Permissions to Regular Roles.* This is done by the Admin Roles using the *assign_role* and *assign_permission* operations defined in Table IV. An example of policy administration process involving the use of these operations is presented in Figure 2; an administration process involving other administrative operations will be similar.

The example begins by creation of the core components and the constraints used in policy administration. This step is performed by the SSO. After this step, the Admin Roles and Admin Permissions in the system have been designated through the AUA and APA relations. In the next step, *assign_role* and *assign_permission* operations are requested, involving assignment of user u2 and permission $\mathtt{p2}_\mathtt{a}$ to role $\mathtt{r1}_\mathtt{a}$ by ARa subject to the constraints $\mathtt{c}_1$ and $\mathtt{c}_2$, respectively. The precondition of these operations involves evaluation of the authorization relations and associated constraints, and the postcondition (after a successful evaluation) is the addition of new tuples in the UA and PA relations.

Let

$AD = \{a, b\};$

$Users = \{u1, u2, u3, u4, u5\};\ AU = \{u1, u2\};$

$AR = \{ARa, ARb\};\ AR_D = \{(a, ARa), (b, ARb)\}$

$AUA = \{(u1, ARa), (u2, ARb)\};$

$APA = \{(ARa, \mathit{assign}), (ARa, \mathit{assignp})\}$

$RR = \{r1, r2, r3\};\ ATTR_{r1} = \{(priority, low)\}$

$RP = \{p1, p2, p3, p4, p5\};\ ATTR_{p2} = \{(propagate, true)\}$

$RR_D = \{(a, r1_a), (a, r2_a), (b, r1_b)\};$

$RP_D = \{(a, p1_a), (a, p2_a), (b, p1_b)\}$

$UA = \{(u1, r1_a), (u2, r1_b)\};$

$PA = \{(r1_a, p1_a), (r1_b, p1_b)\}$

$CR = \{c_1 = (u1, \in, (\mathit{assigned\_users}(r1_a)) \wedge (high, \neq, \mathit{has\_attribute\_value}(r1_a, priority))),$

$c_2 = (p1_a, \in, (\mathit{assigned\_permissions}(r1_a)) \wedge (true, =, \mathit{has\_attribute\_value}(p1_a, propagate)))\}$

Do

O1: $\mathit{assign\_role}(ARa, r1_a, u2, c_1)$

O2: $\mathit{assign\_permission}(ARa, r1_a, p2_a, c_2)$

Pre-condition:O1: $r1_a \in \mathit{administers}(ARa) \wedge (ARa, \mathit{assign}) \in APA \wedge \mathit{evaluate}(c_1) \leftrightarrow \mathit{can\_assign}(ARa, r1_a, c_1)$

O2: $r1_a \in \mathit{administers}(ARa) \wedge (ARa, \mathit{assignp}) \in APA \wedge \mathit{evaluate}(c_2) \leftrightarrow \mathit{can\_assignp}(ARa, r1_a, c_2)$

Post-condition:O1: $UA = UA \cup (u2, r1_a)$

O2: $PA = PA \cup (r1_a, p1_a)$

Fig. 2. An example of policy administration process involving *assign_role* and *assign_permission* operations.

3.3.4 *Administration of Multidomain Policies.* It may be noted that by virtue of the hierarchical ordering of domains and roles, the administrative roles may have authority over multiple (collaborating) domains. We call the administrators acquiring these roles as the *multidomain administrators*. These administrators can use the *map_role* operation defined in Table IV to establish interdomain mapping between roles belonging to multiple domains, and thereby allowing interoperation between their respective policies. Multidomain administration will be handled in the extended administration model and is the topic of next section.

4. EXTENDED ADMIN MODEL FOR SECURE INTEROPERATION

The decentralization in the X-GTRBAC Admin model achieved by delegating authority enables the local domain administrators to define access control policies within their administrative domains. As has been pointed out in Section 1, the other aspect concerning administration of enterprise-wide access control policies is policy integration to allow interoperation among these heterogeneous domains. This requires an access policy that governs information access beyond the individual domains' boundaries. Such a policy is defined by the multidomain administrators. A multidomain administrator may specify both *permitted* and *restricted* interdomain accesses by using *map_role* and *unmap_role* operations defined in Table IV. These operations let the multidomain administrators establish a role mapping among the collaborating domains (as illustrated in Figure 3). However, because of the presence of constraints, authorizations specified by the multidomain administrators may conflict with the underlying access-control policies of constituent domains. These conflicts need to be resolved in a manner such that the authorizations and restrictions defined by the administrators of the constituent domains are not preempted, while also ensuring security of overall interoperation.

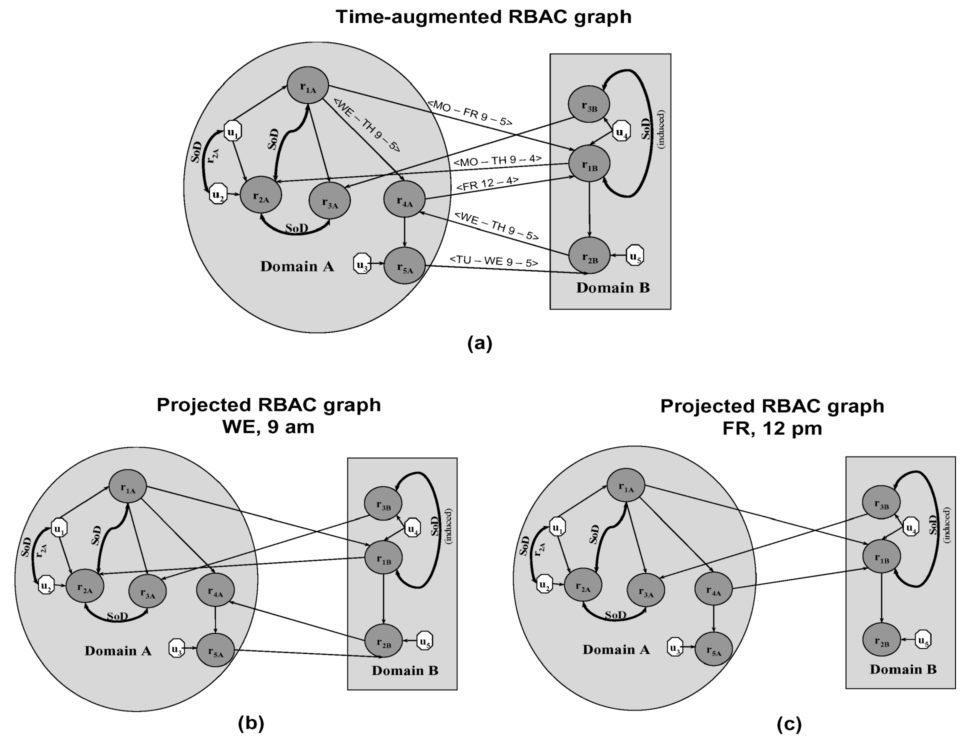


Fig. 3. Multidomain interoperation graphs illustrating role mapping and associated constraints.

In our earlier work [Shafiq et al. 2005], we proposed a policy integration framework for composing an interoperation policy from the RBAC policies of collaborating domains, where the individual domain policies are assumed to be consistent, i.e., conflict-free. In that framework, an integer programming (IP)-based approach was used to determine an optimal solution to policy conflicts. In this paper, we extend the existing policy integration framework to address the following key issues: (1) Secure interoperation of X-GTRBAC policies in the presence of constraints; and (2) decentralized conflict resolution for a multidomain policy.

Because the multidomain policy is configured to manage interoperation, the enablement/disablement of contextual constraints may change both local and remote authorizations, resulting in change in policy state. In addition, multidomain administrators are authorized to add/remove interdomain role mappings within their respective domains, resulting in change in policy configuration. To ensure secure interoperation, the policy conflicts need to be resolved whenever such events occur.

To apply the policy integration framework proposed in [Shafiq et al. 2005], we adopt a graph-based formalism for specifying advanced RBAC policies. In the following, we first introduce the graph-based formalism for specifying the GTRBAC policies and then discuss the proposed policy integration mechanism for secure interoperation.

4.1 Graph-Based Specification Model for GTRBAC

In graph-based models, 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 arbitrarily connected. An edge between a user node u and a role node r indicates that role r is assigned to user u. Edges between role nodes are used to model the role hierarchy within a domain, where we use the role hierarchy as defined in the RBAC model. In addition, interdomain edges may exist between role nodes and are used to capture hierarchical relationship between interdomain roles. A user u can access a role r if there is a path from the user node u to the role node r that consist only of user-role assignment and role-role hierarchy edges. For instance, in Figure 3a user u_1 assigned to role $r_{1\mathrm{A}}$ and $r_{2\mathrm{A}}$ can access all roles except $r_{3\mathrm{B}}$ because of the existence of access paths from the user node u_1 to those role nodes. There can be edges between role and permission nodes (not in figure). The graph model supports specification of SoD constraints, which are particularly relevant in an enterprise. A role-specific SoD constraint disallows assignment and/or activation of conflicting roles to the same user. Similarly, a user-specific SoD constraint prohibits conflicting users from accessing (being assigned to or activating) the same role simultaneously. In the graph, a role-specific SoD constraint between two roles is represented by a double arrow between the corresponding roles. To represent a user-specific SoD constraint between 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 access r_k simultaneously. In the RBAC graph of Figure 3a, role-specific SoD constraints are defined between roles $r_{2\mathrm{A}}$ and $r_{1\mathrm{A},}$ and $r_{2\mathrm{A}}$ and $r_{3\mathrm{A}}$. A user-specific SoD constraint for role $r_{2\mathrm{A}}$ is also specified between users u_1 and u_2. In addition, an SoD constraint is induced between roles $r_{1\mathrm{B}}$ and $r_{3\mathrm{B}}$ because of the interdomain mappings specified by the multidomain administrator. This is due to the fact that (1) roles $r_{2\mathrm{A}}$ and $r_{3\mathrm{A}}$ are in role-specific SoD, (2) role $r_{1\mathrm{B}}$ is mapped to role $r_{2\mathrm{A}}$, and (3) role $r_{3\mathrm{B}}$ is mapped to role $r_{3\mathrm{A}}$. Because of (2) and (3), user u_4 can simultaneously acquire permissions of roles $r_{2\mathrm{A}}$ and $r_{3\mathrm{A}}$ by accessing roles $r_{3\mathrm{B}}$ and $r_{1\mathrm{B}}$ and thereby violate SoD constraint in (1). Therefore, roles $r_{3\mathrm{B}}$ and $r_{1\mathrm{B}}$ must also be in SoD.

The temporal semantics of GTRBAC is incorporated by augmenting the RBAC graph with temporal labels. The RBAC graph of Figure 3a is a time-augmented policy graph comprising two domains A and B. The temporal labels associated with the edges indicate that the access allowed by the edge is temporally constrained within the stated time period. Unlabeled edges indicate accesses that are not temporally constrained. An augmented RBAC graph is first projected into a simple RBAC graph by retaining all the accesses that are allowed at a given time instance. The projected RBAC graph is then used for conflict resolution. For instance, the projections of the augmented RBAC graph at two different time instances are shown in Figures 3b and c. Although an expensive process, we argue that the projection technique is adequate for the temporal framework provided by X-GTRBAC Admin for enterprise-wide access control. This is because the access requirements in enterprise systems, although dynamic, are relatively stable for a given period of time (say during the execution of a project, etc.). In addition, even if the access requirements to different enterprise resources vary between different work shifts, the duration of the shifts is reasonably large to justify the overhead of switching between shifts and improving production (or service) efficiency. Therefore, we believe that very few projections (proportional to the number of work shifts per day in an enterprise—a typical value is 2–3) will be needed. Essentially, the feasibility of this technique is determined by the application domain, and has been articulated in our case. We will return to the example later on in Section 5 to concretely discuss the interoperation strategy for the illustrated RBAC graph of Figure 3. However, before that, we will introduce the security requirements for such an interoperation.

4.2 Security Requirements in a Multidomain System

The goal of policy integration is to allow information and resource sharing without preempting the local authorizations of constituent domains and simultaneously ensuring security of the overall interoperation. As mentioned earlier, interdomain role mappings defined by multidomain administrators may conflict with the authorizations defined in the local access control policies of constituent domains. These conflicts need to be resolved without altering domains' local policies. In particular, the following two principles need to be enforced while establishing secure interoperation [Gong and Qian 1996].

• *Autonomy principle*: If an access is permitted within an individual domain, it must also be permitted under secure interoperation**.** In the context of RBAC, the autonomy principle entails that all the valid role accesses by local users specified in a domain's local access control policy must be supported in the overall interoperation policy. Formally:

$reachable\ (u, r, G_i) \Rightarrow reachable\ (u, r, G)$

where, G_i is an RBAC graph representing the local access control policy of domain i and G is the multidomain RBAC graph corresponding to the overall interoperation policy. The predicate *reachable* (u, r, G) is stated in Table VI; it returns true if the role node r in the RBAC graph G can be reached from the user node u, implying that user u can access role r.

In the RBAC framework discussed above, the violation of autonomy principle may occur because of interdomain constraints introduced by interdomain role mappings, such as induced SoD, which was illustrated in the preceding discussion with reference to Figure 3.

• *Security principle*: If an access is not permitted within an individual domain, it must also not be permitted under secure interoperation. In the context of RBAC, all role accesses prohibited in a domain's local access control policy must not be supported in the overall interoperation policy. Formally:

$\neg\ reachable\ (u, r, G_i) \Rightarrow \neg\ reachable\ (u, r, G)$

The violations of the security principle can be classified into following three classes:

1. *Role assignment violation*: An interoperation policy causes a violation of role assignment constraint of domain k if it allows a user u of domain k to access

Table VI. Predicates and Functions Used in Policy Analysis

Predicate/function	Description
reachable(*u*,*r*,*G*)	Returns True if the role node r in the RBAC graph G can be reached from the user node u. This reachability implies that user u can access role r under the policy specified as RBAC graph G
uassign(*u*,*r*)	Returns True if user u is assigned to role r
passign(*p*,*r*)	Returns True if permission p is assigned to role r
exists(*x*)	Returns True if x ∈ AD
dom(*x*,*y*)	Returns True if y ∈ *dominates*(*x*)
overrideable(*G*)	Returns True if the configuration in graph G is overrideable; false for leaf nodes
dominates(*x*)	Returns set LD of domains locally dominated by domain x. Note that $\|LD\| = 2$
projected-RBAC(*TG*)	Returns projected RBAC graph G corresponding to the temporal RBAC graph TG
IP-formulate(*G*)	Returns an IP problem P for the policy graph G formulated with the desired optimality criterion using the constraint transformation rules listed in Table B.1
graph-prune(*G*,*P*)	Returns G s.t. $G = G - \{(r_i, r_j) \| \exists u_k (u_{k_{r_i}}, = 1$ and $u_{k_{r_j}}, = 0)$ and $domain(r_1) \neq domain(r_j)\}$ i.e., from the graph G, remove all the interdomain edges (r_i, r_j) for which there exists a user u_k, such that the variables $u_{k_{r_i}}, = 1$ and $u_{k_{r_j}}, = 0$ in P

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.

2. *Role-specific SoD violation*: An interoperation 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.
3. *User-specific SoD violation*: Let U_r^c denote the conflicting set of users for role r belonging to domain k. An interoperation 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.

The interoperation policy graphs shown in Figure 3 lead to all three types of policy conflicts described above. For instance, the policy graph in Figure 3b defines the following interoperation between domains A and B:

1. Role r_{1A} in domain A inherits all the permissions available to r_{1B} in domain B by virtue of the interdomain mapping from r_{1A} to r_{1B}.
2. r_{5A} in domain A inherits all the permissions available to r_{2B} in domain B because of the interdomain mapping from r_{5A} to r_{2B}.
3. r_{1B} in domain B inherits all the permissions available to r_{2A} in domain A because of the interdomain mapping from r_{1B} to r_{2A}.
4. r_{2B} in domain B inherits all the permissions available to r_{4A} in domain A because of the interdomain mapping from r_{2B} to r_{4A}.

This interoperation policy is not conflict free. It allows role r_{5A} to access the permissions of its senior role r_{4A} through r_{2B}, which is a violation of *role-assignment constraint*. Moreover, the interdomain role mappings from r_{1A} to

r_{1B} and from r_{1B} to r_{2A} enable u_1 to also access the conflicting role r_{2A} when it accesses role r_{1A}. This is a violation of the *role-specific SoD* constraint between roles r_{1A} and r_{2A}. In addition, the multidomain policy allows u_1 to access the role r_{1A} and u_2 to access the role r_{2A}. u_1 by accessing r_{1A} can acquire permission over r_{2A} through the role r_{1B}. This is a violation of *user-specific SoD constraint*, which prohibits u_1 and u_2 from simultaneously accessing the role r_{2A}.

Conflicts in a multidomain policy occur because of the interdomain role accesses that are defined through *map_role* operation. Such conflicts can be resolved by either modifying the security policy of the affected domains or by restricting interdomain accesses conflicting with the access control policy of any of the collaborating domain. The former is not a viable solution as it affects the autonomy of domains, which is a key requirement for secure interoperation. The latter seeks resolution of conflicts by removing interdomain mappings without triggering any change in the domains' local access control policies, implying that the autonomy of all interoperating domains remain intact. With respect to the latter approach, there may be several choices available for resolving interoperation conflicts and each choice corresponds to removal of a different set of interdomain mappings. An arbitrary selection of interdomain edges for removal may significantly reduce interoperation.

Alternatively, conflict resolution can be formulated as an optimization problem to maximize interdomain information and resource sharing according to some prespecified optimality measure. Various optimality measures for secure interoperation have been defined in the literature [Gong and Qian 1996]. These include maximum direct access, maximum sharing, and minimum representation. Selection of an appropriate optimality measure depends on the underlying applications or processes requiring interdomain interoperation. For instance, maximizing interdomain data accessibility is a meaningful optimization criterion for distributed database systems carrying out transactions that are largely independent of each other. However, this optimality criterion may not be suitable for distributed task-based applications involving a number of tasks, each requiring a set of resource accesses in order to be executed. In this case, attempting to maximize individual interdomain accesses without regard to their semantic dependence with respect to the tasks being executed will not yield a viable solution. Therefore, an appropriate optimality criterion is to maximize the number of overall tasks supported by the collaborative system rather than maximizing individual interdomain accesses.

As an example consider the interoperation policy graph shown in Figure 3b. We define two tasks t_1 and t_2. Suppose the execution semantics of task t_1 require the invoking process/user to access roles r_{3B}, r_{1A}, and r_{4A}. Similarly, t_2 entails accessing roles r_{3B}, r_{1A}, and r_{5A} by the process/user invoking the task. These execution semantics impose an additional requirement on the interoperation between domains A and B and affect the choice of resolution strategy for the conflicts in the interoperation policy graph depicted in Figure 3b. To resolve the policy conflicts enumerated earlier, one of the following sets of edges from the multidomain policy graph of Figure 3b needs to be removed: $\{(r_{1A}\text{-}r_{1B}), (r_{5A}\text{-}r_{2B})\}$, $\{(r_{1A}\text{-}r_{1B}), (r_{2B}\text{-}r_{4A})\}$, $\{(r_{1B}\text{-}r_{2A}), (r_{5A}\text{-}r_{2B})\}$, $\{(r_{1B}\text{-}r_{2A}), (r_{2B}\text{-}r_{4A})\}$. As we will see later, removal of the interdomain role edges $(r_{1B}\text{-}r_{2A})$, $(r_{5A}\text{-}r_{2B})$ yields an optimal resolution with respect to maximum resource accesses; however, none of the tasks t_1 and t_2 can be supported in the resulting interoperation policy. In case the optimality criterion is to maximize task execution, the interdomain edge (r_{1B}-r_{2A}) needs to be retained in the final policy, which, in this case, is contrary to the objective of maximizing individual interdomain accesses. In the remainder of this section, we focus on the formulation of conflict resolution as an IP problem and the discussion of our conflict resolution algorithm. Subsequently in Section 5, we provide an example demonstrating the applicability of our conflict resolution strategy to the inconsistent interoperation policy depicted in Figure 3b.

4.3 IP Formulation of RBAC Policy

We use integer programming (IP)-based approach for resolving policy conflicts in an optimal manner. In the proposed approach, the projected RBAC policy is formulated as the following 0-1 integer program:

$$\begin{aligned}&\text{maximize}\, c^T u_r + d^T t\\&\text{subject to}\, A u_r \leq b\\&\forall t_k \in t,\ t_k - \prod_{u_{ir_j} \in S_{t_k}} u_{ir_j} = 0\\&\forall u_{ir_j} \in u_r,\ u_{ir_j} = 0 \text{ or } 1\\&\forall t_k \in t,\ t_k = 0 \text{ or } 1\end{aligned}$$

Here, c and d are the cost vectors defining the optimality criterion, u_r is a vector representing user role authorization, and t is a vector of tasks. The elements of matrix A correspond to the coefficients of the terms used in the IP constraints. The various variables in the system are used to formulate the constraints and optimality criterion for the IP problem. We now discuss these separately.

4.3.1 *Constraints.* In the IP formulation of RBAC policy, we express constraints such as role assignment, SoD, and permitted/restricted access using equations/inequalities. The variables used in these constraints 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 cannot access role r_j. If user u_i and role r_j are from different domains and $u_{ir_j} = 0$, then there should not be any path from the user node u_i to the role node r_j in the role graph. Note that a given multidomain RBAC policy may be inconsistent and a path may exist between user u_i from one domain and role r_j from another domain, while it may be found that $u_{ir_j} = 0$ in the solution to the IP problem. Based on this solution, the inconsistency is resolved in the final policy by dropping the interdomain edge(s) that lie(s) in the path between the user node u_i and role node r_j.

The elements of the vector t correspond to different task variables. Each task $t_k \in t$ requires certain user-role accesses to be invoked in order to be executed, i.e., a task t_k comprises of a set of u_{ir_j} variables. The set s_{t_k} defines the dependence of task t_k on these variables. The task variable t_k gets assigned a value of 1 when all the associated u_{ir_j} variables are also 1, which is expressed in the second constraint in this IP problem. Semantically, this constraint means that an interoperation policy supports execution of task t_k if all the user-role accesses in the set s_{t_k} are supported. The last two constraints mean that the variables t_k and u_{ir_j} are binary variables. Appendix B tabulates the transformation rules for generating IP constraints from the projected multidomain RBAC graph.

4.3.2 *Optimality Criterion.* We have discussed two optimality criteria in the preceding subsection. The given IP problem can be formulated to support both those criteria based on the selection of variables c and d. In particular, the first criterion regarding maximizing interdomain data accessibility can be specified in the objective function as a sum of all decision variables representing interdomain user to role authorization, i.e., all c_i s corresponding to interdomain user-role variables are assigned a value of "1" and the remaining c_i s and all d_i s are set to "0" [Shafiq et al. 2005]. As mentioned in Section 4.1, the temporal constraint associated with a interdomain role mapping restricts the time during which interdomain information can be accessed via such mapping. Enabling or disabling of interdomain role mappings may introduce conflicts in the interoperation policy. Resolution of such conflicts requires reevaluation of the IP, which may change the prior level of sharing. In order to avoid frequent invocation of the IP, it is desired to achieve maximum sharing for the longest period of time. This requires removing the conflicting interdomain accesses with smaller time spans in favor of the interdomain accesses with larger time spans. This requirement can be specified in the IP objective function by assigning weights to the decision variables in proportion to the remaining time span of the corresponding interdomain role accesses.

The second criterion regarding maximizing the number of collaborative tasks supported by the system can be specified in the objective function by including all task variables associated with the collaboration tasks with uniform weights, i.e., all d_i s are assigned a value of "1" and all c_i s are set to "0".

A correctness result for the IP formulation technique for conflict resolution has been presented in [Shafiq et al. 2005], which establishes the fact that after solving the IP problem and applying the solution to the policy graph, none of the identified conflicts remain in the interoperation policy.

4.4 Conflict Resolution Algorithm

Based on the discussion in the preceding subsections, we now give the conflict resolution algorithm to generate a conflict-free policy for an RBAC policy graph. The RBAC policy graph that we consider in this paper is actually the projection of an augmented graph representing a GTRBAC policy (see Figure 3). It is, therefore, the case that different projections of the GTRBAC policy graph may be obtained at different time instances. This implies that conflict resolution is no more only a static operation; it needs to be performed every time the state of the policy (i.e., the projection) changes. The conflict resolution algorithm has,

Table VII. Decentralized Conflict Resolution Algorithm and Its Subroutines

```
DCR($TG_k, \ldots, TG_n$)
 1.  for i ← k to n
 2.      $G_{i:i} \leftarrow$ projected-RBAC($TG_i$)
 3.  $G_{k-1:k} \leftarrow G_{k:k}$
 4.  $M_{k-1:k} \leftarrow \emptyset$
 5.  for i ← k to n−1
 6.      $\{G_{i:i+1}, M_{i:i+1}\} \leftarrow$ CR2($G_{i-1:i}, G_{i,+1:i+1}$)
 7.  while not (I ← detect_change($TG_1, \ldots,$
     $TG_n, M_{0,1}, \ldots, M_{n-1,n}$))
 8.  ;                                        // busy wait
 9.  m ← min(I)            // minimum leaf node index
10.  k ← m−1
11.  go to step 5
```

```
CR2($G_{a:x}, G_{y:y}$)
 1.  if not exists(x:y)
 2.    Create x:y s.t. dom(x:y,a:x) $\wedge$ dom(x:y,y:y)
 3.    Create $AR_{x:y}$ for x:y s.t. x:y $\in$ domain($AR_{x:y}$).
 4.  if overrideable($G_{a:x}$)
 5.      D ← dominated_domains(a:x)
 6.  else D ← a:x
 7.  $G_{p:q} = D_1$          // $D_i$ refers to ith element in D
 8.  for i ← 1 to |D|−1
 9.      $\{G_{q:q+1}, M_{q:q+1}\} \leftarrow$ CR2($G_{q-1:q}, G_{q+1:q+1}$)
10.      q = q+1
11.  P ← IP-formulate($G_{q-1:q}$)
12.  $G_{q-1:q} \leftarrow$ graph-prune($G_{q-1:q}$,P)
13.  return $G_{q-1:q}, M_{q-1:q}$
```

```
dominated_domains(x:y)
 1.  D ← Ø
 2.  LD = a:x, y:y ← dominates(x:y)  // child nodes
 3.  if exists(a:x)
 4.    if domiantes($D_1$) ≠ Ø
                       // if left child is composite node
 5.      if overidable(Ga:x)
 6.          D ← D ∪ dominated_domains(a:x)
 7.    else D ← D ∪ a:x
 8.  if exists(y:y)
 9.      D ← D ∪ y:y
10.  return D
```

```
detect_change($TG_1, \ldots, TG_n, M_{0,1}, \ldots, M_{n-1,n,}$)
 1.  for i ← 1 to n
 2.      I[i] ← 0              // initialize change vector
 3.  for i ← 1 to n
 4.    for each temporal constraint tc $\in TG_i$
 5.    if ( (enabled(tc,time) and not
       enabled(tc,time-1)) or (not enabled(tc,time)
       and enabled(tc,time-1)) )
 6.                I[i] ← 1
 7.  for i ← 1 to n
 8.    for each temporal constraint tc $\in M_{i-1,i}$
 9.    if ( (enabled(tc,time) and not
       enabled(tc,time-1)) or (not enabled(tc,time)
       and enabled(tc,time-1)) )
10.                I[i] ← 1
11.  return I
```

therefore, been modified from [Shafiq et al. 2005] and significantly extended to incorporate this dynamic aspect.

The main routine Decentralized Conflict Resolution (DCR) and its subroutines are shown in Table VII. The algorithm makes use of several predicates and functions, listed in Table VI. Of particular importance is the notion of dominance among domains introduced on the basis of the organizational structure of the enterprise. In order to represent the multiple domains within the enterprise, we construct a domain tree, which essentially represents an administrative domain hierarchy as defined in Definition 3.1.1. Let each administrative domain be a leaf in a tree, where the tree represents the organizational structure. We use a construction method that starts from the leaves of the tree and moves upward. We use a pair-wise merging method to successively merge child nodes into composite parent nodes. The methodology is as follows.

At the each level, a new leaf is combined with another composite node (except for the first step, when both nodes are leaf nodes). Effectively, there is an Admin Role hierarchy corresponding to the domain hierarchy, which is comprised of the Admin Roles from each domain. The administrator at each leaf in the domain tree is responsible for the domain represented by the leaf. Then, by virtue of the *has_authority_over* function, defined in Table II, the administrator at the next higher level is responsible for the domains represented by its child nodes. Henceforth, we call the immediate child nodes as *locally* dominated domains

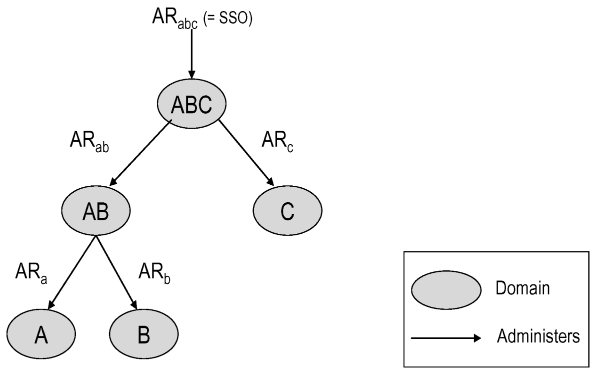


Fig. 4. An organizational domain free.

and the immediate parent as the dominating domain. The construction proceeds in this manner and eventually reaches the root of the tree, which is responsible for the overall multidomain policy of the enterprise (this is typically the SSO).

An example domain tree is shown in Figure 4. It also indicates the Admin Role associated with each domain that has authority over that domain. The leaves A, B, and C represent the individual enterprise domains, administered, respectively, by AR_a, AR_b, and AR_c. The node AB represents the union (in terms of administrative authority) of A and B and is administered by AR_{ab}. Here, AB is the dominating domain and A and B are the locally dominated domains. In this example, the most dominating domain is ABC (root of the tree) and is administered by the SSO. Its *locally* dominated domains are AB, and C, whereas A, B, and C comprise its set of *all* dominated domains. It may be noted that the set of *all* dominated domains always comprises of leaf nodes. On the other hand, the set of locally dominated domains could contain domains, such as AB, which are not leaf nodes and, hence, not an individual enterprise domain, but only represent a union of administrative authority over multiple domains. This distinction is important in our conflict resolution algorithm, and also has impact on the optimality vs. autonomy tradeoff, as shall be shortly discussed.

We note that while the decentralization of policy administration tasks proceeds from the root to the leaves, the conflict resolution tasks take the reverse route. This is necessary to maintain consistency in the overall enterprise policy, while simultaneously preserving the autonomy of the individual domains to the extent possible. The conflict resolution algorithm is, therefore, designed to preserve the autonomy and also allow a tradeoff between autonomy and optimality, if desired. This is achieved as follows. Each multidomain administrator (such as AR_ab in Figure 4) carries out localized conflict resolution within its dominated domains (A and B) and creates a dominating domain (AB) that represents the secure interoperation environment for its locally dominated domains (A and B) according to the resulting conflict-free policy. The node representing the dominating domain (node AB) may, however, itself be a locally dominated domain of another higher level node (ABC) administered by another multidomain administrator (AR_abc). Hence, the conflict resolution is recursively carried out until we reach the root of the tree (ABC).

At each step during the process, the administrator of the dominating domain has the option of keeping or modifying the conflict-free policy obtained from its locally dominated domains. If the policy is kept, it is simply used as it is in the conflict resolution algorithm. If the policy is, however, needed to be modified, the policies of *all* (as opposed to only *locally*) dominated domains would again need to be included in the analysis. This decision could be motivated by optimality considerations, as keeping the policy might lead to a suboptimal overall solution. Consider for instance nodes A, B, and AB in Figure 4. AB represents the secure interoperation environment for A and B according to a conflict-free policy. At the next stage, it is possible that the policy obtained by applying conflict resolution algorithm on AB and C be less optimal than the policy obtained by applying the algorithm directly on A, B, and C. Keeping in mind that this autonomy vs. optimality tradeoff depends on the target enterprise, we propose a hybrid scheme that allows the system to be configured in either mode (i.e., keeping or modifying the input policy). Building upon this discussion of the domain tree, we now summarize the conflict resolution algorithm.

The input to the DCR is the set of temporal graphs (TGs), which represent GTRBAC policies of the individual enterprise domains enumerated in a numerical order. These graphs are first projected onto corresponding RBAC policy graphs (Gs), and then submitted two at a time to the pair-wise conflict resolution CR_2 subroutine. CR_2 creates a composite domain by merging the domains that the incoming graphs correspond to (which therefore become the locally dominated domains of the newly created domain). The dominating domain then carries out the conflict resolution on either the locally dominated domains or all dominated domains, depending on whether the incoming graph is overrideable, i.e., can be modified. Due to the construction method, only the left child could be a composite node, so the incoming graph supplied as the first argument (assumed to correspond to the left child) needs to be checked. The subroutine dominated_domains helps to get the set of all dominated domains, if applicable. CR_2 returns the conflict-free policy graph and the interdomain role mapping for the locally dominated domains. When the control returns to `DCR`, it busy-waits on `detect_change` to detect any change in policy state or configuration in *temporal* (and not projected) graphs of individual domains. The change-detection routine records the indexes of all individual domains where any change occurred. The minimum of these indexes is obtained and DCR is reinvoked on the domain tree upward from that index. The use of minimum index eliminates redundant computation as the complete domain tree need not be changed every time. We note that all composite domains are indexed as a two-digit pair separated by a colon, indicating the domain indexes making up the composite domain. In the case of an individual domain, both digits are the same. The use of two-digit indexes enables carrying out the analysis on the domain tree during the execution of the algorithm. In practice, the numeric subscript could be replaced by a letter, as in Figure 4.

Having completed the discussion on our administration model, we next present an example of a generic enterprise that demonstrates how the features of basic and advanced X-GTRBAC Admin model would be useful for enterprise-wide access control.

Table VIII. A Set of Regular Users

#	Domain	User Id	Eligible role (ER)
1.	domA	u1	R1A
2.	domA	u2	R2A
3.	domA	u3	R3A
4	domB	u4	R1B
5.	domB	u5	R2B

Table IX. A Set of Regular Permissions

#	Domain	Perm Id	Eligible role (ER)
1.	domA	P1	R1A
2.	domA	P2	R2A
3.	domA	P3	R3A
4	domB	P4	R1B
5.	domB	P5	R2B

Table X. A Set of Admin Roles

#	Admin role (AR)	Valid intervals	AR domain
1.	ARa	MO-FR 9-5	domA
2.	ARb	MO-FR 9-5	domB
3.	ARab	MO-FR 9-5	domA, domB
4	ARsp	TEMP	SPECIAL

Table XI. A Set of Admin Permissions

#	Admin permission (AP)	AP domain
1.	AP1 (can_assign,can_assignp)	domA
2.	AP2 (can_assign,can_assignp)	domB
3.	AP3 (can_deassign)	domB
4	AP4 (can_enable)	ALL

5. ENTERPRISE-WIDE ACCESS CONTROL AND X-GTRBAC ADMIN

The administrative concepts presented in X-GTRBAC Admin are now illustrated in the context of a generic enterprise environment. For continuation with preceding discussion, and making it concrete, we use the example presented in Figure 3. Let the users and permissions within the two domains A and B be given in Tables VIII and IX, respectively. Tables X and XI give the candidate users for the Admin Roles and the set of available Admin Permissions, respectively, for the two domains. We note that domain A and domain B are administered by Admin Roles ARa and ARb, respectively. Note, also, that the Admin Role ARab has authority over both domains. The intention here is that while ARa and ARb can carry out policy administration tasks within their own domains, ARab is responsible for administering the interoperation policy between the two domains. We will discuss the general case for scoping the administrative authority in Section 6, where we present a mechanism for implementing our decentralized administration framework. We next observe the administrative features provided by X-GTRABC Admin to administer the enterprise access control policy. The forthcoming discussion of the administration process involving this example uses the XML policy files described in Section 2.2. We remind the reader that the syntax of these files has been provided in Appendix-A.

5.1 Assignment of Administrative Roles and Permissions

The assignment of Admin Roles and Admin Permissions is carried out by the SSO. An Admin Role is represented in our framework using the XRS, which includes information, such as hierarchy and cardinality constraints. It also includes attributes of the role, including associated domains. The assignment of Admin Roles `ARa`, `ARb`, `ARab`, and `ARsp` is handled by the *can_assign_admin* relation, which is represented using XURAS. The assignment may be based on the evaluation of applicable constraints on user attributes which are encoded in a specially designated admin credential. The user credential and associated cardinality constraints are specified using the XUS (cardinality in XUS is represented by `MaxRoles` element). An Admin Permission is represented in our framework using the XPS, which includes attributes of a permission, including associated domains. The assignment of Admin Permissions `AP1`–`AP4` to the Admin Roles may based on the attributes of the role as defined in XRS and is handled by the *can_assignp_admin* relation, which is represented using XPRAS.

For the purpose of this example, we do not explicitly need to indicate the users assigned as administrators and would just use the Admin Roles by name in subsequent discussions. It may be noted that the temporal conditions supplied in Table X restrict the activation of the Admin Roles by the assigned users to only within the stated validity period. Such conditions reflect the realistic scenario within an enterprise, where the activation of Admin Roles may need to be time-constrained. The clear distinction between role assignment and role activation in GTRBAC allows this constraint to be effectively enforced.

From the information in Tables X and XI, we note that `ARa` and `ARb` can be assigned `AP1`, and `AP2`, and `AP3`, respectively, whereas `ARab` can be assigned `AP1`, `AP2`, and `AP3`, because it has administrative authority over both domains to which these permissions belong. `AP4` can also be assigned to any Admin Role, because it is designated as available for ALL domains. On the other hand, the domain for AR4 has been designated as SPECIAL, which implies that it is an Admin Role that may be enabled temporarily during nonusual activity periods, such as special projects. In such cases, additional domains of administrative authority are typically needed according to the scale of the project. Hence AR4 can be configured to act as an Admin Role for the SPECIAL project domain(s) and would remain valid for the TEMP duration of the project. The corresponding Admin Permissions for these Admin Roles would be project-specific, and created by the SSO.

5.2 Assignment of Regular Roles and Permissions

The assignment of regular roles and regular permissions are carried out by Admin Roles similarly using XURAS and XPRAS, respectively. The XURAS represents the *can_assign* relation, whereas the XPRAS represents the *can_assignp* relation. The "eligible role" column in Table IX is used to indicate the roles that a user or permission could be assigned to after the satisfaction of the eligibility criterion, expressed as a set of constraints. The information needed to

Maximize $u_{1r_{1B}} + u_{1r_{2B}} + u_{3r_{2B}} + u_{4r_{2A}} + u_{4r_{3A}} + u_{4r_{4A}} + u_{4r_{5A}} + u_{5r_{4A}} + u_{5r_{5A}}$
Subject to
Constraints derived from Rules 1,2, 3 and 4
c0: $u_{1r_{1A}} = 1$, c1:$u_{1r_{4A}} = 1$, c2: $u_{1r_{5A}} = 1$, c3:$u_{2r_{1A}} = 0$, c4: $u_{2r_{2A}} = 1$, c5:$u_{2r_{3A}} = 0$,
c6: $u_{2r_{4A}} = 0$, c7:$u_{2r_{5A}} = 0$, c8: $u_{2r_{1B}} = 0$, c9: $u_{2r_{2B}} = 0$, c10:$u_{3r_{1A}} = 1$, c11: $u_{3r_{2A}} = 0$,
c12:$u_{3r_{3A}} = 0$, c13: $u_{3r_{4A}} = 0$, c14:$u_{3r_{5A}} = 1$, c15: $u_{3r_{1B}} = 0$, c16:$u_{4r_{1B}} = 1$, c17:$u_{4r_{3B}} = 1$,
c18: $u_{4r_{1A}} = 0$, c20: $u_{5r_{1B}} = 0$, c21:$u_{5r_{2B}} = 1$, c22:$u_{5r_{1A}} = 0$, c23:$u_{5r_{2A}} = 0$,
c24:$u_{5r_{3A}} = 0$, c44: $u_{5r_{3B}} = 0$, c45: $u_{1r_{3B}} = 0$, c46: $u_{2r_{3B}} = 0$. c47: $u_{3r_{3B}} = 0$
Constraints derived from Rule 5
c25: $u_{1r_{1A}} - u_{1r_{1B}} \geq 0$, c26:$u_{1r_{1B}} - u_{1r_{2B}} \leq 0$, c28:$u_{1r_{1B}} + u_{1r_{5A}} - u_{1r_{2B}} \geq 0$,
c29:$u_{3r_{1B}} + u_{3r_{5A}} - u_{3r_{2B}} \geq 0$, c30: $u_{4r_{1B}} - u_{4r_{2A}} \geq 0$, c31:$u_{1r_{1B}} - u_{1r_{2B}} \leq 0$,
c32:$u_{4r_{1A}} + u_{4r_{2B}} - u_{4r_{4A}} \geq 0$, c33:$u_{4r_{4A}} - u_{4r_{5A}} = 0$,
c34:$u_{5r_{1A}} + u_{5r_{2B}} - u_{5r_{4A}} \geq 0$, c35:$u_{5r_{4A}} - u_{5r_{5A}} = 0$,
Constraints derived from Rule 6
c36: $u_{3r_{5A}} - u_{3r_{2B}} - u_{1r_{5A}} + u_{1r_{2B}} - (1 - u_{3r_{2B}})u_{1r_{1B}} = 0$, c37:$u_{5r_{2B}} - u_{5r_{4A}} - u_{4r_{2B}} + u_{4r_{4A}} = 0$,
$c38$: $u_{5r_{2B}} - u_{5r_{4A}} - u_{3r_{2B}} + u_{3r_{4A}} \geq 0$, c39:$u_{5r_{2B}} - u_{5r_{4A}} - u_{1r_{2B}} + u_{1r_{4A}} \geq 0$
Constraints derived from Rule 7
c40:$u_{1r_{2A}} + u_{1r_{3A}} \leq 1$, c41:$u_{1r_{1A}} + u_{1r_{2A}} \leq 1$, c42:$u_{2r_{2A}} + u_{2r_{3A}} \leq 1$, c50:$u_{4r_{2A}} + u_{4r_{3A}} \leq 1$,
Constraints derived from Rule 8
c43:$u_{1r_{2A}} + u_{2r_{2A}} \leq 1$
Constraints derived from Rule 9
c48:$t_1 - u_{4r_{3B}} . u_{4r_{2A}} u_{4r_{4A}} = 0$, c49:$t_2 - u_{4r_{3B}} . u_{4r_{2A}} u_{4r_{5A}} = 0$

Fig. 5. IP corresponding to the RBAC graph of Figure 3b.

evaluate the constraints is obtained from the XUS, XRS, and XPS documents. The users assigned to Admin Roles can then execute the assigned Admin Permissions. For instance, the Admin Role ARa can assign the user u1 to role R1A, because it has the *can_assign* permission (AP1) and required scope (i.e., its domain A is same as the domain of R1A). ARb has *can_assign* permission (AP2) over domain B and can assign u4 to role R1B. Using *can_assignp* permission, the permissions P1, P2, and P3 will also be assigned to the roles R1A, R2A, and R3A by ARa, whereas P4 and P5 will be assigned to the roles R1B and R2B by ARb, respectively.

Any applicable constraints during the process of user-role and permission-to-role assignments for both regular and administrative roles are included in the XURAS and XPRAS documents as temporal or logical expressions. The use of XURAS and XPRAS in this manner facilitates automated fine-grained policy assignments in large enterprises.

5.3 Conflict Resolution

The IP formalism allows secure interoperation to be established while resolving conflicts that arise because of the dynamic resource accesses in the example multidomain policy. Applying the conflict resolution algorithm of Section 4 on the RBAC graph of Figure 3b gives the IP program shown in Figure 5. The optimality criterion of this IP is to maximize all interdomain accesses as can be observed from the objective function that consists of all interdomain user-role variables with uniform weight assignment. The IP is solved using CPLEX, a well known IP solver. The optimal solution to this IP problem has an objective function value of 7 with the following values of interdomain variables: $u_{1r_{1B}} = u_{1r_{2B}} = u_{4r_{3A}} = u_{4r_{4A}} = u_{4r_{5A}} = u_{5r_{4A}} = u_{5r_{5A}} = 1, u_{3r_{2B}} = u_{4r_{2A}} = 0$. Since, in the optimal solution,$u_{3r_{2B}} = 0$ and $u_{4r_{2A}} = 0$, the interdomain edges ($r_{1\mathrm{B}}$-$r_{2\mathrm{A}}$) and ($r_{5\mathrm{A}}$-$r_{2\mathrm{B}}$), therefore, need to be removed from the final policy.

Although the above solution maximizes the individual interdomain accesses, it does not support executions of tasks t_1 and t_2. The execution semantics of task t_1 (t_2) requires accessing roles r_{3B}, r_{2A}, and r_{4A} (r_{3B}, r_{2A}, and r_{5A}) by the user/process invoking the task t_1 (t_2). Constraints c48 and c49 in the IP formulation of Figure 5 capture this semantic dependency. By removing the role mapping (r_{1B}-r_{2A}) from the final interoperation policy, none of the users authorized for role r_{3B} can access role r_{2A}, implying that tasks t_1 and t_2 cannot be executed in the interoperation policy generated by solving the IP of Figure 5. Changing the optimality criterion from maximizing interdomain accesses to maximizing task supportability (*maximize* $t_1 + t_2$) in Figure 5, produces a different interoperation policy in which the interdomain edges ($r_{1\mathrm{A}}$-$r_{1\mathrm{B}}$), ($r_{5\mathrm{A}}$-$r_{2\mathrm{B}}$), and ($r_{3\mathrm{B}}$-$r_{3\mathrm{A}}$) are removed. Since the interdomain edge ($r_{1\mathrm{B}}$-$r_{2\mathrm{A}}$) is retained in this resolution, the resulting interoperation policy, therefore, supports execution of both tasks $t_1 + t_2$. However, the number of individual interdomain accesses in this case is reduced to 5.

The reader will note that both the above solutions to the IP problem eliminate the autonomy and security violations occurring in the initial RBAC graph of Figure 3b. A simple observation of the state of the graph after the removal of afore-mentioned interdomain edges indicates that there remains no such combination of edges that caused those violations.

6. SYSTEM ARCHITECTURE AND IMPLEMENTATION

This section describes the system architecture and the implementation prototype of our administrative model. The architecture is based on the concepts outlined in Sections 3 and 4.

Figure 6 shows the X-GTRBAC Admin system architecture. With reference to Figure 1, we are incorporating support for both aspects of policy administration. Therefore, the architecture has been designed according to Figure 1. The support for administrative extensions related to policy assignments, as described in Section 3, namely Admin Domains, Admin Roles, and Admin Permissions, is provided by the PA (Policy Assignments) module. It carries out the assignment of administrative users to Admin Roles and that of Admin Roles to Admin Permissions. These automated assignments are performed using the *assign_admin_role* and *assign_admin_permission* operations, respectively, as discussed in Section 3. The output of the module is the administrative policy of the enterprise. The CR (Conflict Resolution) module enables secure interoperation among the domains within an organizational structure, as depicted in Figure 4. It implements the decentralized conflict resolution algorithm (DCR) given in Section 4.2.

We use a set of schema definitions that describe the XML documents being shared by the enterprise domains. These definitions are captured using the

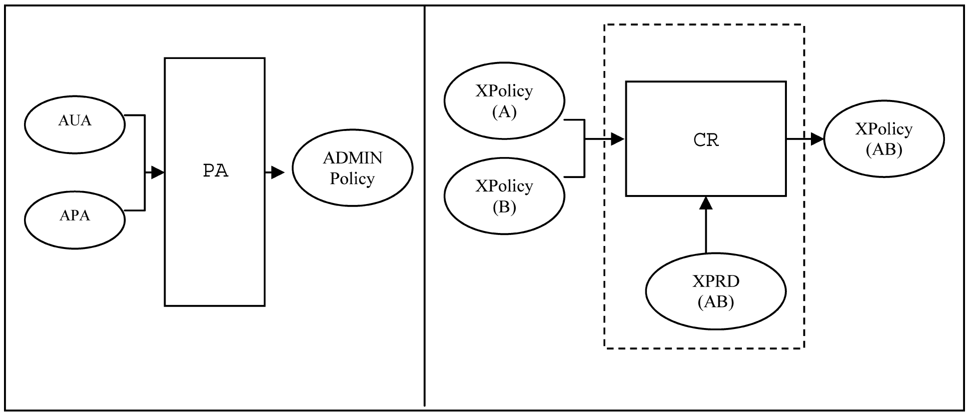
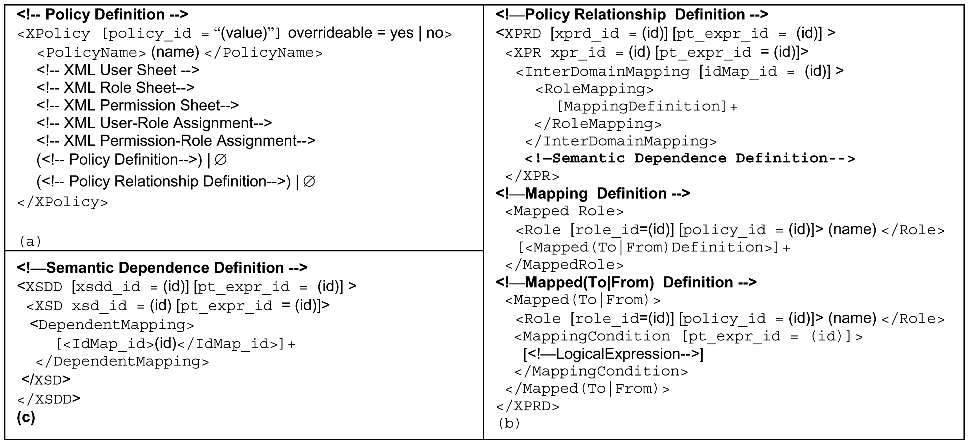


Fig. 6. System architecture for (a) policy assignments (PA); (b) conflict resolution (CR).



```
<!-- Policy Definition -->
<XPolicy [policy_id = "(value)"] overrideable = yes | no>
   <PolicyName> (name) </PolicyName>
   <!-- XML User Sheet -->
   <!-- XML Role Sheet-->
   <!-- XML Permission Sheet-->
   <!-- XML User-Role Assignment-->
   <!-- XML Permission-Role Assignment-->
   (<!-- Policy Definition-->) | ∅
   (<!-- Policy Relationship Definition-->) | ∅
</XPolicy>

(a)
```

```
<!—Semantic Dependence Definition -->
<XSDD [xsdd_id = (id)] [pt_expr_id = (id)] >
 <XSD xsd_id = (id) [pt_expr_id = (id)]>
  <DependentMapping>
     [<IdMap_id>(id)</IdMap_id>]+
   </DependentMapping>
 </XSD>
</XSDD>
(c)
```

```
<!—Policy Relationship  Definition -->
<XPRD [xprd_id = (id)] [pt_expr_id = (id)] >
 <XPR xpr_id = (id) [pt_expr_id = (id)]>
  <InterDomainMapping [idMap_id = (id)] >
    <RoleMapping>
      [MappingDefinition]+
    </RoleMapping>
   </InterDomainMapping>
   <!—Semantic Dependence Definition-->
 </XPR>
<!—Mapping  Definition -->
 <Mapped Role>
  <Role [role_id=(id)] [policy_id = (id)]> (name) </Role>
  [<Mapped(To|From)Definition>]+
 </MappedRole>
<!—Mapped(To|From)  Definition -->
 <Mapped(To|From)>
  <Role [role_id=(id)] [policy_id = (id)]> (name) </Role>
  <MappingCondition [pt_expr_id = (id)]>
   [<!—LogicalExpression-->]
  </MappingCondition>
 </Mapped(To|From)>
</XPRD>
(b)
```

Fig. 7. X-Grammar for (a) XML policy definition (XPolicy); (b) XML policy relationship definition (XPRD); (c) semantic dependence definition (XSDD).

X-Grammar notation introduced Section 2.2. The policy information of each domain (illustrated using RBAC policy graph of Section 4) is represented using an XML document called XPolicy. The XPolicy schema is defined using the specification language defined in Joshi et al. [2004], and is shown in Figure 7a. It captures all the essential policy information, including the users, roles, permissions, and user-to-role and permission-to-role assignments. In addition, if the policy belongs to a dominating domain in the domain tree, then the document also includes imported Policy Definition elements (i.e., policy definitions of the locally dominated domains) and the Policy Relationship Definition elements (i.e., the interdomain role mapping among the locally dominated domains, based on the *map_role* operation defined in Table IV). The XML Policy Relationship Definition (XPRD) schema is shown in Figure 7b. We note that the role mapping may also have temporal constraints associated with it, captured by the

MappingCondition tag.[2] In addition, the XPRD also includes the information regarding the semantic dependence among tasks that must be considered while determining the optimal interoperation, as discussed in Section 4, and illustrated with an example in Section 5. This information is captured by the XML Semantic Dependence Definition (XSDD) schema and is shown in Figure 7c.

The conflict resolution algorithm is implemented as follows. Each locally dominated domain first exports its policies to its dominating domain. For instance, in Figure 4, A and B will both export their XPolicy to node AB. At this point, the administrator of the dominating domain would define any interdomain role mapping among the locally dominated domains using the XPRD schema and appending the Policy Relationship Definition element to the XPolicy document. This document will then be used by the administrator to generate and solve the IP problem for the locally dominated domains. AB generates and solves the IP problem for A and B. The resulting policy will be conflict-free with respect to A and B. This resulting conflict-free policy is itself represented as an XPolicy document. To implement the hybrid scheme of system configuration discussed in Section 4 (i.e., favoring autonomy vs. optimality), the XPolicy document has an overrideable attribute, which can be set to yes or no, depending on the semantics of the target enterprise. A value of no implies that the policy cannot be overridden by the administrator of the dominating domain and, hence, favors autonomy, whereas a value of yes allows the administrator to again include *all* dominated domains in the analysis, thus favoring optimality. The conflict resolution process is recursively carried out until the root of the domain tree is reached, at which point a final XPolicy document is generated. This document is then transmitted to *all* dominated domains (represented by leaf nodes in the domain tree) and represents the overall enterprise interoperation policy. We note that the XPolicy documents exported by the individual domains represent the projected RBAC policy graph. In case any change is detected in policy state or configuration, the corresponding XPolicy documents will be revised and the conflict resolution policies among the affected domains will be regenerated, as per the algorithm described in Section 4.4.

Implementation efforts are underway for supporting the X-GTRBAC Admin model in our existing Java-based prototype, first described in Bhatti et al. [2005]. The `PA` module has been implemented and included in the prototype. The DCR algorithm has been partially implemented as a stand-alone C program. The $\texttt{CR}_2$ subroutine managing secure interoperation for projected policy graphs is currently supported and we are working on including the `detect_change` subroutine to enable dynamic policy analysis. The `CR` module has been coded in C for performance reasons, since it needs to be invoked potentially several times. The Java Native Interface (JNI) API allows the C program to interface with our Java-based prototype.

[2]The pt_expr_id attribute refers to the definition of the periodic time expressions of GTRBAC shown in Table I and the LogicalExpression tag refers to the constraint expression in X-GTRBAC Admin. For details of its XML representation, see Bhatti et al. [2005].

7. DISCUSSION AND CONCLUSION

In this paper, we have presented X-GTRBAC Admin, an administration model for the X-GTRBAC framework. The model presents a solution to the administration problem for enterprise-wide access control, which not only includes authorization management for users and resources within a single domain, but also conflict resolution among access control policies of multiple domains to allow secure interoperation within the enterprise. We maintain that a distinct feature of our approach is that it allows policy administration in the presence of constraints. Our model provides a formal specification of administrative concepts and constraints to facilitate the administration of advanced RBAC policies. The formal model has also been represented using an XML-based grammar specification called X-Grammar, which is well-suited to heterogeneous, interoperable systems, and defines a consistent vocabulary based on XML-Schema to express enterprise access control policies. Both the regular and administrative assignment operations in our X-GTRBAC system are treated uniformly, which keeps the administrative concept simple in practice. A key feature of the model is that it allows decentralization of the policy administration tasks through the abstraction of administrative domains, which not only simplifies authorization management, but is also fundamental to the concept of decentralized conflict resolution presented in the paper. The model formalizes the organizational structure of a multidomain enterprise using role and domain hierarchies, and incorporates a decentralized conflict resolution algorithm to allow secure interoperation in a multidomain environment. A generic enterprise example has been provided to consolidate the ideas presented in the paper. We have also presented a comprehensive software architecture and discussed the system implementation.

The decentralized conflict resolution mechanism is a significant feature of our model. It allows the domain administrators to define their security requirements and allows secure interoperation across the heterogeneous enterprise domains while respecting the autonomy and security requirements of the constituent domains. No earlier administrative model (ARBAC, [Crampton and Loizou 2002]) has addressed the issues of conflict resolution in a multidomain environment. There has been research on policy management along related lines, such as [Dawson et al. 2000; Bonatti et al. 1996, 2002; Gong and Qian 1996]. However, to the best of our knowledge, this is the first approach that deals with conflict resolution in a dynamic environment using a temporal access control framework. There are tradeoffs, however, that need to be made in the favor of either autonomy of domains or optimality of the overall solution. These tradeoffs depend on the target enterprise and, hence, our model supports a hybrid scheme that allows favoring one criterion over the other. While the autonomy criterion is met affordably using the domain tree construction, the optimality criterion is expensive to meet in a highly dynamic environment due to the heavy processing requirements at each level of the domain tree. We have, therefore, studied several heuristics (such as Lagrangian relaxation, tabu search, and simulated annealing [Zhang 2004]) that would make compliance with this criterion less expensive to achieve. In other words, it will reduce the amount of processing overhead encountered to solve the IP problem for each instance (i.e., projection) of the policy graph. This, however, is not without its cost—the approximation results in a near-optimal solution. It is desirable to find better heuristics that can bridge the gap between the near optimal and optimal solution, although attaining the latter with long-lived guarantees is, most likely, not feasible. Addressing this concern will be the focus of further work.

Several extensions to our model can be foreseen. Our present analysis of multidomain policies has assumed that the individual domain policies are consistent. An ongoing work deals with removing this assumption and addressing the consistency problem for a single-domain policy. We intend to explore other issues related to administration of policies in multidomain environments, such as assignment criteria of domain administrators, interplay of administrative authority and advanced role hierarchies, and evaluating the administrative authority in the wake of domain merging or splitting. Part of the complexity in determining administrative authority in a multidomain environment is introduced because of the constraint specification in our framework. The contextual constraints (both temporal and nontemporal) imposed on a user-to-admin-role assignment may lead to a situation when no administrator remains eligible to be assigned to a domain, whereas, there may also be times when multiple candidates are available. Another factor that complicates this issue is the trustworthiness of the domain administrators. A protocol must exist that allows the multiple domains in the enterprise to reach a consensus on the authority of the multidomain administrators managing their access control policies. Addressing these concerns is a policy design issue and needs to be considered as part of the administration problem. These challenges need to be addressed for effective administration of access control policies in a widely distributed dynamic enterprise.

APPENDIX A

```
<!-- XML User Sheet>     ::=
<XUS [xus_id = (id) ] >
 <User user_id = (id)>
  <UserName> (name) </UserName>
  <CredType cred_type_id = (id)
             type_name= (type name) >
     <!-- Attribute List>
 </CredType >
 <MaxRoles>(number)</MaxRoles>
 </User>
</XUS>
```

Fig. A.1. X-Grammar for XUS.

```
<!-- XML Role Sheet> ::=
<XRS [xrs_id = (id) ]>
 <Role role_id =(id) role_name =(role name)>
     <CredType cred_type_id = (id)
              type_name= (type name) >
      <!-- Attribute List>
     </CredType >
   [<!--{En|Dis}abling Constraint>]
   [<!--[De]Activation Constraint>]
   [<Cardinality> (number) </Cardinality>]
 </Role>
</XRS>
```

Fig. A.2. X-Grammar for XRS.

```
<!—XML Permission Sheet> ::=
<XPS [xps_id = (id) ]>
 <Permission perm_id = id
  <Object type= (type name)  id= (id)>
  [<!-- Attributes>] </Object>
  <Operation>(op) </Operation>
 </Permission>
</XPS>
```

Fig. A.3. X-Grammar for XPS.

```
<!-- XML User-to-role Assignment Sheet>  ::=
<XURAS [xuras_id = (id) ]>
   {<!-- User-to-role Assignment>}+
</XURAS>
```

```
<!-- User-to-role Assignment>              ::=
<URA ura_id=(id) role_name=(name)>
<[De]AssignUsers>
      {< !--[De]Assign User>}+
</[De]AssignUsers>
</URA>
```

```
<!--[De]Assign User >    ::=
<[De]AssignUser
    user_id=(id)>
 <!--[De]Assign User Constraint>
</[De]AssignUser>
```

```
 <!--[De]Assign User Constraint> ::=
<[De]AssignUserConstraint
   [op = {AND|OR|NOT|XOR}]>
   <!--[De] Assign User Condition>
</[De]AssignUserConstraint>
```

```
<!--[De]Assign User Condition>  ::=
<[De]AssignUserCondition
   cred_type="type_name"
   [{pt_expr_id=(id) |
    d_expr_id=(id)}] >
   [<!-- Logical Expression>]
</[De]AssignUserCondition>
```

Fig. A.4. X-Grammar for XURAS.

```
<!-- XML Permission-to-role Assignment Sheet>  ::=
<XPRAS [xpras_id = (id) ]>
   {<!-- Permission-to-role Assignment>}+
</XPRAS>
```

```
<!-- Permission-to-role Assignment>       ::=
<PRA pra_id=(id) role_name=(name)>
<[De]AssignPermissions>
      {< !--[De]Assign Permission>}+
</[De]AssignPermissions>
</PRA>
```

```
<!--[De]Assign Permission >    ::=
<[De]AssignPermission
     [{pt_expr_id=(id) |
    d_expr_id=(id)}]
   {<PermId>(id)</PermId>}+
</[De]AssignPermission>
```

Fig. A.5. X-Grammar for XPRAS.

APPENDIX B

Table B.1. Transformation Rules for Generating IP Constraints from RBAC Policy Graph

Category	ID	Rule	Meaning
Hierarchy and assignment	1	$\neg reachable(u_i, r_j) \Rightarrow u_{ir_j} = 0$	If there is no access path from a user node u_i to role/permission node r_j, then ui is not authorized to access r_j
	2	Let A_u be the set of users assigned to r_j. $\sum_{u \in A_u} u_{r_j} \geq 0$, where, p_k is a dynamic permission assigned to r_j.	At least one of the users from the set of users assigned to a role must access that role in any feasible solution of IP
	3	For an intradomain I hierarchy edge from role r_j to r_k, $u_{ir_j} - u_{ir_k} \leq 0$	Any user u_i assuming role r_j also assumes r_k, if role r_j is senior to r_k
	4	Let $U_{Ik} = \{u \mid \neg uassign(u, r) \wedge (r = r_k \vee r \geq^r_k)\}$ and $R_{Ik} = \{r_j \mid r_j \geq r_k\} \cdot \forall u \in U_{IK}$, $\sum_{r_j \in R_{IS}} u_{r_j} - u_{r_k} \geq 0$.	Any user u not assigned to r_k or any of its senior roles can access r_k only if u is able to access at least one role in the set R_{Ik}
	5	Consider a user u_i and a role r_k such that $domain(u_i) \neq domain(r_k)$. Let $R_m = \{r \mid domain(r) = domain(r_k) \wedge reachable(u_i, r_k)\}$, and $\mathrm{R_c} = \{r \mid r \geq_{r_k} \wedge domain(r_k) \neq domain(r)\}$ and $\mathrm{R_{pc}} = \{r_p \mid \exists r \in \mathrm{R_c}$ such that$(r_p = r \wedge uassign(u, r)) \vee (r_p \geq r \wedge domain(r) = domain(r_p))\}$ The following IP constraints define the conditions for u_i to access r_k. $\forall r_m \in \mathrm{R_m},\ u_{ir_m} - u_{ir_k} \leq 0$ $\sum_{r_m \in R_m} u_{ir_m} + \sum_{r_n \in \mathrm{R_c}} u_{ir_n} - u_{ir_k} \geq 0,$ $\sum_{r_m \in Rm} u_{ir_m} + \sum_{r_p \in \mathrm{R_{pc}}} u_{ir_p} - u_{ir_k} \geq 0$	A user u_i may access a cross-domain role r_k only if one of the following two conditions holds: (i) u_i is authorized for a cross-domain role r_m such that $domain(r_m) = domain(r_k)$ and $r_m \geq_I r_k$. (ii) u_i is authorized for role r_n and there is an interdomain edge from r_n to r_k
	6	For an interdomain edge from r_k to r_l, the following constraints define the interdomain access semantics for any two users u_i and u_j via $(r_k - r_l)$. *if* $domain(u_i) = domain(u_j) = domain(r_k)$ then $(u_{ir_k} - u_{ir_l}) - (u_{jr_k} - u_{jr_l}) = 0$ *else* $(u_{ir_k} - u_{ir_l}) -$ $(u_{jr_k} - u_{jr_l}) \geq 0$	If user u_i is able to access r_l through the cross-domain edge (r_k, r_l), then any user u_j, if authorized for role r_k, can also access r_l through (r_k, r_l)
SoD	7	For a role SoD constraint between r_j and r_k, $\forall u \in \texttt{Users}, u_{\mathrm{r}} + u_{\mathrm{r}} \leq 1$	Conflicting roles or permissions cannot be accessed by same user simultaneously
	8	Let U_c be the set of conflicting users for role r_k, then $\sum_{u \in U_c} u_{r_k} \leq 1$	Conflicting users cannot access same role/permission concurrently
Task-dependence	9	Let S_T be a set of user role accesses that need to be invoked for execution of task T_i, then $T_i - \prod_{u_r \in S_T} u_r = 0$	A task Ti can execute if all the user-role accesses in the semantic dependency set of $\mathrm{T_i}$ are satisfied.

REFERENCES

BACON, J., MOODY, K., AND YAO, W. 2002. A model of OASIS role-based access control and its support for active security. *ACM Transactions on Information and System Security (TISSEC) 5*, 4(Nov.).

BHATTI, R., JOSHI, J. B. D., BERTINO, E., AND GHAFOOR, A. 2005. X-GTRBAC: An XML-based policy specification framework and architecture for enterprise-wide access control. *ACM Transactions on Information and System Security (TISSEC)*, *8*, 2 (May).

BONATTI, P. A., SAPINO, M. L., AND SUBRAHMANIAN, V. S. 1996. Merging heterogeneous security orderings. *ESORICS*. 183–197.

BONATTI, P. A., VIMERCATI, S., AND SAMARATI, P. 2002. An algebra for composing access control policies." *ACM Transactions on Information and System Security*, *5*, 1 (Feb.). 1–35.

CRAMPTON, J. AND LOIZOU, G. 2002. Administrative scope and role hierarchy operations. In *Proceedings of 7th ACM Symposium on Access Control Models and Technologies* (June).

DAWSON, S., QIAN, S., AND SAMARATI, P. 2000. Providing security and interoperation of heterogeneous systems. *Distributed and Parallel Databases*, *8*, 1, 119–145.

FERRAIOLO, D. F., SANDHU, R., GAVRILA, S., RICHARD KUHN, D., AND CHANDRAMOULI, R. 2001. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC), 4*, 3 (Aug.).

GONG, L. AND QIAN, X. 1996. Computational issues in secure interoperation. *IEEE Transaction on Software and Engineering, 22*, 1 (Jan.).

JOSHI, J. B. D., BERTINO, E., LATIF, U., AND GHAFOOR, A. 2005. Generalized temporal role based access control model (GTRBAC)- Specification and modeling. *IEEE Transaction on Knowledge and Data Engineering, 17*, 1 (Jan.).

JOSHI, J. B. D., BERTINO, E., GHAFOOR, A. 2002. Temporal hierarchies and inheritance semantics for GTRBAC. In *Proceedings of 7th ACM Symposium on Access Control Models and Technologies* (June).

JOSHI, J. B. D., BHATTI, R., BERTINO, E., AND GHAFOOR, A. 2004. X- RBAC An access control language for multidomain environments. *IEEE Internet Computing, 8*, 6, 40–50 (Nov./Dec.).

OH, S. AND SANDHU, R. 2002. A model for role administration using organization structure. In *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies* (June).

SANDHU, R., COYNE, E. J., FEINSTEIN, H. L., AND YEOMAN, C. E. 1996. Role based access control models. *IEEE Computer 29*, 2 (Feb.).

SANDHU, R. 1998. Role activation hierarchies. In *Proceedings of the 3rd ACM Workshop on Role-Based Access Control* (Oct.). 33–40.

SANDHU. R. AND MUNAWER, Q. 1999. The ARBAC99 model for administration of roles. In *Proceedings of the 15th Annual Computer Security Applications Conference* (Dec.).

SHAFIQ, B., JOSHI, J., BERTINO, E., AND GHAFOOR, A. 2005. Secure interoperation in a multidomain environment. Accepted for publication in IEEE Transaction on Knowledge and Data Engineering, *17*, 11 (Nov.).

ZHANG, H. 2001. Improving constrained nonlinear search algorithms through constraint relaxation. Masters thesis, University of Illinois at Urbana Champaign, Urbana, IL.

Received October 2004; revised April 2005; accepted September 2005