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

Rafae Bhatti
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN, USA
rafae@purdue.edu

James B. D. Joshi
School of Information Sciences
University Of Pittsburgh
Pittsburgh, PA, USA
jjoshi@.mail.sis.pitt.edu

Elisa Bertino
Department of Computer Sciences and CERIAS
Purdue University
West Lafayette, IN, USA
bertino@cs.purdue.edu

Arif Ghafoor
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN, USA
ghafoor@purdue.edu

ABSTRACT
Access control in enterprises is a key research area in the realm of Computer Security because of the unique needs of the target enterprise. As the enterprise typically has large user and resource pools, administering the access control based on any framework could in itself be a daunting task. This work presents X-GTRBAC Admin, an administration model that aims at enabling policy administration within a large enterprise. In particular, it simplifies the process of user-to-role and permission-to-role assignments, and thus allows decentralization of the policy administration tasks. Secondly, it also allows for specifying the domain of authority of the system administrators, and hence provides mechanism to distribute the administrative authority over multiple domains within the enterprise. The paper also illustrates the applicability of the administrative concepts presented in our framework for enterprise-wide access control.

Categories and Subject Descriptors
D.4.6 [**Security and Protection**]: Access controls; H.2.7 [**Database Administration**] Security, integrity, and protection.

General Terms
Design, Security, Theory.

Keywords
Role based access control, decentralized administration, temporal constraints, XML.

1. INTRODUCTION
Modern day enterprises are faced with the challenge of achieving efficient resource utilization to maintain a competitive edge, and simultaneously ensuring secure interoperation across its constituent domains [1,2]. The access control challenges for an enterprise range from (i) the need to be able to support an access control policy at multiple points of enforcement (i.e. administrative domains) within the enterprise, and to express and communicate the policies in a language that supports interoperation between the collaborating domains, to (ii) the need to be able to express the sophisticated real-time constraints specific to the dynamically changing access requirements within the enterprise. These challenges have been highlighted in [3], 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 [4]. X-GTRBAC augments GTRBAC with XML to allow for supporting the policy enforcement in a heterogeneous, distributed environment. The work presented in [3] also provides a software architecture and a prototype implementation for X-GTRBAC.

Although the X-GTRBAC framework has been designed with the goal of facilitating enterprise-wide access control, the administration of the model may pose several challenges due to the huge pool of enterprise users and resources. In fact, any access control scheme may not be fruitful unless proper administrative mechanisms are provided to ensure effective policy administration. Although X-GTRBAC has a mechanism to automate the user-to-role and permission-to-role assignments, the task of managing a huge number of users and resources 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 [5]. To mitigate this concern, we introduce X-GTRBAC Admin, the administration model for the X-GTRBAC framework. The primary focus of this paper is to elucidate the administrative concepts related to X-GTRBAC and discuss the motivation and specification of the proposed administration model.

The remainder of this paper is organized as follows. We begin with the background and motivation of our particular approach. The salient features of the X-GTRBAC specification language are thereby outlined. We next present X-GTRBAC Admin, the administrative model for the X-GTRBAC framework for enterprise-wide access control, and consolidate the ideas presented with the discussion of a generic enterprise example. A survey of related work in the area of access control schemes and associated administration models is then provided. The paper concludes with a discussion on the merits of our particular work, and a sketch of future research goals.

SACMAT'04, June 2–4, 2004, Yorktown Heights, New York, USA.

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

X-GTRBAC is an XML-based policy specification framework that builds on the GTRBAC model [4]. GTRBAC extends the widely accepted Role Based Access Control (RBAC) model proposed in the NIST RBAC standard [6]. RBAC uses the concept of roles to embody a collection of permissions within an organizational setup. Permissions are associated with roles through a permission-to-role assignment, and the users are granted access to resources through a user-to-role assignment [7]. A major advantage of the RBAC model is that it simplifies authorization administration in large enterprises. RBAC models have been shown to be policy-neutral, in that they can be used to represent a variety of security policies, including both DAC and MAC policies [8]. Although several approaches have been presented in the literature based on RBAC to address various aspects of security administration within an enterprise, they have their own drawbacks that render them unsuitable for enterprise-wide access control [3]. GTRBAC provides a generalized mechanism to express a diverse set of fine-grained temporal constraints on user-to-role and permission-to-role assignments in order to meet the dynamic access control requirements of an enterprise. X-GTRBAC framework augments the GTRBAC model with XML to allow for supporting the policy enforcement in a heterogeneous, distributed environment. The motivation for using XML as the language of choice for specifying GTRBAC policies is the heterogeneity of collaborating entities, within a large distributed enterprise environment, that enable high level information system services. The functional entities within an enterprise, connected through multiple media, and each comprised of heterogeneous information systems that are linked together by the Enterprise Computing (EC) technology [1], require a common policy specification language to efficiently express and enforce the enterprise level access control policy. As XML provides a uniform, vendor-neutral representation of enterprise data, and allows a mechanism for interchange, sharing and dissemination of information content across heterogeneous systems, the access control needs of an enterprise can adequately be addressed through an XML-based framework.

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

RBAC Model [6] 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:* Users × Roles, the user assignment function, that assigns users to roles;
- *assigned_users(r:* Roles*)*→ 2^{Users}, the mapping of role *r* onto a set of users. Formally: *assigned_users(r)* = {*u* ∈ Users | (*u*,*r*) ∈ UA}
- *PA:* Roles × Permissions, the permission assignment function, that assigns permissions to roles;
- *assigned_permissions(r:* Roles*)* → $2^{\text{Permissions}}$, the mapping of role *r* onto a set of permissions. Formally: *assigned_permissions(r)* = {*p* ∈ Permissions | (*p*,*r*) ∈ *PA*}
- *user:* Sessions → Users, which maps each session to a single user;
- *role:* Sessions → 2^{Roles} that maps each session to a set of roles;
- *RH* □ Roles × Roles, a partially ordered role hierarchy (written ≥).

Session s_i has the permission of all roles *r'* junior to roles activated in the session, i.e.

{*p* | (∀ r in roles(s_i) and all *r'* ≤ r)[(*p*,*r*) or (*p*,*r'*)∈ *PA*]}

GTRBAC Model [4] The GTRBAC model incorporates a set of language constructs for the specification of various temporal constraints on roles, including constraints on their activations as well as on their enabling times, 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 constraints in GTRBAC allow the specification of the following constraints and events:

1. *Temporal constraints on role enabling/disabling*: These constraints allow one to specify the time intervals during which a role is enabled. When a role is enabled, the permissions assigned to it can be acquired by a user by simply activating the role. It is also possible to specify a role duration. When such a duration is specified, the enabling/disabling event for a role is initiated by a constraint-enabling expression that may be separately specified at run-time by an administrator or by a trigger.
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 enable duration or activation constraints. Another set of run-time events allow users to make activation requests to the system.
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 based on them.

A periodic expression is written as *(I,P),* where *I* is an interval and *P* is a set of infinite number of intervals. (*I,P)* represents the

set of all intervals such that P is contained in I. D is used to express the duration specified for a duration constraint. The temporal constraint types and expressions in GTRBAC are

context-free grammar called X-Grammar, which follows the same notion of terminals and non-terminals as in BNF, but supports the tagging notation of XML that also allows expressing attributes

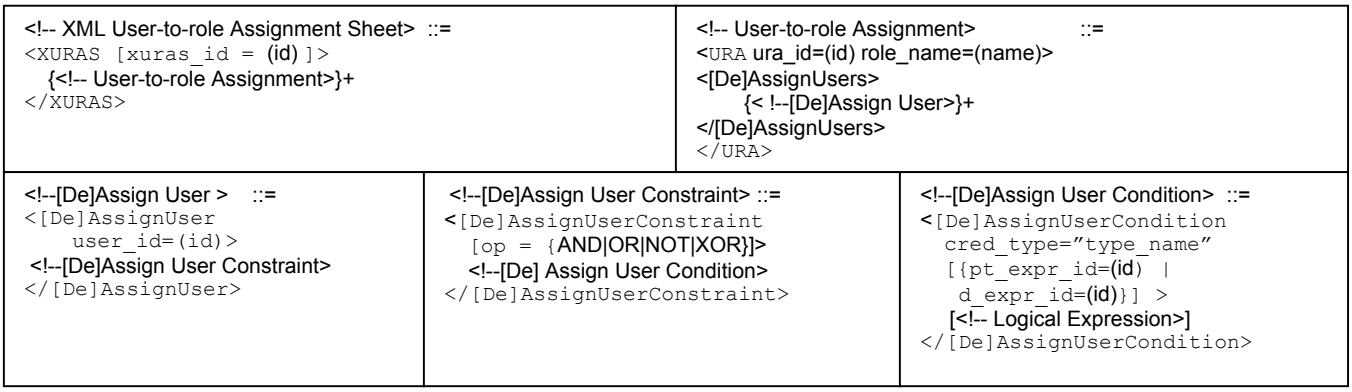

Figure 1: X-Grammar for XURAS

Table 1. 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], $assign_U$/$deassign_U$ *r* to *u*)
	Permission-to-role assignment	([I, P, D], $assign_P$/$deassign_P$ *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	($assign_U$/de-$assign_U$ *r* to *u* after Δt)
		(enable/disable *r* after Δt)
		($assign_P$/de-$assign_P$ *p* to *r* after Δt)
		(enable/disable *c* after Δt)

summarized in Table 1.

X-GTRBAC allows specification of all the elements of the GTRBAC model. These specifications are captured through a

within element tags. The detailed specification of these elements of X-GTRBAC framework can be found in [3]. For the purposes of our present discussion, we focus in the next subsection on the mechanisms of user-to-role and permission-to-role assignments using their corresponding X-Grammar representations. We then introduce in Section 3 the administrative extensions to the GTRBAC model, and present the formal definition and X-Grammar for the components of X-GTRBAC Admin.

2.2 Motivation for an Admin Model

The assignment and activation conditions on roles can be specified in X-GTRBAC as constraint statements. As mentioned earlier, our framework makes a distinction between assignment and activation of a role. We consider the result of a user-to-role assignment operation as the set of eligible users who could potentially activate the specified roles. Activation of a role only takes place for the eligible users when an access request is made, subject to the evaluation of an associated activation constraint. Hence the assignment conditions capture the static (i.e. assignment-time) context available through supplied user-credentials, and the activation conditions capture the dynamic (i.e. activation-time) context available at the time when the access requests are made. Both the assignment-time and activation-time constraints are provided by the System Security Officer (SSO) using the X-Grammar for GTRBAC elements and functions. The X-Grammar for *user* specifies a list of user credentials that may be used in assignment to roles. Similarly, the X-Grammar for *role* specifies a list of role attributes that may be parameters of the context conditions which need to be dynamically evaluated for any role enabling/disabling or activation/deactivation, or for assignment of eligible permissions to the role. The structure allows evaluation of nested conditions expressed by multiple *logical expressions* within a constraint statement. An XML User-to-Role Assignment Sheet (XURAS) is created by the SSO to supply the assignment conditions on user-to-role assignment. Similarly, the X-Grammar for the XML Permission-to-Role Assignment Sheet (XPRAS) is used to specify the assignment conditions on permission-to-role assignment. The X-Grammar for the corresponding sheets is shown in Figures 1 and 2.

<!-- 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>

Figure 2: X-Grammar for XPRAS

The evaluation of assignment constraint expressions in the model has direct relevance to our current discussion related to the administrative concepts in X-GTRBAC[1]. This mechanism allows the specification of automated assignment of users to roles based on the user credentials. Credential based dynamic assignments of users to roles allows the administration of access control policies by defining rules on credential attributes. Similarly, permission-to-role assignment mechanism automates the process of associating permissions with roles. However, as we have discussed, administering these policy assignments would be a challenging task in large enterprises, as the administration of roles becomes increasingly complex with the increase in the size of the user and resource pools of the enterprise. Hence, in order to attain effective and scalable enterprise wide access control, our framework needs to be augmented with an administration model. We next present X-GTRBAC Admin as a natural extension to the X-GTRBAC framework.

3. X-GTRBAC Admin

X-GTRBAC Admin is introduced to simplify the process of user-to-role and permission-to-role assignments within the X-GTRBAC framework. The latter lends itself well to an administrative extension because the original model has emphasized separation of language schemas to provide distinct specification of definitions of RBAC elements, user-to-role and permission-to-role assignments and hierarchical and separation of duty constraints. Hence, this modular approach not only makes it easy to extend one component of the model independently of the other, but also complements the decentralized administration goal by distributing the tasks into multiple domains, each responsible for its own set of policy specifications. For example, the task of assigning roles to users is distinct from that of assigning permissions to roles within the enterprise, and hence the two assignment specifications can be constructed independent of each other. Furthermore, these tasks could further be separated into multiple domains within the enterprise. To enforce common vocabulary, however, definition sheets for the different entities (like credential types, separation of duty constraints, temporal constraints) within the system are provided that must be adhered to across all domains.

We now turn to the specification of our administrative model. In order to include the administrative concept in our X-GTRBAC framework, the specification language is extended to include the specification of an Administrator Role (AdminRole) and an Administrative Permission (AdminPermission). An important notion introduced here is that of an Administrative Domain (Admin Domain) which is the key to scalable decentralization of the administrative tasks within the enterprise. Each Admin Role and Admin Permission is associated with an Admin Domain.

The formal extension to the GTRBAC model is presented below.

Definition: The X-GTRBAC Admin model consists of the following extensions to its GTRBAC component:

- AD = $\{ad_1, \ldots, ad_k\}$, a set of administrative domains
- AU = $\{au_1, \ldots, au_k\}$, a set of administrative users, AU □ Users
- RR = $\{rr_1, \ldots, rr_k\}$, a set of regular roles
- RO = $\{ro_1, \ldots, ro_k\}$, a set of regular operations
- AR = $\{ar_1, \ldots, ar_k\}$, a set of administrative roles
- AO = $\{ao_1, \ldots, ao_k\}$, a set of administrative operations
- The set of regular roles RR for a domain ad ∈ AD is defined as
  $RR_D = \{(ad, rr) \mid ad \in AD, rr \in RR\} \subseteq RR$
- The set of regular permissions RP for a domain ad ∈ AD is defined as
  $RP = AD \times RO = \{(ad, ro) \mid ad \in AD, ro \in RO\}$
- The set of administrative roles AR for a domain ad ∈ AD is defined as
  $AR_D = \{(ad, ar) \mid ad \in AD, ar \in AR\}$
- The set of administrative permissions AP in domain ad ∈ AD is defined as
  $AP = AD \times AO = \{(ad, ao) \mid ad \in AD, ao \in AO\}$
- *domain(r)* returns the domain of a role. Formally: *domain(r | r:* RR or *r:* AR*)* = $\{d \in AD \mid (d,r) \in RR_D \text{ or } (d,r) \in AR_D\}$

[1]The activation constraints are an enforcement mechanism, and hence not directly part of the administrative component of the model. The administration problem is conventionally viewed as one of dealing with user-to-role and role-permission assignments. This process is independent of what activation conditions occur on roles, and those are specified separately by the SSO in the X-Grammar for Roles.

- *administers(ar)* returns the set of all regular roles administered by an administrative role. Formally: *administers(ar | ar:* AR*)* = {*rr* | ($\forall$ *ad* $\in$ domain(*ar*))[(*ad*,*rr*) $\in$ RR_D]}
- ***assigned_users****(rr:* RR_D) $\rightarrow$ 2^{Users}, the mapping of regular role *rr* onto a set of users. Formally: *assigned_users(rr)* = {*u* $\in$ Users | (*u*, *rr*) $\in$ UA}
- ***assigned_permissions****(rr:* RR_D) $\rightarrow$ $2^{Permissions}$, the mapping of regular role *rr* onto a set of permissions. Formally: *assigned_permissions(rr)* = {*p* $\in$ Permissions | (*p*,*rr*) $\in$ PA}
- *AUA:* AU $\times$ AR, the administrative user assignment function, that assigns users to Admin Roles;
- *assigned_admin_users(ar:* AR_D)$\rightarrow$ 2^{AU}, the mapping of administrative role *ar* onto a set of users. Formally: *assigned_admin_users(ar)* = {*au* $\in$ AU | (*au*, *ar*) $\in$ *AUA*}
- *APA:* AR $\times$ AP, the administrative permission assignment function, that assigns Admin Permissions to Admin Roles;
- *assigned_admin_permissions(ar:* AR_D) $\rightarrow$ 2^{AP}, the mapping of administrative role ar onto a set of administrative permissions. Formally: *assigned_admin_permissions(r)* = {*ap* $\in$ AP | (*ap*,*r*) $\in$ *APA*}

Admin Role: An administrator in an Admin Role is authorized to handle assignment of users to regular roles within a given Admin Domain. This authority is given by a set of associated Admin Permissions (which are discussed below). An Admin Role is represented in our framework in an XML Admin Role Sheet (XARS), an instance of which is shown in Figure 3. Typically a set of selected candidate users for the Admin Role within various Admin Domains of the enterprise would be created by the respective SSOs. We introduce a credential `admin` to specifically identify a set of users being considered for AdminRoles, and an optional "`target_domain`" credential to indicate a restriction on their target domains. The assignment of such users to Admin Roles may involve evaluation of other user-specific credentials, as is needed in the case of regular roles, and may as well be based on context conditions (such as a day_time vs. night_time administrator, or regular_hours vs. emergency_hours administrator). This assignment is handled by an *AUA* function similar to the *UA* function of the original model, and is represented in an X-Grammar syntax similar to that of XURAS of Figure 1. Admin Roles are constrained by enabling and activation constraints similarly as regular roles, and have a cardinality attribute that is also interpreted similarly. In addition, the scope of the administrative authority for the Admin Roles is restricted to a set of Admin Domains within the enterprise. Each Admin Role may have authority over multiple domains. This scope is defined by the SSO or the system designers when the policy sheets are composed, and is updateable at runtime by the SSO. X-GTRBAC Admin is, thus, designed to allow specification of domains of authority in order to provide a fine-grained mechanism to

```
<!-- XML Admin Role Sheet> ::=
<XARS [xars_id = (id) ]>
  {<!-- Admin Role Definition>}+
</XARS>
```

```
<!-- Admin Role Definition> ::=
<AdminRole admin_role_id = (id)
       admin_role_name = (role name)>
   [<!--Attributes>]
   [<!--{En|Dis}abling Constraint>]
   [<!--[De]Activation Constraint>]
   {<DomainID> (id) </DomainID>}+
   [<Cardinality> (number) </Cardinality>]
</AdminRole>
```

Figure 3: X-Grammar for XARS

```
<!-- XML Admin Permission Sheet> ::=
<XAPS [xaps_id = (id) ]>
  {<!-- Admin Permission Definition>}+
</XAPS>
```

```
<!-- Admin Permission Definition> ::=
<AdminPermission admin_perm_id = id
                 domain= (id)>
  {<PermId>(id)<PermId>}+
</AdminPermission>
```

Figure 4: X-Grammar for XAPS

The assignment functions in X-GTRBAC Admin are modified to include the domain of the users, roles and permissions. The titles in bold indicate the changed definitions. We next explain the usage of the model for the assignment of AdminRole and AdminPermissions within the various domains across an enterprise.

distribute the administrative authority according to the functional units within the enterprise. This not only results in simplified policy administration, but also keeps in check undue authorizations through cascading or collusion that could inflict damage onto the system. Note that the administrative level constraints imposed by the XARS introduce domain-specific restriction on top of those enforced by the XURAS. This means that the assignment of a user to a regular role per the XURAS

Table 2: A set of regular users.

#	Domain	User Id	Eligible Role (ER)
1	ENG	john	R1, R5
2	ENG	nancy	R2
3	HR	george	R3, R6
4	FIN	carla	R4

Table 3: A set of regular permissions.

#	Domain	Perm Id	Eligible Role (ER)
1	ENG	P1	R1
2	ENG	P2	R2
3	HR	P3	R3
4	FIN	P4	R4

Table 4: A set of Admin Roles.

#	Admin Role (AR)	Valid Intervals	AR Domain
1	AR1	MO-FR 9-5	ENG
2	AR2	SA-SU 10-4	ENG
3	AR3	MO-FR 9-5	HR, FIN
4	AR4	TEMP	SPECIAL

Table 5: A set of Admin Permissions.

#	Admin Permission (AP)	AP Domain
1	AP1 (can_assign,can_deassign)	ENG
2	AP2 (can_assign)	HR, FIN
3	AP3 (can_deassign)	HR, FIN
4	AP4 (can_review)	ALL

may be executed by an administrator in an Admin Role whose administrative domain specified in XARS is the same as that of the regular role. Both XURAS and XARS could thus be used jointly to constrain both the context and scope, respectively, of the user-to-role assignment. Hence the modularity of the language schemas allow the SSO to configure the system in various modes, depending on the level of decentralized administration deemed necessary for the target enterprise.

Admin Permission: An Admin Permission specifies a collection of permissions associated with an Admin Role belonging to a particular Admin Domain. An Admin Permission is represented in our framework in an XML Admin Permission Sheet (XAPS), an instance of which is shown in Figure 4. Typically a set of available permissions for the various Admin Domains within the enterprise would be created by the respective SSOs. We introduce *can_assign*, *can_deassign, can_enable*, *can_disable,* and *can_review* as the basic set of Admin Permissions. The meanings of these permissions are straightforward; for instance, *can_assign* permission for a given domain means that the corresponding Admin Role can assign users to roles within that domain; and so on. Because the assignment of Admin Permissions to Admin Roles is based on the attributes of the role and the context conditions provided in the role definition, it is handled by an *APA* function similar to the *PA* function of the original model, and is represented in an X-Grammar syntax similar to that of XPRAS of Figure 2. A prerequisite for this assignment is that the domain of the Admin Permission should be the included in the set of Admin Domains for the Admin Role. The mechanism of assignment of Admin Permissions thus contains the scope of authority of the administrators by restricting the set of available permissions that could be assigned by them to roles, and hence prohibiting any permission flow outside their respective domains. Note that the administrative level constraints imposed by the XAPS introduce domain-specific restriction on top of those enforced by the XPRAS. This means, for instance, that the assignment of a permission to a regular role per the XPRAS may be executed by an administrator in an Admin Role who has been assigned an Admin Permission *can_assign* such that the permission belongs to the corresponding domain specified in XAPS. Both XPRAS and XAPS could thus be used jointly to constrain both the context and scope, respectively, of the permission-to-role assignment. We again maintain that this separation of administrative and access layers leads to a flexible decentralized administration scheme for the target enterprise.

We next present an example of a generic enterprise that demonstrates how the features of X-GTRBAC Admin would be useful in our X-GTRBAC framework for enterprise-wide access control.

4. 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. Let the users and permissions from within various domains within the enterprise be given in Tables 2 and 3 respectively. We assume the user-to-role and permission-to-role assignment criteria for the regular roles have been specified by the SSO, using the XURAS and XPRAS sheets in our framework. The last column in these tables, hence, lists the "eligible" role that the user or permission could be associated with, provided the assignment conditions are

satisfied. Tables 4 and 5 give the candidate users for the Admin Roles and the set of available Admin Permissions, respectively, for the various domains within the enterprise. We next observe the administrative features provided by X-GTRABC Admin to administer the enterprise access control policy.

Assignment of administrative roles and permissions: The assignment of administrators to Admin Roles AR1-AR4, and the assignment of Admin Permissions AP1-AP4 to these Admin Roles is done by X-GTRBAC Admin by using a similar mechanism as the XURAS and XPRAS shown in Figures 1 and 2 respectively. 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 discussion. It may be noted that the context conditions supplied in Table 4 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. We emphasize that our framework allows for context conditions other than time to be specified as well. For instance, the role activation may also depend on a pre-requisite event sequence to have completed, as is typically the case in Workflow Management Systems (WFMS). Such pre-requisite conditions may be expressed as constraint conditions in X-GTRBAC, and dynamically evaluated at the time of a role activation request.

From the information in Tables 4 and 5, we note that AR1 and AR2 can only be assigned AP1, whereas AR3 can be assigned AP2 and AP3 because it has administrative authority over the respective domains to which these permissions belong. Also, AP4 can 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 non-usual 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.

Assignment of regular roles and permissions: The administrators in Admin Roles can then execute the assigned permissions within their respective domains. For instance, the Admin Role AR1 (or AR2) can assign the user john to role R1 because it has the required permission (AP1) and required scope (i.e. its domain is same as the domain of R1). AR3 has *can_assign* permission (AP2) over the domains of HR and FIN, and can hence assign george and carla to their respective eligible roles. Also, the permissions P1 and P2 will be acquired by the roles R1 and R2, whereas P3 and P4 will be acquired by the roles R3 and R4, respectively.

Hierarchical relationships between roles: The Admin Roles in an enterprise may be related by I, A or IA temporal hierarchy relations proposed in [9]. Hence, the inheritance semantics desired in the target enterprise can be incorporated in the X-GTRBAC Admin by modeling the Admin Role hierarchy in the appropriate manner.

Independent, interoperable administrative domains: The XML documents containing user, role, and permission information from Tables 2-5 for these various domains could be composed independently of each other. The respective SSOs would have a common vocabulary available to them to express the domain-specific, yet enterprise-conformant and interoperable policies using the syntax and semantics of the X-GTRBAC specification language.

5. RELATED WORK AND DISCUSSION

There has been a growing interest in administration models built on RBAC and related schemes. One aspect of the administration models is the process of user-to-role assignment (some schemes have used the term "role activation" to include both assignment and activation in a single step). Although role assignment/activation process has been investigated in the RBAC context by the research community [10, 11], none incorporates all the features outlined in this work that are essential to enterprise-wide access control. While the role assignment scheme in [11] is based only on static attributes of a user with no support for context-dependent constraints, the one in [10] supports dynamic conditions on role activation. It, however, relies on the notion of appointment certificates to assign roles to eligible users, and does not explicitly recognize role hierarchies- a feature that discounts role relationships which are useful in various access decisions. An administration model for RBAC (ARBAC99) has been proposed in [5]. The model also uses RBAC itself for role administration within an RBAC system, and introduces the notion of an administrator role, with administrative permissions. It uses *can_assign* and *can_revoke* relations that can be interpreted to determine (i) the "role range" that an administrator role has authority over, and (ii) the "pre-requisite role" (also called pre-requisite condition) needed to exercise that authority. The conditions it specifies are static, and would not be a viable approach for a dynamically changing enterprise environment, where the administrators' authority may need to be restricted based on context conditions. Also, certain weaknesses in the model have been highlighted in [12]. The most significant of them include (i) undesired flow of permissions from a role in the "role range" to another role outside the "role range" because of role hierarchy relationships, and (ii) unrestricted assignment of permissions from the "pre-requisite roles" to the roles in "role range". An ARBAC02 model has been presented in [12] to overcome these weaknesses, and it uses the organization structure as the basis for pre-requisite conditions, instead of pre-requisite roles in a role hierarchy. Although using the organization structure as a pre-requisite condition avoids the dependencies that arise because of using role hierarchies, it still does not facilitate administration in large enterprises with context-dependent access control requirements, because a constraint expressed even in terms of organizational unit's parameters is still a static constraint. Our credential mapping mechanism captures the essence of the ARBAC02 model because it uses the attributes

specific to the organization as a criterion for role assignment. In addition, the optional constraint expressions together with the predicate grammar of X-GTRBAC can be used to specify any restrictive constraint on role assignment based on hierarchical relationships between roles, should the need arise to do so. Hence, we see our approach as providing a balance between both ARBAC99 and A-RBAC02 models.

Along another but related direction, Kern et al [13] have proposed a role-based administrative approach called A-ERBAC after their Enterprise Role-Based Access Control model. They build on their notion of "enterprise roles", which they claim are helpful in reducing the administration effort required to maintain users and their access rights in large enterprises. A-ERBAC implements the administrative security system as a component system within the ERBAC model itself. The administrators are defined as accounts in this system, and receive access rights via roles containing administrative permissions. Administrative accounts and permissions are normal ERBAC objects. They discuss "scopes" of authority for administrator accounts, which are related to the organizational structure. The model emphasizes separation of administrative domains for user-to-role and permission-to-role assignments, much like the separation of language schemas for the corresponding assignments in our X-GTRBAC framework. However, this model is also inadequate for supporting enterprise-wide access control for the same arguments as those for ARBAC02. They have augmented their work with the discussion of a commercial security administration tool implementing these concepts. Their observations regarding performance gains achieved through separation of administration and access layers in an application security system match our initial results obtained during the on-going implementation effort on our prototype system [3].

We maintain that a distinct feature of our approach is that it is suitable for generic, heterogeneous enterprise environments, with varying levels of access control requirements, because of the salient features provided by the X-GTRBAC framework. These include a semantically rich specification language that supports context-aware constraint expressions, an XML-based representation well-suited to heterogeneous, interoperable systems, and a consistent vocabulary to express access control policies. A common vocabulary further enhances reusability of the language schemas, in that the same set of schema definitions can be imported in multiple assignments, and hence significantly reduces the overhead of having to process similar constraint expressions for a typical several hundred users in a large enterprise. All the objects in our X-GTRBAC system, including roles created for administration, are treated uniformly which keeps the administrative concept simple in practice. Thus, a resulting benefit that accompanies our framework is the fact that the user-to-role assignment mechanism can also be applied to the users being assigned to administrative roles, in addition to those being assigned to regular roles. An analogous fact holds for permission-to-role assignments. These features further facilitate policy administration tasks.

In addition to the above merits, another major advantage in the realm of EC technology is the availability of widely-adopted XML-based standards for integration into external applications. The fact that ours is a "pure" XML framework would not only enhance interoperability, but would also make it a light-weight deployable component within the distributed EC network of the target enterprise.

6. CONCLUSION

In this paper, we have presented X-GTRBAC Admin, an administration model for the X-GTRBAC framework. We have elucidated the administrative concepts related to X-GTRBAC, and motivated the need for the proposed administration model. X-GTRBAC Admin achieves simplification of policy administration tasks by defining language schemas that facilitate the user-to-role and permission-to-role assignments within the enterprise. Our administration model integrates very well within our existing framework because of the modular design of the latter which emphasizes separation of language schemas for various policy specification tasks. A generic enterprise example has been provided to consolidate the ideas presented in the paper. We plan to augment our existing X-GTRBAC prototype system with the administrative extensions, and report our implementation experiences in some future work. We also intend to explore the issues related to administration of policies in multi-domain environments, and how the set of Admin Permissions would need to be extended, for instance, to allow modifications in role hierarchy, or to export a set of roles to another domain. Also of interest would be to provide consistency and availability guarantees for the system, to avoid a situation where the context constraints prevent a valid administrative authority to be assigned to or exercised by any user in the system. These challenges need to be addressed for effective administration of access control policies in a widely-distributed dynamic enterprise.

7. ACKNOWLEDGMENTS

Portions of this work have been supported by the sponsors of the Center for Education and Research in Information Assurance and Security (CERIAS) at Purdue University, and the National Science Foundation under NSF Grant# IIS-0242419.

8. REFERENCES

[1] Overview of Enterprise Computing http://faculty.washington.edu/jtenenbg/courses/455/s02/sessions/ec_overview.ppt

[2] XACML 1.0 Specification http://xml.coverpages.org/ni2003-02-11-a.html

[3] R. Bhatti, "X-GTRBAC: An XML-based Policy Specification Framework and Architecture for Enterprise-Wide Access Control", Masters thesis, Purdue University, May 2003. Available as CERIAS tech. report 2003-27.

[4] J. B. D. Joshi, Elisa Bertino, Usman Latif, Arif Ghafoor, "Generalized Temporal Role Based Access Control Model (GTRBAC)", Submitted to IEEE Transaction on Knowledge and Data Engineering. Available as CERIAS tech. report 2001-47.

[5] R. Sandhu and Q. Munawer. The ARBAC99 model for administration of roles. In Proceedings of the 15th Annual Computer Security Applications Conference, Dec 1999.

[6] D. F. Ferraiolo , R. Sandhu , S. Gavrila , D. Richard Kuhn , Ramaswamy Chandramouli, "Proposed NIST standard for role-based access control", ACM Transactions on Information and System Security (TISSEC), Volume 4 , Issue 3 (August 2001).

[7] R. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman, "Role Based Access Control Models", IEEE Computer Vol. 29, No 2, February 1996.

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

[9] J. B. D. Joshi, Elisa Bertino, Arif Ghafoor, "Temporal hierarchies and inheritance semantics for GTRBAC", In proceedings of 7th ACM Symposium on Access Control Models and Technologies, June 2002

[10] J. Bacon, K. Moody, W. Yao, "A model of OASIS role-based access control and its support for active security", ACM Transactions on Information and System Security (TISSEC) Volume 5 , Issue 4 , November 2002.

[11] M. A. Al-Kahtani, R. Sandhu, "A Model for Attribute-Based User-Role Assignment", In proceedings of 18th Annual Computer Security Applications Conference, Las Vegas, Nevada, December 2002.

[12] S. Oh, R. Sandhu, "A model for role administration using organization structure", In proceedings of the seventh ACM symposium on Access control models and technologies, June 2002

[13] A. Kern, A. Schaad, J. Moffett, "An administration concept for the enterprise role-based access control model", In proceedings of 8th ACM Symposium on Access Control Models and Technologies, June 2003