CERIAS Tech Report 2003-27

X-GTRBAC: AN XML-BASED POLICY SPECIFICATION FRAMEWORK AND ARCHITECTURE FOR ENTERPRISE-WIDE ACCESS CONTROL

by Rafae Bhatti

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

X-GTRBAC:

AN XML-BASED POLICY SPECIFICATION FRAMEWORK AND ARCHITECTURE FOR ENTERPRISE-WIDE ACCESS CONTROL

A Thesis

Submitted to the Faculty

of

Purdue University

by

Rafae A. Bhatti

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

May 2003

To my parents,
for without them I wouldn’t have accomplished this milestone.

ACKNOWLEDGMENTS

Acknowledgement is due to my advisor Prof. Arif Ghafoor, without whose support this thesis would not have seen the light of day. Also to be acknowledged are my committee members, Prof. Charlie Hu and Prof. Hong Tan, for their valuable input and encouragement. The guidance from Prof. Elisa Bertino at Università di Milano, Milano, Italy has been instrumental in the evolution of this work. Last but not the least, the research presented in this thesis has been partially supported by the National Science Foundation under the NSF Grant # IIS-0209111.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABSTRACT

Bhatti, Rafae A., M.S.E.C.E, Purdue University, May 2003. X-GTRBAC: An XML-based Policy Specification Framework and Architecture for Enterprise-Wide Access Control. Major Professor: Arif Ghafoor.

Modern day enterprises exhibit a growing trend toward adoption of enterprise computing services for efficient resource utilization, scalability and flexibility. These environments are characterized by heterogeneous, distributed computing systems exchanging enormous volumes of time-critical data with varying levels of access control in a dynamic business environment. The enterprises are thus faced with significant challenges as they endeavor to achieve their primary goals, and simultaneously ensure enterprise-wide secure interoperation among the various collaborating entities. Key among these challenges are providing effective mechanism for enforcement of enterprise policy across distributed domains, ensuring secure content-based access to enterprise resources at all user levels, and allowing the specification of temporal and non-temporal context conditions to support fine-grained dynamic access control. This thesis investigates these challenges, and presents X-GTRBAC, an XML-based GTRBAC policy specification language and its implementation for enforcing enterprise-wide access control. Our specification language is based on the GTRBAC model that incorporates the content- and context-aware dynamic access control requirements of an enterprise. An X-GTRBAC system has been implemented as a Java application. We discuss the salient features of the specification language, and present the software architecture of our system. A comprehensive example is included to discuss and motivate the applicability of the X-GTRBAC framework to a generic enterprise environment. An application level interface for implementing the policy in the X-GTRBAC system is also provided to consolidate the ideas presented in the thesis.

1. INTRODUCTION

The dynamic economic and technical environment surrounding modern day business operations requires today's enterprises to strive ever-more to stay competitive in the marketplace [1]. The fast paced advancements in information technology infrastructure has placed growing demands on the enterprise to rise above the competition by making its operations more expandable, responsive to external factors, and, on top of all, secure. This has motivated many enterprises to adopt enterprise computing (EC) services for efficient resource utilization, scalability and flexibility [2]. Such enterprises are also known as Computer Integrated Enterprise (CIE). The CIE is characterized by heterogeneous, distributed computing systems exchanging enormous volumes of time-critical data with varying levels of access control through the use of EC technology. The EC framework highlights the durability and maintainability of the assets of the enterprise, and emphasizes upon the scalability and flexibility of system infrastructure to allow it to evolve with time. While adopting such strategies is vital to the success of enterprise's business operations, it poses some serious challenges in terms of ensuring an enterprise-wide secure interoperation among the various collaborating entities.

These challenges belong to various domains within the CIE, and each needs to be addressed in order to achieve the overall enterprise goals. The access control policy of a large enterprise has many elements and many points of enforcement [3]. Elements of policy may be managed by the Information Systems department, Human Resources, the Legal department and the Finance department. And the policy may be enforced by the extranet, mail, WAN and remote-access systems; platforms which inherently implement a permissive security policy. For these reasons, there is a pressing need for a common language for expressing enterprise-wide policy. Additionally, since most enterprises are still migrating from legacy systems to newer infrastructure, the uniformity among the communication protocols across the heterogeneous systems is essential to timely and

accurate execution of enterprise-level policies. Another concern faced by large enterprise applications is the number of users or clients accessing the enterprise resources, which runs in tens of thousands. Since these resources would typically have privilege levels associated with them, a mechanism needs to be provided to allow only authorized users to access the requested resource. Hence, exercising content-based access control is a significant challenge in securing large enterprises. Yet another dimension that complicates access control specification is the dynamic nature of context-based conditions attached with access decisions. An enterprise may grant/revoke access to resources to certain individuals based on their level of involvement in the current stage of product life cycle. The temporal constraints are also extended to sharing of information between various users based on their degree of relevance to the resource at a particular time. To adequately satisfy these kinds of conditions in a dynamic environment, hence, constitutes another significant challenge. The primary goal of this thesis is to investigate these challenges and propose an XML-based access control specification language that adequately addresses them.

Our specification language is based on Generalized Temporal Role Based Access Control (GTRBAC) model [4]. It is a generalized temporal extension of the widely accepted Role Based Access Control (RBAC) model proposed in the NIST RBAC standard [5]. RBAC model, because of its generality, can be used for defining a diverse set of access control policies. Another advantage of the RBAC model is that it simplifies authorization administration in large enterprises. RBAC models have also been shown to be policy-neutral [6], and can be used to represent a variety of security policies, including both DAC and MAC policies [7]. 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 (See Section 1.1). GTRBAC extends RBAC to allow a generalized mechanism to express a diverse set of fine-grained temporal constraints related to both roles and permissions in order to meet the dynamic content-based context-aware access control requirements of an enterprise. The X-GTRBAC framework presented in this work outlines an XML-based specification language that focuses on encapsulating all the basic

features set forth in the GTRBAC model. Our framework augments the GTRBAC model with XML to allow for supporting the policy enforcement in a heterogeneous, distributed environment. It is based on XML schemas, and not DTDs, and hence has more expressive power and support for data types as compared to DTD-based approaches [8].

1.1 Related Work

The specification of security policies for enterprises has recently emerged as an active research area. The advent of the RBAC model has generally been hailed as a promising sign in the industry and research community for its potential to simplify authorization administration in large enterprises. Related work has broadly spanned the aspects of both presenting system architecture and implementation of RBAC-based technologies for enterprise security administration, and complementing the RBAC model to introduce extended policy specification frameworks. In the following, we summarize the efforts in both these directions, and then highlight the significance of our particular work.

Several approaches have been presented in the literature to address various aspects of security administration within an enterprise. An XML based approach to specify enterprise RBAC policies has been reported in [9]. Ferraiolo et. al. use the RBAC model to address access control needs of enterprise computing environments [10]. They have attempted to present an RBAC-based approach as an alternative to ACL-based access control scheme used within a corporate intranet, and have illustrated its use through a reference implementation. Kern et. al. present an ERBAC model for Enterprise-Wide Role-Based Access Control [1]. Their model uses the notion of enterprise roles with parameterization, and is reported by the authors as being helpful in reducing the administration effort required to maintain users and their access rights in large enterprises. They have augmented their work with the discussion of a commercial security administration tool. All these schemes, however, are not suited to enterprises with dynamic content- and context-aware access control requirements since they provide no explicit mechanism to support evaluation of dynamic user credentials and context conditions.

There has been an effort in the research community to complement the RBAC model with additional features to allow extended policy specification frameworks. A temporal extension to RBAC has been presented in the TRBAC model [11]. TRBAC supports the specification of temporal constraints on role enabling, and the dependencies among them, and hence provides a mechanism to enforce time-dependent access control. A generalized GTRBAC model presented in [4] is capable of providing a wider range of temporal constraints, including periodic as well as duration constraints on user-to-role assignments, permission-to-role assignments, and role activation. The GTRBAC model extends the syntactic structure of TRBAC; however, the notion of user credentials is not supported in both these schemes to allow the assignment of authenticated users to a particular role.

Two closely related works have been reported in the literature by the industrial community. The OASIS XACML [3] specification is based on an extension of XML to define an access control specification that supports notions similar to those of user credentials and context-based privilege assignments. It, however, does not directly support the notion of roles, and hence lacks the essential features as separation of duty constraints, role hierarchy, and cardinality. The absence of roles also prohibits the provision of a comprehensive mechanism to supply and evaluate sophisticated temporal constraints on assignments of users to privileged tasks, since direct user-to-permission assignments violate the very principles of scalability and manageability that motivate the use of GTRBAC (See Section 2.2). The OASIS model for active security presented in [12] addresses the context-aware access control requirements within large scale systems. It is an extension of the RBAC model with parameters based on first order logic, and allows fine-grained evaluation of dynamic user credentials and context conditions. The paper emphasizes on the formal logic-based semantics of the model with its own merits, but it does not detail any implementation architecture to enforce the same. The implementation, however, is related by the authors to a middleware architecture that supports asynchronous events, and requires a mechanism that allows the communicating systems to acquire support for asynchronous operations. Although this scheme is designed to be scalable and manageable for distributed environments, the fact that it

relies on extending the client's capabilities to make them able to communicate with the OASIS server adds significant overhead to its wide-scale deployment. Since our framework is entirely XML-based, our approach allows for adopting the XML-based middleware architecture [13] that is emerging as a widely accepted standard for communication among distributed applications, and thus significantly reduces the burden attached thereto. Our system hence captures the combined semantics of both the OASIS models, and achieves secure enterprise-wide inter-operation with dynamic fine-grained access control. We, therefore, maintain that the XML-based GTRBAC approach to address enterprise-wide access control presented in this thesis holds its novelty among the related schemes.

1.2 Organization of the Thesis

The thesis is organized as follows. Chapter 2 provides the motivation for adopting the component technologies, namely XML and GTRBAC, and gives an introduction to each one. It also discusses the RBAC model and its extensions as a step toward introducing GTRBAC. The detailed X-GTRBAC specification language is outlined in Chapter 3. This chapter also includes a discussion of the implementation architecture and system design for X-GTRABC. Chapter 4 discusses a comprehensive example of a generic CIE to illustrate the applicability of the X-GTRBAC framework. It provides the specifications for an enterprise access control policy, and a mapping thereof to our framework. Chapter 5 presents the conclusion of this work.

2. XML-BASED GTRBAC MODEL: MOTIVATION AND SPECIFICATION

This chapter provides the motivation for adopting the component technologies of our X-GTRBAC framework, namely XML and GTRBAC, and then goes on to introduce the salient features of each one. It also discusses the RBAC model and its extensions as a step toward introducing GTRBAC. Additionally, it further investigates, as part of the motivation, the access control challenges discussed in Chapter 1.

2.1 XML

The following two sub-sections provide the motivation and introduction, respectively, of the eXtensible Markup Language (XML).

2.1.1 Motivation

The use of XML is primarily motivated by the vast heterogeneity of collaborating entities exhibited by the distributed enterprise environment. The various functional units within an enterprise, connected through multiple media, and each comprising of several computing systems ranging from old to new, are linked together by the EC technology. Hence the need arises for a common language to efficiently express and execute the enterprise access control policy. 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. XML is, therefore, a natural choice as the basis for the common policy language, due to the ease with which its syntax and semantics can be extended to accommodate the unique requirements of an enterprise, and the widespread support that it enjoys from all the main platform and tool vendors [3].

2.1.2 Introduction

The eXtensible Markup Language (XML) [14] evolved from a simple subset of SGML [15], and is now hailed as the most promising technology for information interchange across heterogeneous, distributed domains [16]. XML is a meta-language that

lets users design their own markup language, and hence allows them to define an agreed-upon vocabulary for application-specific tasks. XML achieves this by offering an extensible set of markup tags to create custom documents, and a set of related technologies for their interpretation.

```
<enterprise>
 <depts>
  <engineering>
    <engg_manager job_id= "EM">
         <name>John</name>
         <level>5</level>
         <…> .. </..>
    </engg_manager>
    <product_engineer job_id="PE">
         <name>Paul</name>
         <shift>1</shift>
         <…> .. </..>
    </product_engineer>
  </engineering>
  <design>
    <design_manager job_id= "DM">
         <name>Adams</name>
         <grade>A</grade>
         <…> .. </..>
    </design_manager>
    <design_engineer job_id="DE">
         <name>Charlie</name>
         <skill>auto</skill>
         <…> .. </..>
    </design_engineer>
  </design>
 </depts>
</enterprise>
```

(a)

```
<xs:schema>
 <xs:element name ="enterprise">
  <xs:complexType>
   <xs:element name = "depts">
    <xs:complexType>
     <xs:element name = "engineering">
      <xs:complexType>
       <xs:element name = "engg_manager">
        <xs:complexType>
         <xs:attribute name ="job_id" type="xs:string"/>
         <xs:element name = "name" type="xs:string"/>
         <xs:element name = "level" type="xs:string"/>
          …….
        <xs:/complexType>
       <xs:/element>
       <xs:element name = "product_engineer">
        <xs:complexType>
         <xs:attribute name ="job_id" type="xs:string"/>
         <xs:element name = "name" type="xs:string"/>
         <xs:element name = "shift" type="xs:string"/>
          …….
        <xs:/complexType>
       <xs:/element>
      <xs:/complexType>
     <xs:/element>
     ……
  <xs:/complexType>
 <xs:/element>
<xs:schema>
```

(b)

Figure 2.1: (a) An XML instance document, and (b) its schema

Each XML document has both a logical and a physical structure [17]. Physically, the document is composed of units called entities. An entity may refer to other entities to cause their inclusion in the document. A document begins in a "root" or document entity. Logically, the document is composed of declarations, elements, comments, character references, and processing instructions, all of which are indicated in the document by explicit markup. Additionally, elements may contain attributes as well. The structure of the XML document is expressed through an XML schema [18]. A schema itself is an XML document that defines the valid syntax of an XML instance document, where the term instance document denotes a document conforming to the associated schema. Figure 2.1 illustrates an XML instance document and its corresponding schema. Here, the

structure of the various XML tags in the instance document is governed by the schema definition. For instance, the second line in the schema definition declares "enterprise" as the root element of the document. The "depts" element is then added to the root as a child element. The hierarchy is similarly extended to incorporate all the desired elements. The "engg_manager" and "design_manager" elements have a "job_id" attribute used to identify the particular job. Also note that these two elements, and others that may be part of the document, define their own set of child elements. The choice of the tags depends upon their relevance to the actual element in a particular application. This extensible naming mechanism hence allows the creation of customized documents that capture the application-specific needs of any enterprise. For interested readers, the detailed specifications of XML and XML Schemas can be found at [14, 18].

2.2 GTRBAC Model

The following two sub-sections provide the motivation and introduction, respectively, of the Generalized Temporal Role-Based Access Control (GTRBAC) model.

2.2.1 Motivation

The motivation behind adopting the GTRBAC model for enforcing enterprise-wide access control is to incorporate within the access control model the following set of capabilities:

***Content-based context-aware access*:** Information access within an enterprise may need to be restricted based on the information content and the contextual information obtained at the time the access requests are made. For example, an external client may not be allowed to access the Product Design manual from the enterprise document repository. Or it may be the case that only the authorized component technicians are allowed to access the plant inventory for a manufacturing enterprise, and such access would be restricted to the components related to the technician's job function. Hence the need arises to exercise content-based access control for all users accessing enterprise resources. The access control model should also capture security-relevant environmental context and incorporate it in its access control decisions. Access requests may be decided based on several context parameters, such as time or location. An example of location

parameter is user domains, which are classified by IP addresses. In the manufacturing enterprise, the access control model could allow all users who submit an access request from within the Design Department intranet to access the Product Design manual at any time for easy reference. This access may, however, be limited to be read-only for certain less privileged users. The time parameter needs to be incorporated in the model to express the time-dependent access constraints within an enterprise. For instance, in addition to the restriction that a particular component technician should be granted access to only the relevant components of the product, it is quite likely that such access is further restricted to only the times when the product is in the manufacturing stage. Another view of time-dependent constraints captures the periodic nature and associated duration of enterprise tasks. One example for such constraint could be a rule that the Vendor Contracts may only be allowed to be accessed by vendors in the second week of every quarter of every year, and need to be submitted within two weeks of that time. More complicated context conditions allow for expressing even more sophisticated constraints. As an example, it may be required for the manufacturing enterprise that the Product Engineering work should only start after the Product Design work has been completed. This requires for checking various context parameters to evaluate the stated condition, and allow for the requested or scheduled task to be executed.

Heterogeneity of subjects and objects: For a generic enterprise, heterogeneity implies the diversity of users and resources across the component systems making up the enterprise. Object heterogeneity may exist in the form of different types of enterprise resources that need to be protected. The resources can range from Purchase and Marketing Contracts, to Design and Engineering Manuals, to various Product Assemblies and Components. Furthermore, the information content therein can evolve with time as new resources are added and old ones removed or updated, introducing scalability problems in privilege management. Subject heterogeneity implies that users have diverse activity profiles, characteristics and/or qualifications that may not be known a-priori. Such activity profile is needed by the EC technology to dynamically confer privileges to authenticated users by upgrading their current role. This is needed because the trust level of the user may be elevated, as the product goes through the various stages within the enterprise, and access

to more privileged resources may accordingly be allowed. For instance, a Product Technician with sufficient experience may be elevated to the role of Product Supervisor, and allowed to access the Product Assemblies component at the time of manufacturing of the product assembly. Subject heterogeneity complicates access control specification.

2.2.2 Introduction

As mentioned in Chapter 1, GTRBAC is a generalized temporal extension of the RBAC model. Before introducing GTRBAC, the RBAC model is, therefore, introduced.

2.2.2.1 RBAC model

The RBAC model as proposed in the NIST RBAC standard consists of the following four basic components: a set of users `Users`, a set of roles `Roles`, a set of permissions `Permissions`, and a set of sessions `Sessions`. A user is a human being or an autonomous agent. A role is a collection of permissions needed to perform a certain job function within an organization. A permission is an access mode that can be exercised on objects in the system, and a session relates a user to possibly many roles. When a user logs in the system, he/she establishes a session and, during the session, can request to activate some subset of roles he/she is authorized to assume. An activation request is granted only if the corresponding role is enabled at the time of the request and the user is entitled to activate the role at that time. If the activation request is satisfied, the user issuing the request obtains all the permissions associated with the role he/she has requested to activate. On the sets `Users`, `Roles`, `Permissions`, and `Sessions`, several functions are defined. The user-to-role assignment (*UA*) and the permission-to-role assignment (*PA*) functions model the assignment of users to roles and the assignment of permissions to roles respectively. A user can be a member of many roles and a role can have many members. Moreover, a role can have many permissions and the same permissions can be associated with many roles. The user function maps each session to a single user, whereas function role establishes a mapping between a session and a set of roles (that is, the roles which are activated by the corresponding user in the session). On `Roles`, a hierarchy is defined, denoted by $\geq$. If $r_i \geq r_j$, r_i, $r_j \in$ `Roles` then r_i inherits the permissions of r_j. In such a case, r_i is a senior role and r_j a junior role. The following definition formalizes the above discussion:

Definition 2.2.2.1 (**RBAC model**): [5] The RBAC model consists of the following components:

- Sets `Users`, `Roles`, `Permissions` and `Sessions` representing the set of users, roles, permissions, and sessions, respectively;
- *PA*: `Roles` → `Permissions`, the permission assignment function, that assigns permissions to roles;
- *UA*: `Users` → `Roles`, the user assignment function, that assigns users to roles;
- *user*: `Sessions` → `Users`, which maps each session to a single user;
- *role*: `Sessions` → 2^{Roles} that maps each session to a set of roles;
- $RH \subseteq$ `Roles` × `Roles`, a partially ordered role hierarchy (written ≥).

The RBAC model differentiates itself from traditional access control models in that the permissions in RBAC are not directly associated with users, but with roles. Roles are created by the security administrators to reflect the various functional categories of users within the enterprise. Users are then assigned membership to roles, and these roles are in turn assigned permissions. Permissions are actually composed of an object-to-operations mapping. The element relationships of Core RBAC model are illustrated in Figure 2.2.

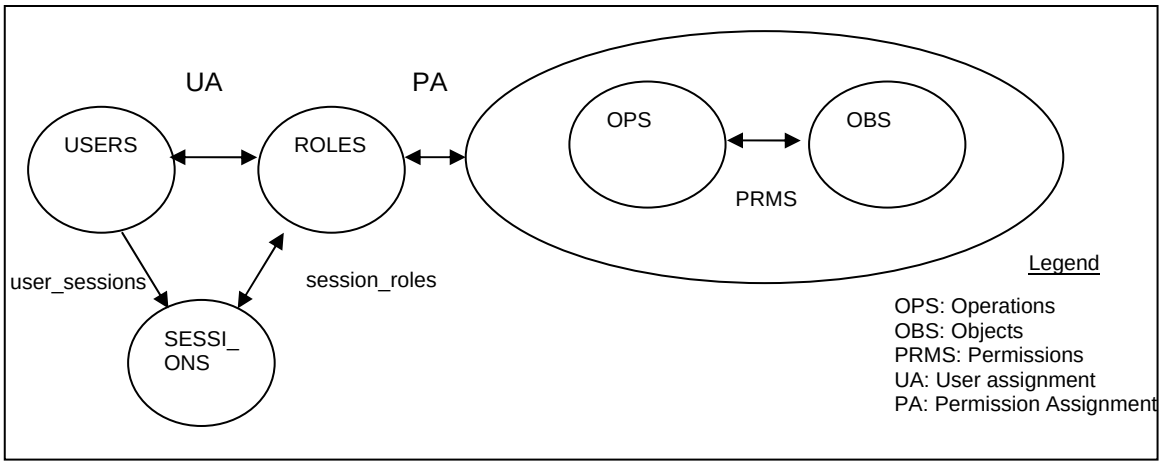

Figure 2.2: Core RBAC elements and their relation (Source: NIST RBAC standard [5]).

RBAC approach naturally fits into an organizational context as users are assigned to organizational roles that have well-defined responsibilities and privileges. The RBAC model proposed by NIST, because of its generality, can be used to express a very wide range of security policies including discretionary and mandatory, as well as user-defined organizational specific policies [6, 7, 19, 20]. Many benefits of an RBAC approach include its support for security management and the principle of least privilege [6]. For example, we can easily manage a change in a user's responsibility or role within the organization by assigning him/her the new role and removing him/her from the old one. Furthermore, use of role hierarchies and grouping of objects into object classes based on responsibility associated with a role makes the management of permissions very easy. By configuring the assignment of the least set of privileges from a role set assigned to a user when he/she activates the role, inadvertent damage can be minimized in a system.

2.2.2.2 Temporal extensions to RBAC

Because of its relevance and above-mentioned benefits that it provides, the RBAC model has been widely investigated and several extensions to it have been proposed. A set of such extensions related to the temporal dimension of the model shall now be discussed. An initial temporal extension to RBAC has been proposed in the Temporal RBAC (TRBAC) model. This has been motivated by the fact that in many organizations, functions may have limited or periodic temporal duration. Consider, for instance, the case of a component technician in a manufacturing enterprise and assume that any technician is to be authorized to work only when a supervisor is at work. If both the technician and supervisor are represented as roles, enforcing such a requirement entails constraining the enabling of a technician role only during the specified temporal interval when the supervisor role is enabled. TRBAC allows the specification of such temporal conditions. The main features it provides include the periodic enabling/disabling of roles and temporal dependencies among them expressed by means of role triggers, which are active rules that are automatically executed based on the enabling and/or disabling of roles. Priorities are associated with both the triggers and periodic enabling/disabling of roles to handle possible conflicts that can arise, when the simultaneous enabling/disabling of a role is required. In such cases, a combination of priority and denials-take-precedence rule

are used to resolve the conflicts. TRBAC further allows an administrator to issue run-time requests for enabling and disabling a role and restricted handling of role activations by a user. TRBAC, however, is inadequate to express a variety of useful temporal constraints. In particular, TRBAC does not include temporal constraints on (i) user-to-role and permission-to-role assignments, and (ii) activations of roles by users. In (i), TRBAC assumes that only roles can be transient, i.e., only they are enabled and disabled at different time intervals. In this thesis, we motivate the point that in a typical enterprise environment, roles, as well as users and permissions assigned to them, may also be transient. Because of (ii), TRBAC does not use a well-defined, separate notion of role enabling and role activation, and hence cannot enforce a fine-grained access control at the user level for role activation. The GTRBAC model distinguishes between the notions of role activation from that of role enabling to incorporate various activation constraints on role activations at the individual user level. It also extends the temporal constraint enforcement mechanism to user-to-role and permission-to-role assignments. It thus allows the specification of a more complete set of temporal constraints related not only to role enabling, but also to user-to-role assignment, permission-to-role assignment, and role activation. The notions of triggers and safety, as provided in TRBAC, are also accordingly extended to capture the enhanced language semantics. The X-GTRBAC framework hence builds upon the elaborate and consistent set of specifications laid out in the GTRBAC model.

2.2.2.3 GTRBAC specifications

The GTRBAC model allows the specification of an elaborate set of temporal constraints on role enabling/disabling, activation/deactivation, and user-to-role and permission-to-role assignment/de-assignment. These constraints are composed of periodic-time expressions that capture the valid periods of time when the corresponding constraint may be satisfied. This expression has three parts: (i) a start-time expression, which indicates a valid start time, (ii) an interval expression, which indicates the interval within which the constraint may be satisfied, and (iii) a duration expression, which indicates the valid duration for the constraint. Note that all parts of the periodic-time expression are optional, and the absence of any one of them means that there is no

corresponding time restriction. The model additionally allows various content and context conditions to be specified. These conditions are captured through status expressions for roles, assignments, or other environmental parameters. Specifically, the following temporal constraints and events are allowed in GTRBAC:

Temporal constraints on:

(a) ***role-enabling:*** These constraints allow the specification of the valid time periods during which a role is enabled/disabled. Additional content-and context conditions may also be supplied within the constraint.

(b) ***role-activation:*** These constraints allow the specification of the valid time periods during which a role is activated/deactivated. Additional content-and context conditions may also be supplied within the constraint. Role activation only occurs as a run-time event.

(c) ***user-to-role assignment:*** These constraints allow the specification of the valid time periods during which a user may be assigned to a role. Additional content-and context conditions may also be supplied within the constraint.

(d) ***permission-to-role assignment:*** These constraints allow the specification of the valid time periods during which a permission may be assigned to a role. Additional content-and context conditions may also be supplied within the constraint.

Run-time events: A set of run-time events allows a user or an administrator to dynamically initiate actions. One such example captured in our framework is activation/deactivation of a role. Note that activation requests for a role may only be supplied by the user, since role activation is done at the user's discretion.

Triggers: Triggers allow for expressing dependency among GTRBAC events, as well as capturing the past events and defining future events based on them. Triggers may not include any activation event in their head, for the reason cited above.

In addition to the above temporal constraints, the GTRBAC model supports the following separation of duty constraints as per the NIST RBAC standard:

(a) ***Static separation of duty:*** The semantics of static separation of duty require that no `n` roles that are part of a "Static Separation of Duty Role Set" (SSD_Role_Set) be assigned to the same user, where `n` is any positive integer.

(b) ***Dynamic separation of duty:*** The semantics of dynamic separation of duty require that no `m` roles that are part of a "Dynamic Separation of Duty Role Set" (DSD_Role_Set) be simultaneously active in the same session of the same user, where `m` is any positive integer.

The formal expressions for the specification of periodic time and the temporal constraints and events in GTRBAC are re-produced below for completeness.

Definition 2.2.3.1 (**Periodic expression**) [21]: Given calendars C_d, C_1, …, C_n, and time occurrences $O_{1, ..., } O_n$, a periodic expression `P` is defined as:

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

where $O_1 = \mathtt{all}$, $O_i \in 2^N \cup \{\mathtt{all}\}$, $C_i \sqsubseteq C_{i-1}$ for i = 2,.., n, $C_d = C_n$, and $x \in \mathbb{N}$.

Periodic-time expression: Periodic time expression 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 denoting the lower and upper bounds that are imposed on instants in `P`.

The formalism for periodic expressions is based on the one used in [22], and relies on the notion of calendars. A calendar is defined as a countable set of contiguous intervals[1], numbered by integers called indexes of the intervals. In our discussion, we assume the existence of a set of calendars containing the calendars *Days*, *Weeks*, *Months*, and *Years*, where *Days* is the calendar with the finest granularity, i.e., it is the basic calendar. 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 of each interval in terms of calendar C_d. 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, O_i is omitted

[1] Two intervals are contiguous if they can be collapsed into a single one (e.g., [1, 2] and [3, 4]).

when its value is `all`, whereas it is represented by its unique element when it is a singleton. $x.C_d$ is omitted when it is equal to $1.C_n$.

Event expression: A simple event expression has one of the following forms:

(a). `enable` *r* or `disable` *r* where *r* ∈ `Roles`.

(b). `assign` *r* `to` *u* or `de-assign` *r* `to` *u*, where *r* ∈ `Roles` and *u* ∈ `Users`.

(c). `assign` *p* `to` *r* or `de-assign` p `to` r, where *p* ∈ `Permissions` and *r* ∈ `Roles`.

Role status expression: Role status expressions have one of the following forms:

(a). `enabled` *r* or ¬`enabled` *r* (or `disabled` *r*), where *r* ∈ `Roles`.

(b). `activated` *r*, where *r* ∈ `Roles`.

(c). `active` *r* `for` *u* or ¬`active` *r* `for` *u*, where *r* ∈ `Roles` and *u* ∈ `Users`.

Assignment status expression: Assignment status expressions have the following forms

(a). `assigned` *r* `to` *u* or ¬`assigned` *r* `to` *u*, where *r* ∈ `Roles` and *u* ∈ `Users`.

(b). `assigned` *p* `to` *r* or ¬`assigned` *p* `to` *r*, where *r* ∈ `Roles` and *p* ∈ `Permissions`.

Run-time request: A run-time request expression has one of the following forms:

(a). a user's run-time request expression to activate a role has the form:

s: `activate` *r* `for` *u* `after` Δt, or

s: `deactivate` *r* `for` *u* `after` Δt

where *r* ∈ `Roles` and *u* ∈ `Users`, *s* is the session attached to the request, and Δt is the duration.

(b). an administrator's run-time request expression has the form:

E `after` Δt

where *E* is an event expression and Δt is the duration expression.

Triggers: A trigger expression has the form

$E_1, \ldots, E_n, C_1, \ldots, C_k \rightarrow E$ `after` Δt

where E_is are event expressions or run time requests, C_is are role status expressions or assignment status expressions, *E* is an event expression such that $E \notin$ {`s:activate r for u`}, and Δt is a duration expression.

Table 2.1 summarizes the temporal constraint types and expressions of the GTRBAC model. The GTRBAC model extends the safety notion of the TRBAC model to show that there exists an execution model for it. Hence the consistency of our framework is implied by the consistency of GTRBAC model.

Table 2.1
Temporal constraints and event expressions in GTRBAC

Constraint categories	**Events**	**Expression**
Enabling	Role enabling	(I, P,D, `enable/disable` *r*)
Activation	Role activation	<!--only occurs as a run-time event -->
Assignment Constraint	User-to-role assignment	([I, P, D], $\texttt{assign}_U$/$\texttt{deassign}_U$ *r* `to` *u*)
	Permission-to-role assignment	([I, P, D], $\texttt{assign}_P$/$\texttt{deassign}_P$ *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	($\texttt{assign}_U$/`de-assign`$_U$ *r* `to` *u* `after` Δt)
		(`enable/disable` *r* `after` Δt)
		($\texttt{assign}_P$/`de-assign`$_P$ *p* `to` *r* `after` Δt)
		(`enable/disable` *c* `after` Δt)

This chapter motivated and introduced the component technologies of the X-GTRBAC framework. In the next chapter, we present its specification language and system architecture.

3. X-GTRBAC SPECIFICATION LANGUAGE AND ARCHITECTURE

This chapter presents the specification language and system architecture for our X-GTBRAC framework. The specification is designed to serve dual goals. One, it attempts to model the RBAC elements and incorporate the functional specifications as per the NIST RBAC standard [5]. Additionally, it provides the syntactic and semantic constructs needed to enforce the temporal constraints as per the GTRBAC model. The system architecture is designed using XML and Java based technologies.

3.1 Modeling RBAC Elements

Initial goal of the X-GTRBAC specification language is to model the five basic RBAC elements (as shown in Figure 2.2) and their associated set-relations. To represent the RBAC elements in XML, we generate schema definitions for “user”, “role”, and “permission”. Note that schema definition is not necessary for “operation” and “object” elements because they are included in a “permission” definition as per the RBAC standard, and hence their relationship with other RBAC elements is captured in the “permission” schema. We introduce a BNF-like grammar, called X-Grammar, to present an overview of the specification language for the RBAC elements in an XML-syntax.

X-Grammar: The X-Grammar follows the same notion of terminals and non-terminals as in BNF, but supports the tagging notation of XML that also allows expressing attributes within element tags. The non-terminals are expressed as <!--“non_terminal_name”> XML tags, and terminals as standard XML tags. Optional tags are placed within square brackets “[ ]”. Group portions of a production are included in curly brackets “{}”, with the repeat count indicated by a subscript. The default count is one. A “*” and a “+” indicates a count of “zero or more” and “one or more” respectively, whereas a “-” is used to provide a range. A “|” indicates alternates within a production set, and exactly one can be chosen. Any data placed in parenthesis “( )” is not part of the terminal symbol, and

shall be supplied by the security administrator. The X-Grammar has been adopted for a clear expression of the specification language constructs. The corresponding schemas for the XML sheets listed in the thesis are provided in Appendix–A.

3.1.1 Users and credentials

To evaluate the users being assigned to a particular role, the specification language uses the notion of credentials as discussed in [23]. A "credential type" is created by the security administrator to group users based on their credentials, and hence enforcing a common set of attribute-value pairs for a given group. This set of attributes constitutes the "cred_expr" for the given credential type. A credential type definition schema (XCredTypeDef) is supplied as part of the specification language to facilitate the creation of new credential types.

<!-- Definitions of Credential Types> ::= <XCredType [xctd_id = (id)] > {<!-- Credential Type Definition>}+ </XCredType>	<!-- Credential Type Definition> ::= <CredType cred_type_id = (id) type_name= (type name) > <AttributeList> {<!-- Attribute Definition>}+ </AttributeList> </CredType >	<!-- Attribute Definition> ::= <Attribute> <AttributeName usage = "mand | opt" type = (type)> (name) </AttributeName >

Figure 3.1: X-Grammar for XCredTypeDef

With respect to the grammar for the XCredTypeDef sheet shown in Figure 3.1, mand indicates that the attribute is mandatory whereas opt indicates that it is optional. The credential information in XCredTypeDef sheet provides a vocabulary to express the credentials needed by the users of any organization in order for them to be considered for assignment to specific roles. Users and their credentials are expressed in the form of an XML document that we refer to as XML User Sheet (XUS). The grammar for XUS is

<!-- User Definitions >::= <Users> {<!-- User Definition>}+ </Users>	<!-- User Definition> ::= <User user_id = (id)> <UserName> (name) </UserName> {<!--CredType>}+ <MaxRoles>(number)</MaxRoles> </User>
<!—CredType > ::= <CredType cred_type_id = (id) type_name= (type name) > <!-- Credential Expression> </CredType>	<!-- Credential Expression> ::= <CredExpr> {<(attribute name)> (attribute value) </(attribute name)>}+ </CredExpr>

Figure 3.2: X-Grammar for XUS

shown in Figure 3.2. The “max_roles” tag indicates the maximum number of roles that a user can be assigned to. As shall be elaborated in the next section, user credentials may be updated dynamically to capture the activity profile of the user.

3.1.2 Roles

Roles are also created by the security administrator. A role has an associated set of credentials that must be satisfied by the users who are assigned to that role. Roles and their associated information is expressed in the form of an XML document that we refer to as XML Role Sheet (XRS). The grammar for XRS is shown in Figure 3.3.

<!-- XML Role Sheet> ::= <XRS [xrs_id = (id)]> {<!-- Role Definition>}+ </XRS>	<!-- Role Definition> ::= <Role role_id = (id) role_name = (role name)> [<!--{En|Dis}abling Constraint>] [<!--[De]Activation Constraint>] {<SSDRoleSetID> (id) </SSDRoleSetID>}* {<DSDRoleSetID> (id) </DSDRoleSetID>}* [<Junior> (name) </Junior>] [<Senior> (name) </Senior>] [<Cardinality> (number) </Cardinality>] </Role>

Figure 3.3: X-Grammar for XRS

The “role_name” is a unique role identifier. The cardinality of a role is the maximum number of users assigned to it at any time. If none is explicitly supplied, it is assumed unlimited.

3.1.2.1 Separation of duty constraints

Each role definition contains optional “SSD_Role_Set_id” and “DSD_Role_Set_id” tags which refer to the set of roles that are collectively in static and dynamic separation of duty respectively as per the NIST RBAC standard. Each of SSD_Role_Set and DSD_Role_Set has a cardinality attribute that gives the maximum number of roles that a user may be assigned to or can simultaneously have active in his/her sessions from the set. The SSD_Role_Sets and DSD_Role_Sets are supplied in a separate XSoDDef sheet. The grammar for XSoDDef sheet is shown in Figure 3.4.

<!-- Separation of Duty Definitions> ::=
<XSoDDef [xsod_id = (id)] >
[<!--SDRoleSets>]
[<!--DSDRoleSets>]
</XSoDDef>

<!-- SSDRoleSets > ::=
<SSDRoleSets>
{<!--SSDRoleSet>}+
</SSDRoleSets>

<!-- DSSDRoleSets > ::=
<DSDRoleSets>
{<!--DSDRoleSet>}+
</DSDRoleSets>

<!-- SSDRoleSet> ::=
<SSDRoleSet>
{<SSDRole ssd_role_set_id =(id)
ssd_cardinality = (number)>
(role name)
</SSDRole>}+
</SSDRoleSet>

<!-- DSDRoleSet> ::=
<DSD dsd_role_set_id =(id)
dsd_cardinality = (number)>
(role name)
</DSDRole>}+
</DSDRoleSet>

Figure 3.4: X-Grammar for XSoDDef sheet

3.1.2.2 Role hierarchies

The optional "junior" and "senior" tags referring to junior and senior roles are used to capture hierarchical relationships in the RBAC model. The exact semantics that should be enforced on roles within an hierarchy are determined by the target enterprise. However, the notion of "authorized roles" and "authorized permissions" as stated in the NIST RBAC standard is supported by our framework. "Authorized roles" refers to the set of assignable roles for a user, including the roles that are directly assignable to the user based on his/her own credentials, as well as the "junior" roles of all such roles. "Authorized permissions" is the set of corresponding permissions for the authorized roles.

3.1.2.3 Temporal and non-temporal context-based constraints

We now discuss the tags of a role that capture the semantics of both temporal and non-temporal constraint specification and related information. All these tags are optional since their omission simply implies the absence of constraints in any given specification. This set of constraints is supplied in a separate XTempConstDef sheet. The grammar for XRS constraints specification is shown in Figure 3.5. The grammar for corresponding XTempConstDef sheet is shown in Figure 3.6.

The "Attributes" tag of the role contains 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. This role parameterization facilitates in capturing the status expressions in GTRBAC model as discussed in Chapter 2. The

context conditions may be based on parameters such as time, system load, etc., or on status expressions such as "whether role R has been enabled by user U". The "(En|Dis)abling Constraint" and "[De]Activation Constraint" tags contain a set of conditions, where each condition is composed of possibly multiple logical expressions for specification of the respective constraints based on both temporal and non-temporal context-dependent parameters. The constraint tag has an optional op-code attribute that determines the evaluation logic of the expressions within the constraint. An op-code of (i) "AND" implies that all constituent expressions must be true for the constraint to be true, (ii) "OR" implies that at least one expression must be true for the constraint to be true, and (iii) "NOT" implies that none of the expressions must be true for the constraint to be true. The op-code defaults to "AND" if none is specified.

Each condition tag may contain a "pt_expr_id" or "d_expr_id" attribute that refers to a periodic-time or a duration expression respectively. These expressions are the XML representation of the periodic-time expression framework provided in the GTRBAC model, and bind the corresponding condition with the respective periodic expression. We give an XML representation for each of the start-time, interval, and duration expressions that together constitute the periodic-time expression. Following the notion of "calendars" used in the GTRBAC model, the start time expression consists of "calendar sets", where each calendar is a unit of time, e.g. years, months, weeks, etc. As an example, an event that occurs at the start of the second week of every first and eighth month of every odd year would be represented by using "{odd}" as the Year set, "{1,8}" as the Month Set, and "{2}" as the Week Set. The optional "pt_id_ref" attribute indicates start time with

<!--{En|Dis}abling Constraint> ::= <{En|Dis}abConstraint [op = {AND|OR|NOT}] > {<!--{En|Dis}abling Condition>}+ </{En|Dis}abConstraint>	<!--[De]Activation Constraint> ::= <[De]ActivConstraint [op ={AND|OR|NOT}]> {<!--[De]ActivationCondition>}+ </[De]ActivConstraint>
<!--{En|Dis}abling Condition> ::= <{En|Dis}abCondition [{pt_expr_id=(id) | d_expr_id=(id)}] > [<!-- Logical Expression>] <{En|Dis}abCondition>	<!--[De]Activation Condition> ::= <[De]ActivCondition [d_expr_id=(id)] > [<!-- Logical Expression>] </[De]ActivCondition >

Figure 3.5: X-Grammar for XRS constraints

reference to the provided periodic-time expression id. If it is supplied, then the start time is the same as that of the referenced periodic time. Note that a "pt_id_ref" is provided only when the calendar sets are not provided, and vice versa. Any new start time is always explicitly defined using new calendar sets. An interval is given by a (begin_date, end_date) pair, and a duration is specified as (calendar, calendar_length) pair. The semantics of the periodic time expression thus dictate that the associated event can only occur if the start time expression is satisfied by the time of request, and such time falls within the interval specified by the interval expression. The duration of the event, if it occurs, would be governed by the duration expression.

The "Logical Expression" tag contains a set of predicates, where each predicate may contain a context-condition expressed in terms of role attributes, or embed within itself another logical expression. Hence, the structure allows evaluation of nested conditions expressed by multiple logical expressions. The predicates are composed of context-based parameters, where the "NameParam" tag contains the name of the parameter to be evaluated, and the "ValueParam" tag contains its value that is to be checked according to the given "Operator". For instance, any attribute supplied as part of user credential expression may be compared for a pre-requisite value needed for certain

<!-- Definitions of Temporal Constraints> ::= <XTempConstDef [xtcd_id = (id)]> [<!—Interval Expression>] [<!-- Periodic Time Expression>] [<!-- Duration Expression>] </XTempConstDef>	<!-- Periodic Time Expression> ::= <PeriodicTimeExpr pt_expr_id = (id) [d_expr_id = (id)] [i_expr_id = (id)] > <!-- Start Time Expression> </PeriodicTimeExpr>
<!—Interval Expression> ::= <IntervalExpr i_expr_id = id)> <begin> (date)</begin> <end> (date)</end> <IntervalExpr>	<!-- Duration Expression> ::= <DurationExpr d_expr_id = (id)> <cal>{Years|Months|Weeks|Days}</cal> <len> (number)</len> </DurationExpr>
<!-- Start Time Expression> ::= <StartTimeExpr [pt_id_ref =(pt_id)] > [<Year>{all|odd|even} /<Year>] [<!--MonthSet>] [<!--WeekSet>] [<!--DaySet>] </StartTimeExpr>	<!--MonthSet> ::= <MonthSet> {<Month>{1|..|12}</Month>}1-12 </MonthSet >
<!--WeekSet> ::= <WeekSet> {<Week>{1|..|4}</Week>}1-4 </WeekSet >	<!--DaySet> ::= <DaySet> {<Day>{1|..|7}</Day>}1-7 </DaySet >

Figure 3.6: X-Grammar for XTempConstDef sheet

role assignment or activation by supplying the attribute name as “NameParam”, the required values as ”ValueParam”, and the comparison operator as “Operator”. The “FuncParam” is an optional tag which is useful if the parameters in question can only be evaluated through a system review function, expressed as the status expressions of GTRBAC model. Multiple parameter names may be passed to functions that evaluate multiple parameters, with the distinction among parameter types made with the “type” attribute. As an example of predicates, we might evaluate status expressions for a role by supplying a status condition such as “`active` r `for` u” as “FuncParam”, the role name and the user id as two instances of “NameParam”, and the value of either “True” or “False” as the “ValueParam”. In such situations where a boolean output is returned, only “eq” operator is useful for comparison. The “Logical Expression” tag also has an optional op-code attribute that determines the evaluation logic of the predicates. On the similar lines as the constraint tag, an op-code of (i) “AND” implies that all constituent predicates must be true for the logical expression to be true, (ii) “OR” implies that at least one predicate must be true for the logical expression to be true, and (iii) “NOT” implies that none of the predicates must be true for the logical expression to be true. The op-code defaults to “AND” if none is specified. The grammar for logical expression specification is shown in Figure 3.7.

<!-- Logical Expression> ::=
<LogicalExpr [op = {AND|OR|NOT}]>
  {<!-- Predicate>}+
</LogicalExpr>

<!-- Predicate> ::=
<Predicate>
  [{<Operator> {gt|lt|eq|neq} </Operator>
    [<FuncParam>(function name)</FuncParam>]
    {<NameParam [type=(role|user|attribute)]>
        (parameter name)</NameParam>}+
    <ValueParam>(value)</ValueParam> }
  |
    < !--LogicalExpression>]
</Predicate>

Figure 3.7: X-Grammar for Logical Expression

3.1.2.4 Triggers

The grammar for constraint specification is also used to capture the trigger mechanism of GTRBAC model. Since the predicates within a logical expression can include both temporal and non-temporal context-based parameters, they allow for

specification of context-based triggers in our X-GTRBAC framework. This set of triggers is supplied in a separate XTrigDef sheet. The grammar for XTrigDef sheet is shown in Figure 3.8.

<table>
<tr>
<td><!--Triggers Specification> ::=
<XTrigDef [xtd_id = (id)]>
{<!-- Trigger>}*
</XTrigDef></td>
<td><!--Trigger> ::=
<Trigger [trig_id = (id)]>
<Head {role_name = (name) | perm_id = (id) }
[user_id = (id)]
action = {enable | disable |
assign | deassign | deactivate} >
</Head>
</Body>
<!--Triggering Constraint>
</Body>
</Trigger></td>
</tr>
<tr>
<td><!--Triggering Constraint> ::=
<TrigConstraint
[op = {AND|OR|NOT}] >
{<!--Triggering Condition>}+
</TrigConstraint></td>
<td><!—Triggering Condition> ::=
<TrigCondition>
[<!-- Logical Expression>]
<TrigCondition></td>
</tr>
</table>

Figure 3.8: X-Grammar for XTrigDef sheet

The “Head” tag of the trigger has an attribute that indicates the target role or the permission on which the trigger action is performed. An optional “user_id” attribute is also supplied for triggers that need to perform the action with respect to certain individual users. The triggering constraint in the “Body” tag is semantically similar to the constraints discussed above, and is evaluated in an analogous manner. The action associated with the trigger is performed if the constraint evaluates to true.

3.1.3 Permissions

The permissions for a given system are defined in terms of “objects” and associated “operations”. The “operations” component of the permission is typically system-dependent, such as read, write, delete, create, operate etc. The security administrator creates the permissions that associate the objects in the system with corresponding operations. The set of permissions for a system is expressed in the form of an XML document that we refer to as XML Permission Sheet (XPS). The grammar for XPS is shown in Figure 3.9.

<!-- XML Permission Sheet> ::= <XPS [xps_id = (id)]> {<!-- Permission Definition>}+ </XPS>	<!-- Permission Definition> ::= <Permission perm_id = id [prop= (prop op)] > <Object type= (type name) id= (id)/> <Operation> (access op) </Operation> </Permission>

Figure 3.9: X-Grammar for XPS

The "perm_id" is a unique permission identifier. An object in our framework can represent any system resource, such as documents, or inventory products, to which permission is being assigned. Each object is represented by a unique id and an associated type attribute. The access control requirements for various object types in an enterprise are therefore handled uniformly by our X-GTRBAC framework. The extent of the access is defined by the associated operation, indicated by an access opcode which is one of an enumerated set of values in the system. The resources in the system are modeled as XML, and the natural hierarchical structure of XML DOM is used to capture the physical object hierarchy. An object hierarchy could be composed of either documents, or document elements (in case of XML documents), or a series of inventory products organized according to their order of assembly, or any other organization of system resources. A permission can, hence, have an optional propagation option, given by the "prop" attribute, which indicates whether or not it propagates down the object hierarchy. We allow the propagation options "no_prop", "first_level" and "cascade" [24]. If no propagation option is explicitly supplied, it is assumed to be "no_prop", i.e. no propagation. However, the security administrator can specify a different propagation option at the time of permission-to-role assignment if a role demands sufficient privileges.

3.2 Policy Administration

The information about users, roles and permissions, and the related credentials, separation of duty constraints, temporal constraints, and triggers, available from the corresponding XML documents are used in the process of policy administration. The security administrator uses these XML sheets to specify the policy base for the protected enterprise resources. The documents generated in this phase include an XML User-to-Role Assignment Sheet (XURAS) and an XML Permission-to-Role Assignment Sheet

(XPRAS). These assignments are specified through XML schemas. Keeping the user, role, and permission specifications separate from their assignments allows independent design and administration of the policy, and hence supports a modular implementation of the X-GTRBAC system.

Table 3.1
The XML sheets comprising the XML Policy Base

Primary Policy Sheets	Policy Definition Sheets
XUS	XCredTypeDef
XRS	XSoDDef
XPS	XTempConstDef
XURAS	XTrigDef
XPRAS	

The policy sheets in the policy base are summarized in Table 3.1. The information from the policy base is used to enforce the authorization constraints. More specifically, the users are allowed access to resources based on their assigned roles per the XURAS and the associated permissions per the XPRAS. The grammar for the specification language for the generation of these assignment documents is presented below. The corresponding schemas are provided in Appendix–B.

3.2.1 User to role assignment

The grammar for XURAS is shown in Figure 3.10. Each "UserRoleAssignment" (URA) tag has an associated "role_name" attribute, and contains a set of "AssignUsers" tags containing the set of users who are to be considered for potential assignment to the specified role. Each such user is identified by the "user_ id" attribute of the corresponding "AssignUser" tag. This tag also contains the assignment constraint for this particular user. The assignment constraint has a "cred_type" attribute that specifies the credential type that the user must possess in order to be considered for a potential role assignment. The remaining part of the constraint is semantically similar to the constraints discussed above, and is evaluated in an analogous manner. The user is assigned to the specified role if the constraint evaluates to true. Similar logic applies to de-assignment of users from roles. Note that a special user with user_id = "any" is recognized by the

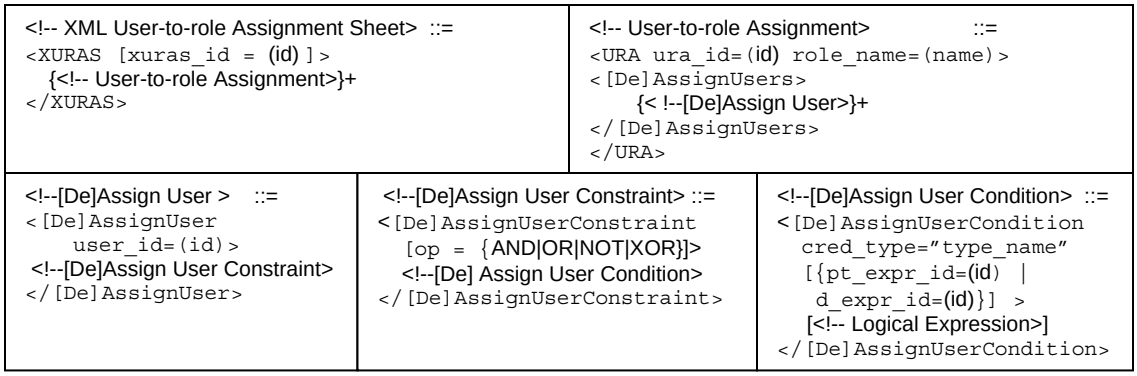

Figure 3.10: X-Grammar for XURAS

system as an unknown user, who may be required to supply additional assignment conditions in order to be assigned to a particular role. If no explicit conditions are specified, then any user could be assigned the particular role, which usually is the "guest" role in most enterprise applications.

3.2.2 Permission to role assignment

The grammar for XPRAS is shown in Figure 3.11. Each "PermissionRoleAssignment" (PRA) tag has an associated "role_name" attribute, and contains a set of "AssignPermission" tags containing the set of permissions that are to be potentially assigned to the specified role. Each such permission is identified by a "PermId" tag within the corresponding "AssignPermission" tag. Note that the permissions would typically be subject to periodic-time or duration constraints, and hence we allow the option of specification of periodic-time or duration constraint expression for the permission assignment. The permission is assigned to the specified role if the

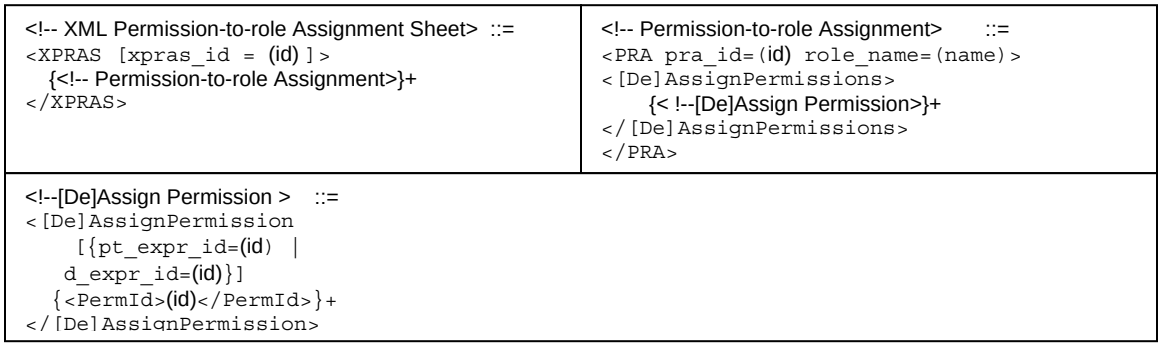

Figure 3.11: X-Grammar for XPRAS

constraint evaluates to true. Similar logic applies to de-assignment of permissions from roles.

3.3 System Architecture and Implementation

In this section, we present the system architecture of X-GTRBAC. We first provide an overview of the system components and technologies, and then discuss the implementation details to illustrate the process of specification and enforcement of an enterprise's access control policy.

3.3.1 Overview

The X-GTRBAC framework allows the XML-based enterprise policies to be specified and enforced through a Java-based GUI-enabled application. The application code is readily integrated into a Web browser by an application-to-applet transformation mechanism provided by Java.

The overall system design is depicted in Figure 3.12. As indicated in the figure, the two main sub-systems of X-GTRBAC Module are the XML Processor and the GTRBAC Processor. The XML processor is implemented in Java using Java API for XML Processing (JAXP). Custom modules have been designed to get the DOM instance of parsed XML documents and forward them on to the GTRBAC Processor. The GTRBAC Module then administers and enforces the policy according to the supplied policy information.

The policy information is contained in the XML Policy Base. A document composition module external to X-GTRBAC is provided to compose the policy documents. This module composes the policy sheets listed in Table 3.1. The policy sheets from the XML Policy Base are then loaded into the X-GTRBAC Module by the security administrator. Since X-GTRBAC can act as both stand-alone and web-deployable application, it may be invoked from either the local system, or remotely through an XML-aware browser. Hence, the X-GTRBAC Module seamlessly interfaces with an external client across distributed domains over an interconnect network (i.e. LAN, WAN etc.). The client may submit an access request through any standard XML-based Web services messaging protocol, like SOAP [13]. Similarly, the access authorization is returned via the same protocol.

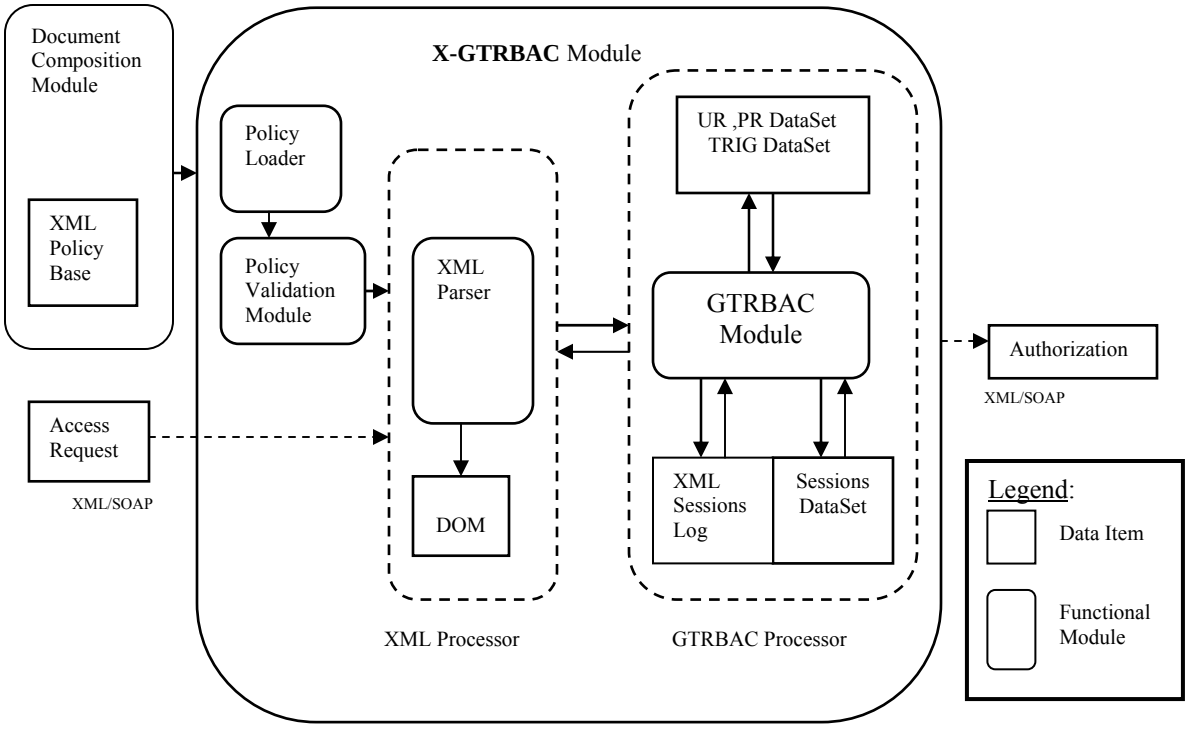

Figure 3.12: X-GTRBAC System Architecture

3.3.2 XML processor

The XML Processor contains the XML Parser and the DOM tree representations of the supplied XML documents. The X-GTRBAC system provides a Policy Loader to load the policy sheets for a given policy. As a next step, functionality is provided via a Policy Validation Module to validate the policy sheets in terms of existence checking and type conformance. This means that all users, roles, and permissions referenced in XURAS, XPRAS and XTrigDef sheet must exist in the corresponding XUS, XRS and XPS respectively. Also, all the referenced data must exist in the corresponding definition files. This means that (i) the credential types associated with the users in XUS must conform to the type definitions in the XCredTypeDef sheet, (ii) the separation of duty constraint sets referenced in the XRS must be present in XSoDDef sheet, and (iii) the periodic-time, start-time, interval, and duration expressions referenced in XRS must be present in XTempConstDef sheet. This validation support is provided by Apache Xalan

XSLT engine built into JAXP. Once the policy sheets are validated, the corresponding DOM tree representation is generated and passed on to the GTRBAC Processor. A facility is provided to display the instance of the DOM tree via the X-GTRBAC GUI.

3.3.3 GTRBAC processor

The GTRBAC Processor contains the GTRBAC Module and associated data items generated by the GTRBAC Module. It performs the policy administration and enforcement tasks.

Policy administration: The GTRBAC Module provides functionality to parse the DOM tree structures supplied by the XML Processor, and retrieves the relevant information into its internal data structures. The policy assignments are checked against the RBAC consistency rules, similar to those outlined in [25], against violations of any SSD, DSD, or cardinality constraints. A consistent assignment means, for instance, that a user in question will be assigned by the GTRBAC Module to the corresponding role because it satisfies all the required credential and consistency conditions. The permissions in the system are also assigned to roles under similar consistency notions. It may be noted that for all the users who have been assigned to roles, the actual role activation would occur when the user actually logs into the system and requests a role. The notion of role assignment in this context is of static type, i.e. it implies that the user has been declared as assignable to the said role based on already supplied credential information. There can also be a dynamic role assignment for an unknown user based on his/her credentials supplied at the time of login. These static and dynamic role and permission assignments, together with the role activation and enabling rules and triggers information, create the complete internal representation of the XML Policy Base within the GTRBAC Processor for enforcement of the policy. A collection of these policy information items are referred to as UserRole (UR) datasets, PermissionRole (PR) datasets, and TRIG dataset. A facility is provided to display the UR, PR and TRIG datasets via the X-GTRBAC GUI.

Policy enforcement: The information from the internal data structures is then used by the GTRBAC Module to enforce the policy and manage user sessions. The initial login into the system will create a default session for the user with a pre-specified “minimal” set of roles activated based on the supplied user credentials. The initial login can be the

"user_id" from the XUS, if it is a known user, or a "user_id" of "any", as discussed above. In addition to the default set of activated roles, more roles can also be activated if the user credentials so allow. Any triggers associated with role activation or other events are handled by the GTRBAC Module based on the information from the TRIG dataset. Access to resources is requested in the form of an XML Access Request (XAR) that specifies the "object type" and "object id" of the requested resource. An XAR could be submitted locally or remotely as an assertion in SOAP or similar XML-based messaging protocol. This access request is then evaluated based on the currently activated roles for this user. Only those resources may be accessed during a session for which the activated set of roles has associated permissions. Both the login information and XARs for a user are stored in an XML Access Sheet (XAS). The session-related information is contained in the Sessions Dataset within the GTRBAC processor. This information is extracted from an activity log maintained for every user by the GTRBAC module which we refer to as an XML Sessions Sheet (XSS). A session parameter is included in the XSS to record the domain from which the user is requesting access. In addition to the domain of the requesting user, the XSS also contains the attributes such as "login_time", "login_date", and "duration" of the session. These attributes are used to capture the activity profile of the user. Such information is constantly updated into the Sessions DataSet, where it can be dynamically processed, and incorporated into the access decisions. This feature is useful in certain situations where context information may be an important decision parameter, as discussed in Chapter 2. The grammar for a typical XAS and XSS is shown in Figure 3.13.

```
<xas [xas_id= (id)] >
 <login login_id= (id)>
    [<!--CredType>]
 </login>
 ……….
 <xar xar_id= (id)>
    {<Object type= (type name) id= (id)/>}+
</xar>
<xas>
```

(a)

```
<xss [xss_id= (id)] >
 <session>
  <session_id> (id) </session_id>
  <user_id> (user id) </user_id>
  <role_name> (role name) </role_name>
  <domain> (domain name) </domain>
  <login_time> (time) </login_time>
  <login_date> (date) </login_date>
  <duration> (duration) </duration>
  <active> {Yes|No} </active>
 </session>
</xss>
```

(b)

Figure 3.13. X-Grammar for (a) XAS (b) XSS

This chapter discussed the specification language and system architecture of X-GTRBAC. In the next chapter, we discuss a CIE example and motivate the applicability of our model by providing a mechanism to specify the CIE access control policy in X-GTRBAC framework.

4. X-GTRBAC AND COMPUTER INTEGRATED ENTERPRISE

This chapter presents a CIE application that is currently being implemented on our system, and discusses how the CIE specifications can be systematically mapped to our X-GTRBAC framework to highlight the latter's significance.

4.1 CIE Policy Specification

The access control policy for the CIE is essentially composed of the domain level policies described for each domain within the enterprise. These domain level policies capture the specifications of roles, users, permissions, and the related assignments for their respective domains. In essence, each such policy captures a minimal set of specifications that should include the following:

Functional roles and hierarchies: We let the roles in the CIE be represented by a functional role hierarchy that assigns, at each level of the system, a role that is needed to carry out the associated function. This role hierarchy captures the semantics of the top-down requirements interfacing[2] and bottom-up requests interfacing[3] within the enterprise [26]. In addition, it associates with the role at each level a set of responsibilities, and corresponding permissions to carry out those responsibilities. These responsibilities and permissions are captured in the domain level policies that are supplied according to the specific needs and requirements of the enterprise. The functional role hierarchy for the CIE in our application is shown in Figure 4.1. The corresponding policy name for each domain is placed in an oval above each column. The overall policy of the enterprise may then be composed of the combination of all domain level policies. Along the edges are placed the names of possible resources that are accessed by the respective roles at

[2] Interfacing requirements of management and staff to factory floor
[3] Interfacing requests to management and staff from factory floor

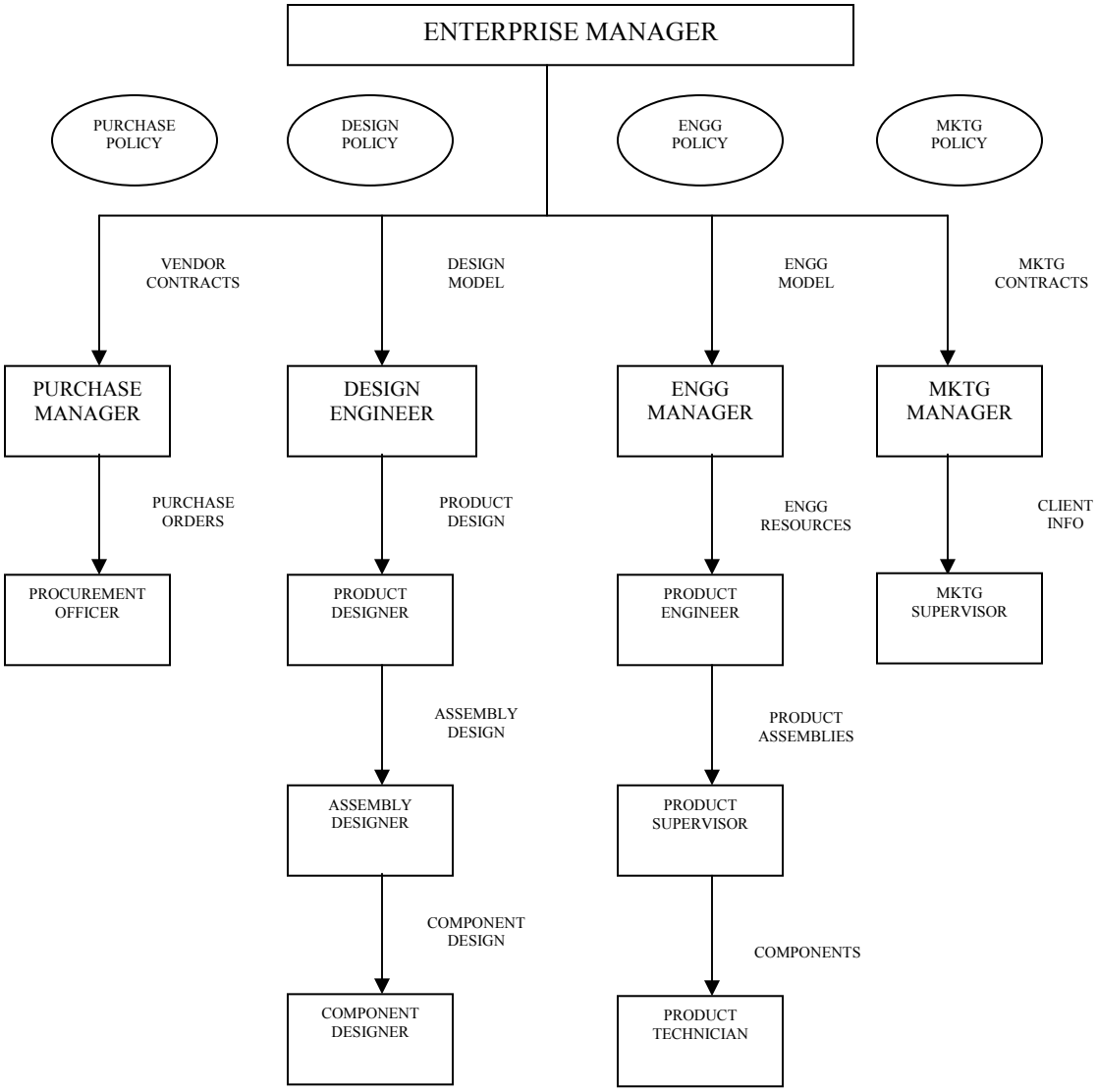

Figure 4.1: The functional role hierarchy and accessed system resources at each level.

each subsequent level. Only a subset of relevant functional modules has been shown in the hierarchy to illustrate the applicability of X-GTRBAC policy specification framework. The hierarchy can accordingly be extended and new policies defined as per need for a specific enterprise.

Role enabling and activation constraints: The functional role hierarchy imposes a partial ordering on the timing, order, and extent of accesses by the various roles. This constitutes the temporal semantics of access control within the CIE, and could be captured by a

Directed Acyclic Graph (DAG), such as the one shown in Figure 4.2. This particular graph represents the execution time-frame of a certain project scenario for the CIE being implemented in our system. The project requires the pooling of human, technical, commercial and engineering resources from various domains within the CIE. The timing between two events is captured on the connecting arrows. Where it is not explicitly stated, the default duration is 1 week. Note that the total time to finished product according to the DAG is then 7 weeks. Based on the duration of the individual tasks in the project, the corresponding roles in the CIE need to be enabled and disabled. The enabled roles would have further constraints on activation in situations where there exist other activation constraints. The permissions for the enabled and assigned roles would also be constrained according to the involvement of the role in the current stage of the project. The temporal constraint specification mechanism provided by X-GTRBAC would be used to transform the temporal constraints specified by the DAG into XML policies for the CIE.

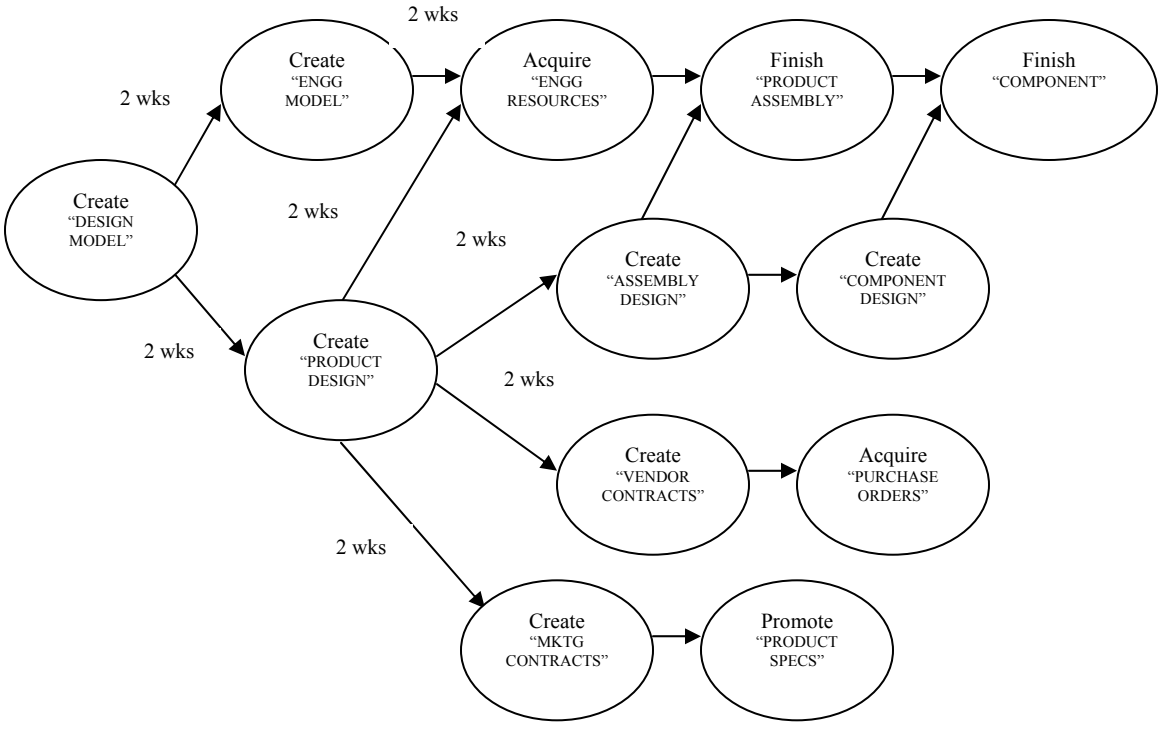

Figure 4.2: The DAG representing the execution time-frame for a project within the CIE

In the light of the preceding discussion on functional roles and tasks, we list in Table 4.1 a subset of constraints that is implemented in the CIE represented by the role hierarchy and DAG of Figures 4.1 and 4.2 respectively.

Table 4.1
A subset of constraints derived from the role hierarchy of Figure 4.1 and DAG of Figure 4.2 for the CIE

#	Constraint Type	Role	Constraint Description
1.	Enabling	`Design Manager`	Is enabled only starting 1st week of every quarter of year 2003
2.	Activation	`Design Manager`	May be activated only by one user at a time
3.	Enabling	`Engg Manager`	Is enabled only : (i) starting 3rd week of every quarter of year 2003, and (ii) if `Design Manager` role is enabled
4.	Activation	`Engg Manager`	May be activated only if `Design Manager role` is activated
5.	Enabling	`Product Designer`	Is enabled only: (i) starting 3rd week of every quarter of year 2003, and (ii) if `Design Manager` role is enabled
6.	Activation	`Product Designer`	May be activated only if `Design Manager role` is activated
7.	Enabling	`Product Engineer`	Is enabled only: (i) starting 5th week of every quarter of year 2003, and (ii) if `Product Designer AND Engg Manager` role is enabled
8.	Activation	`Product Engineer`	May be activated only if Product `Designer AND Engg Manager role` is activated
9.	Enabling	`Purchase Manager / Marketing Manager`	Is enabled only: (i) starting 5th week of every quarter of year 2003, and (ii) if `Product Designer` role is enabled
10.	Activation	`Purchase Manager / Marketing Manager`	May be activated only if Product `Designer role` is activated
11.	Static Separation of Duty (SSoD)	`Purchase Manager and Marketing Manager`	Both these roles may not be assigned to the same user at any given time
12.	Dynamic Separation of Duty (DSoD)	`Product Designer and Product Engineer`	Both these roles may not be simultaneously active in the same session by the same user
13.	Trigger	All roles	All active roles are de-activated at the start of 8th week of every quarter of year 2003

User credentials: An enterprise would ordinarily supply the set of users who would typically assume one of the functional roles within the CIE, and their associated set of

credentials that may be used in determining their assignment to particular roles. Without loss of generality, we list in Table 4.2 a subset of the users we consider in our example, along with their associated credentials. We assume for simplicity the convention that the credential types are named so as to reflect the current level of responsibility, or role, held by the user. In general, they may be named differently from the actual role name of the user. It should also be noted that the credential expression for a user with more than one different credential types (such as `george` in Table 4.2) is the union of the credential expressions of each of those credential types.

Table 4.2
A subset of users and associated credentials for the CIE

#	User Id	Credential Type	Credential Expression
1.	`john`	`Product Designer`	age=39, level=B, qualification=MS
2.	`nancy`	`Product Engineer`	age=36, experience=15, qualification=MS
3.	`george`	`Assembly Designer` `Product Supervisor`	age=29, level=D, experience=5, qualification =BS
4.	`carla`	`Product Supervisor`	age=28, experience=5, qualification =BS
5.	`smith`	`Procurement Officer`	age=32, level=B, region=northeast
5.	`dorothy`	`Procurement Officer` `Marketing Supervisor`	age=34, level=C, experience=20, region=midwest

Role assignment: The roles are assigned to users consistent with their supplied credentials. Such assignments may be constrained by temporal or non-temporal context conditions. Once again, without loss of generality, a subset of the role assignments considered in our example is listed in Table 4.3.

Permissions: The available permissions within the CIE represent the set of operations that may be performed on the available enterprise resources by eligible roles. The specification of these permissions is system dependent. We list in Table 4.4 a subset of the permissions assumed to be typically available in our example.

Permission assignment: The permission assignment determines the extent of access of various roles within the CIE. The roles are assigned permissions consistent with their responsibilities within the CIE. Such assignments may be constrained by temporal or

non-temporal context conditions. A typical set of permission assignments for the CIE considered in our example is listed in Table 4.5.

Table 4.3
A subset of role assignments in the CIE

#	Role	User Id	Credential Type	Assignment Condition
1	Design Manager	john	Product Designer	age>35 or level=A, qualification=PhD
2	Engg Manager	nancy	Product Engineer	age>35 or experience>10,qualification=MS
3	Product Designer	george	Assembly Designer	age>20 or level=B, qualification =BS
4	Product Engineer	george carla	Product Supervisor	age>20 or experience =5, qualification =BS
5.	Purchase Manager	smith dorothy	Procurement Officer	age>30 or level=B, region=midwest
6	Marketing Manager	dorothy	Marketing Supervisor	age>30 or experience>10, region=midwest

Table 4.4
A subset of available permissions in the CIE

#	Permission ID	Object ID	Object Type	Allowed Operation
1.	P1	Design Model	Document	All
2.	P2	Design Model	Document	Read
3.	P3	Engg Model	Document	All
4.	P4	Engg Model	Document	Read
5.	P5	Product Design	Document	All
6.	P6	Product Design	Document	Read
7.	P7	Engg Resources	Material Equipment	Operate
8.	P8	Vendor Contracts	Document	All
9.	P9	Marketing Contracts	Document	All

We capture the mapping of enterprise specifications to X-GTRBAC framework in Table 4.6. The table lists the functions and tasks within the CIE and the corresponding component that is responsible for it in the X-GTRBAC system. We next outline the process of representing this CIE policy in our X-GTRBAC framework.

Table 4.5
A subset of permission assignments in the CIE

#	Role	Permission ID	Assignment Condition
1.	Design Manager	P1	Is assigned starting 1st week of every quarter of year 2003 for 6 weeks
2.	Engg Manager	P2 P3	Is assigned starting 3rd week of every quarter of year 2003 for 2 weeks Is assigned starting 3rd week of every quarter of year 2003 for 4 weeks
3.	Product Designer	P2 P5	Is assigned starting 3rd week of every quarter of year 2003 for 4 weeks Is assigned starting 3rd week of every quarter of year 2003 for 4 weeks
4.	Product Engineer	P4 P6 P7	Is assigned starting 5th week of every quarter of year 2003 for 2 weeks Is assigned starting 5th week of every quarter of year 2003 for 1 week Is assigned starting 5th week of every quarter of year 2003 for 2 weeks
5.	Purchase Manager	P2 P8	Is assigned starting 5th week of every quarter of year 2003 for 1 week Is assigned starting 5th week of every quarter of year 2003 for 2 weeks
6.	Marketing Manager	P2 P9	Is assigned starting 5th week of every quarter of year 2003 for 1 week Is assigned starting 5th week of every quarter of year 2003 for 2 weeks

Table 4.6
The mapping of CIE specifications to X-GTRBAC framework

CIE Function / Task	Responsible X-GTRBAC module	Related X-GTRBAC data element
Specify Users and Credentials	Policy Loader	XUS, XCredTypeDef
Specify Functional Roles and Hierarchy	Policy Loader	XRS, XSoDDef
Specify Available Permissions	Policy Loader	XPS
Specify Task Scheduling and Timing (DAG)	Policy Loader	XTempConstDef
Specify Task Dependencies	Policy Loader	XTrigDef
Specify User Eligibility for Functional Roles	Policy Loader	XURAS
Specify Permission Criteria for Functional Roles	Policy Loader	XPRAS
Validate the Enterprise Policy	Policy Validation Module	Entire XML Policy Base
Generate Enterprise Policy Documents	XML Processor	Internal DOM tree structure
Create User-to-role / Permission-to-role Mapping	GTRBAC Module	UR and PR DataSets
Create and Maintain User Sessions	GTRBAC Module	XSS, Sessions DataSets
Request Access to Enterprise Resource	External GUI (local or remote)	XAR
Enforce Access Control Policy	GTRBAC Processor	UR, PR, Sessions DataSets

4.2 A CIE X-GTRBAC Policy

The specification language discussed in Chapter 3 can be used to compose the policy sheets for the CIE based on the mapping given in Table 4.6. The policy specification is then loaded into our implemented system for enforcement. Presented below is a discussion of the composition and implementation of the policy in our X-GTRBAC framework.

4.2.1 Policy definition sheets

In order to supply the necessary information needed to enforce an access control policy, the security administrator of the CIE loads the basic policy definitions related to credential types, separation of duty constraints, temporal constraints, and trigger specification. The policy definition sheets containing this information are shown in Figures 4.3-4.6.

4.2.2 Primary policy sheets

The security administrator next creates the primary policy sheets related to the users, roles, permissions, user-to-role assignments, and permission-to-role assignments. These sheets refer to the supplemental information provided by the policy definition sheets to specify an elaborate set of temporal and non-temporal context-based constraints for the enterprise access control policy. As discussed in Chapter 3, the information from both these sets of sheets is read into the X-GTRBAC module to constitute a complete representation of the XML Policy Base for policy enforcement. The primary policy sheets for the CIE are shown in Figures 4.7-4.11.

In particular, note that (i) the credential expression shown in XUS of Figure 4.7 captures the credential expression #1 in Table 4.2 for user `john`, and (ii) the activation and enabling constraints on `Design Manager` role in the XRS shown in Figure 4.8 capture the constraints # 1 and #2 of Table 4.1. Similarly, the reference to the dynamic separation of duty role set in `Product Designer` role captures the constraint # 12 of Table 4.1. Also note that (i) the role assignment shown in XURAS of Figure 4.10 captures the assignment condition #1 in Table 4.3 for user `john`, and (ii) the permission

assignments shown in XPRAS of Figure 4.11 capture the assignment conditions # 1 and #4 in Table 4.5 for `Design Manager` and `Product Engineer` roles respectively.

XCredTypeDef:

```
<?xml version="1.0" encoding="UTF-8"?>
<XCredTypeDef xctd_id="CIE_XCTD">
 <CredentialType cred_type_id="cPD"
       type_name="Product Designer">
  <AttributeList>
       <AttributeName type="integer"
          usage="mand">age</AttributeName>
       <AttributeName type="string"
          usage="mand">level</AttributeName>
       <AttributeName type="integer"
          usage="mand">qualification
         </AttributeName>
  </AttributeList>
 </CredentialType>
  ................
</XCredTypeDef>
```

Figure 4.3: Part of the XCredTypeSheet to define the user credentials specified in Table 4.2.

XSoDDef:

```
<?xml version="1.0" encoding="UTF-8"?>
<XSoDDef xsod_id="CIE_XSOD">
 <SSD_Role_Sets>
  <SSD_Role_Set SSD_Role_Set_id="SSD1"
                SSD_cardinality="1">
     <SSD_Role>Purchase Manager</SSD_Role>
     <SSD_Role>Marketing Manager</SSD_Role>
   </SSD_Role_Set>
 <DSD_Role_Sets>
  <DSD_Role_Set DSD_Role_Set_id="DSD1"
                DSD_cardinality="1">
     <DSD_Role>Product Designer</DSD_Role>
     <DSD_Role>Product Engineer</DSD_Role>
   </DSD_Role_Set>
 </DSD_Role_Sets>
</XSoDDef>
```

Figure 4.4: The XSoDDef sheet to define the separation of duty constraints specified in Table 4.1.

XTempConstDef:

```
<?xml version="1.0" encoding="UTF-8"?>
<XTempConstDef xtcd_id="CIE_XTCD">
 <IntervalExpr i_expr_id="Year2003">
        <begin>1/1/2003</begin>
        <end>12/31/2003</end>
  </IntervalExpr>
  <DurationExpr d_expr_id="SixWeeks">
        <cal>Weeks</cal>
        <len>6</len>
  </DurationExpr>
  ...........
  <PeriodicTimeExpr pt_expr_id="PTQuarterWeekOne"
       i_expr_id="Year2003">
    <StartTimeExpr>
        <Year>all</Year>
        <MonthSet>
            <Month>1</Month>
            <Month>4</Month>
            <Month>7</Month>
            <Month>10</Month>
        </MonthSet>
        <WeekSet>
            <Week>1</Week>
        </WeekSet>
    </StartTimeExpr>
  </PeriodicTimeExpr>
 ............
</XTempConstDef>
```

Figure 4.5: Part of the XTempDefSheet to define the temporal constraints specified in Table 4.1.

XTrigDef:

```
<?xml version="1.0" encoding="UTF-8"?>
<XTrigDef xtd_id="CIE_XTD">
 <Trigger trig_id = "disableAll">
   <Body role_name = "all"
         action = "disable" >
   </Head>
   <Body>
    <TrigConstraint>
      <TrigCondition
       pt_expr_id="PTQuarterWeekEight"/>
    </TrigConstraint>
   </Body>
 </Trigger>
</XTrigDef>
```

Figure 4.6: The XTrigDef sheet to define the trigger specified in Table 4.1.

XUS:

```
<?xml version="1.0" encoding="UTF-8"?>
<XUS xus_id="CIE_XUS">
<User user_id="john">
   <UserName>John</UserName>
   <CredType cred_type_id="cPD"
         type_name="Product Designer">
     <CredExpr>
        <age>39</age>
        <level>B</level>
        <qualification>MS</qualification>
     </CredExpr>
   </CredType>
   <MaxRoles>2</MaxRoles>
  </User>
 ..........
</XUS>
```

Figure 4.7: Part of the XUS to define the users specified in Table 4.2.

XPS:

```
<?xml version="1.0" encoding="UTF-8"?>
<XPS xps_id="CIE_XPS">
 <Permission perm_id="P1">
        <Object object_type="Document"
         object_id="DesignModel"/>
        <Operation>all</Operation>
 </Permission>
 ...........
 <Permission perm_id="P7">
        <Object
         object_type="MaterialEquipment"
         object_id="EnggResources"/>
        <Operation>operate</Operation>
 </Permission>
 ...........
</XPS>
```

Figure 4.9: Part of the XPS to define the permissions specified in Table 4.4.

XRS:

```
<?xml version="1.0" encoding="UTF-8"?>
<XRS xrs_id="CIE_XRS">
  <Role role_id="rDM" role_name="Design Manager">
   <Junior>Product Designer</Junior>
   <Cardinality>1</Cardinality>
     <EnabConstraint>
      <EnabCondition
         pt_expr_id="PTQuarterWeekOne"/>
    </EnabConstraint>
    <ActivConstraint>
      <ActivCondition>
        <LogicalExpr>
        <Predicate>
         <Operator>eq</Operator>
         <NameParam type=role>Design Manager
           </NameParam>
         <FuncParam>activated</FuncParam>
         <Value_param>false</ValueParam>
        </Predicate>
       </LogicalExpr>
      </ActivCondition>
    </ActivConstraint>
  </Role>
  <Role role_id="rPD" role_name="Product
        Designer">
    <DSD_Role_Set_id>DSD1</DSD_Role_Set_id>
    <Senior>Design Manager</Senior>
    <Junior>Assembly Designer</Junior>
     <EnabConstraint>
      <EnabCondition
         pt_expr_id="PTQuarterWeekThree"/>
    </EnabConstraint>
  </Role>
  .......
</XRS>
```

Figure 4.8: Part of the XRS to define the roles illustrated in the role hierarchy of Figure 4.1, and capture the constraints on them specified in Table 4.1.

XURAS:

```
<?xml version="1.0" encoding="UTF-8"?>
<XURAS xuras_id="CIE_XURAS">
 <URA ura_id="uraDM" role_name="Design
    Manager">
  <AssignUsers>
   <AssignUser user_id="john">
    <AssignConstraint>
      <AssignCondition cred_type="Product
    Designer">
       <LogicalExpr op="AND">
        <Predicate>
         <LogicalExpr op="OR">
           <Predicate>
             <Operator>gt</Operator>
             <NameParam>age</NameParam>
             <ValueParam>35</ValueParam>
           </Predicate>
           <Predicate>
             <Operator>eq</Operator>
             <NameParam>level</NameParam>
             <ValueParam>A</ValueParam>
           </Predicate>
         </LogicalExpr>
        </Predicate>
        <Predicate>
         <Operator>eq</Operator>
         <NameParam>qualification</NameParam>
         <ValueParam>PhD</ValueParam>
        </Predicate>
       </LogicalExpr>
      </AssignCondition>
    </AssignConstraint>
   </AssignUser>
  </AssignUsers>
 </URA>
.......
</XURAS>
```

Figure 4.10: Part of the XURAS to define the role assignments specified in Table 4.3.

XPRAS:

```
<?xml version="1.0" encoding="UTF-8"?>
<XPRAS xpras_id="CIE_XPRAS">
 <PRA pra_id="praDM" role_name="Design
    Manager">
  <AssignPermissions>
   <AssignPermission d_expr_id="SixWeeks">
    <PermId>P1</PermId>
   </AssignPermission>
  </AssignPermissions>
 </PRA>
.......
<PRA pra_id="praPE" role_name="Product
    Engineer">
  <AssignPermissions>
   <AssignPermission d_expr_id="TwoWeeks">
    <PermId>P4/PermId>
   </AssignPermission>
   <AssignPermission d_expr_id="OneWeek">
    <PermId>P6</PermId>
   </AssignPermission>
   <AssignPermission d_expr_id="TwoWeeks">
    <PermId>P7</PermId>
   </AssignPermission>
  </AssignPermissions>
 </PRA>
.......
</XPRAS>
```

Figure 4.11: Part of the XPRAS to define the permission assignments specified in Table 4.5.

4.2.3 Implementation experiences

This sub-section discusses our implementation experiences with the CIE example on our working prototype system.

The policy sheets are loaded into the X-GTRBAC system through the Policy Loader module. The XML Processor loads the policy sheets as DOM, and the GTRBAC module stores the policy information from the DOM into internal system data structures. Based on this information, some of the policy assignments effected by the GTRBAC processor are shown in Figure 4.12. In particular, we note the following:

(i) `john` has not been assigned the `Product Designer` role since he does not have the required qualification (`PhD`).

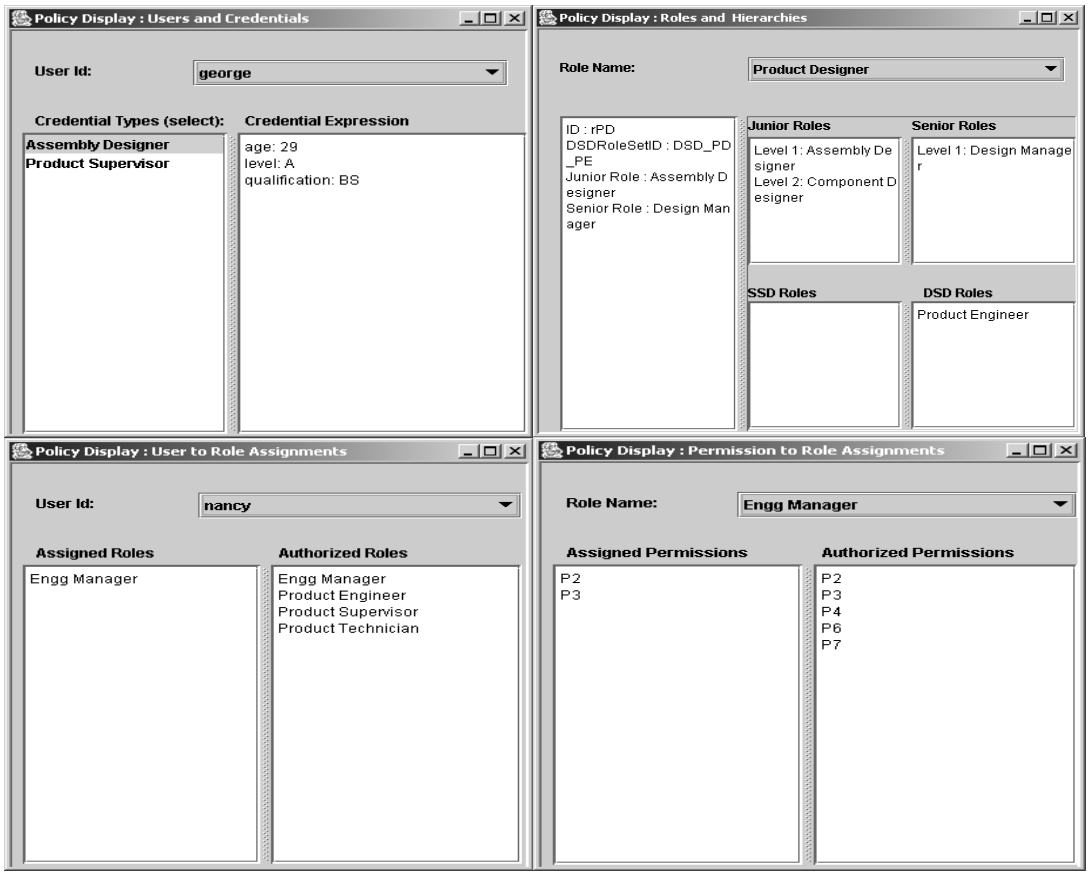

Figure 4.12: Snapshots of Policy Display, clockwise from top left: (i) User Credentials for `george`, (ii) Information for `Product Designer` role, (iii) User to Role Assignment for `nancy`, and (iv) Permission to Role Assignment for `Engg Manager` role

(ii) `nancy` has been assigned the `Engg Manager` role.

(iii) `dorothy` is assignable to both `Purchase Manager` and `Marketing Manager` roles. However, she can be assigned only to one of them (a policy validation rule) because of the static separation of duty constraint # 11 in Table 4.1.

(iv) `george` has been assigned to both `Product Designer` and `Product Engineer` roles, however, he may only have one of them activated at any given time due to the dynamic separation of duty constraint # 12 in Table 4.1. This follows from the notion of role assignment and activation as treated in our framework.

(v) `smith` cannot assume the role of `Purchase Manager` since his region is not `midwest`.

(vi) `nancy` has been authorized the permission `P7` by virtue of being senior to the `Product Engineer` role.

Note that the permissions of the roles within the CIE are constrained according to the policy specification by including the duration expression within the permission assignment constraints. The fact that all roles need to be disabled after the specified project duration expires will be handled by the `disable` action trigger that would fire at the start of the 8th week to disable all roles. In case a role is activated up to the end of specified duration, the semantics of GTRBAC model require that the trigger first deactivates the role, and then disables it. It may be mentioned that a role may also have explicit duration constraints if it so requires. Also indicated in the figures are the authorized roles and permissions that are acquired by virtue of the role hierarchy.

The policy administration process thus creates a complete internal representation of the specified enterprise policy. The policy enforcement phase then uses this information to allow the users to create sessions and access permitted resources. The various context conditions supplied within the activation constraints are then evaluated to make access control decisions, as discussed in Chapter 3. We invoke individual sessions for the users, and apply a 3 level security mechanism to effectively enforce the access control policy: (i) the user may only activate a role if he/she already meets the assignment criteria for it, and this restriction is imposed through the X-GTRBAC GUI by allowing only the assigned roles to appear in a drop down list of roles to choose from; (ii) the role activation goes through only if the role is enabled at that particular instance; (iii) when in an activated role, the user is restricted to request access to only those resources that the activated roles for the user have permission on, and this restriction is also imposed by allowing a selection from a drop down list of available accessible resources corresponding to the assigned permissions of the activated role. Hence, the 3 stage security mechanism ensures the enforcement of access control policy by restricting user access to only his/her available set of resources, and preventing any possibility for even requesting access to any unauthorized resource.

This chapter presented a comprehensive example of a CIE application currently being implemented in our X-GTRBAC framework. The next chapter concludes this thesis by providing a compendium of the accomplishments of this research, and outlining some future work.

5. CONCLUSION

In this thesis, we have highlighted the challenges for enterprise-wide access control and presented X-GTRBAC framework, an XML-based specification language based on the GTRBAC model and its implementation, which addresses them. Our specification language provides compact representation of access control policies for a generic CIE, while incorporating the security relevant features that have been motivated in this thesis to allow content-based, context-aware access control. The language conforms to the GTRBAC model, and hence incorporates the features from the NIST RBAC model and the temporal extensions proposed thereupon. We have emphasized separation of language schemas to provide efficient specification of definitions of RBAC elements, user-to-role and permission-to-role assignments, hierarchical and separation of duty constraints, and an elaborate set of temporal and non-temporal constraints. Such separation allows for an extensible design of the enterprise access control policy. The language can be used to specify GTRBAC policies for securing heterogeneous, distributed enterprise resources, and allows dynamic evaluation of user credentials and context information to provide fine-grained access control. An implementation based on Java has also been presented, and the system architecture has been illustrated to highlight its salient features. Our framework hence allows an enterprise's access control policy to be expressed in XML, and enforced through the X-GTRBAC system module. We also discussed a comprehensive example to motivate and illustrate the applicability of the model to a generic CIE. Our implementation experiences have also been presented on our working prototype system.

We plan to extend this work by analyzing the effect of more complex temporal role hierarchies in GTRBAC, as have been investigated in [27]. Also to be considered is enhanced support to allow more elaborate constraint specification for dynamically

changing access control requirements, including time-constrained cardinality and dynamic separation of duty constraints [28].

A major direction for future will be extending our X-GTRBAC framework to distributed inter-enterprise environments. This poses several challenges, the key amongst them being heterogeneity management. Each individual system of a multi-enterprise environment can have its own access control policy at a local level, and the integration of these local policies entails various challenges regarding reconciliation of semantic differences between local policies, secure interoperability, containment of risk propagation, global level policy management, etc. [29]. When local policies of individual enterprises are integrated to generate a global policy or meta-policy that governs the rules for access mediation within a multi-enterprise environment, semantic differences and inconsistencies among the local policies must be resolved in order to ensure secure interoperation. Efficient administration and management of global level policy becomes challenging, particularly as local policies evolve with time. We plan to explore the promise of our XML-based GTRBAC framework for its support for information management and access control in distributed systems to handle the heterogeneity challenges posed by such environments.

LIST OF REFERENCES

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

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

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

[4] J. B. D. Joshi, Elisa Bertino, Usman Latif, Arif Ghafoor, "Generalized Temporal Role Based Access Control Model (GTRBAC) (Part I) - Specification and Modeling", Submitted to IEEE Transaction on Knowledge and Data Engineering. Available as technical report at:
https://www.cerias.purdue.edu/infosec/bibtex_archive//archive/2001-47.pdf

[5] David F. Ferraiolo , Ravi Sandhu , Serban Gavrila , D. Richard Kuhn , Ramaswamy Chandramouli, “Proposed NIST standard for role-based access control”, ACM Transactions on Information and System Security, Volume 4, Issue 3, August 2001

[6] R. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman, “Role Based Access Control Models”, IEEE Computer Volume 29, Issue 2, February 1996.

[7] 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, Volume 3, Issue 2, February 2000.

[8] Why XML Schema beats DTDs hands-down for data
http://www-106.ibm.com/developerworks/xml/library/x-sbsch.html

[9] N. N. Vuong, G. S. Smith, Y. Deng, “Managing Security Policies in a Distributed Environment Using eXtensible Markup Language (XML)”, Symposium on Applied Computing, March 2001

[10] D. F. Ferraiolo, J. F. Barkley, D. R. Kuhn, "A Role Based Access Control Model and Reference Implementation Within a Corporate Intranet", ACM Transactions on Information and System Security, Volume 2, Issue 1, Feb 1999.

[11] E. Bertino, P. Bonatti, E. Ferrari, "TRBAC: A temporal role-based access control model", ACM Transactions on Information and System Security, Volume 4 , Issue 3, August 2001.

[12] 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 Systems Security, Volume 5, Issue 4 , November 2002.

[13] Simple Object Access Protocol (SOAP) 1.1
http://www.w3.org/TR/SOAP/

[14] eXtensible Markup Language (XML) 1.0, W3C Recommendation 6 October 2000
http://www.w3.org/TR/REC-xml

[15] Standard Generalized Markup Language (SGML)
ISO 8879. Information Processing -- Text and Office Systems - Standard Generalized Markup Language (SGML), 1986

[16] Web Services XML's Role
http://www.webreference.com/js/tips/011028.html

[17] XML Tutorial
http://www.javacommerce.com/tutorial/xmlj/intro.htm

[18] W3C XML Schema
www.w3.org/XML/Schema

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

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

[21] E. Bertino, C. Bettini, E. Ferrari, P. Samarati, "An Access Control Model Supporting Periodicity Constraints and Temporal Reasoning", ACM Transactions on Database Systems, 23(3):231-285, September 1998.

[22] M. Niezette and J. Stevenne, "An efficient symbolic representation of periodic time", In proceedings of First International Conference on Information and Knowledge Management, 1992.

[23] E. Bertino, S. Castano, E. Ferrari, M. Mesiti, "Controlled Access and Dissemination of XML Documents", Workshop On Web Information And Data Management, November 1999.

[24] E. Bertino, S. Castano, E. Ferrari, "Securing XML Documents with Author X", IEEE Internet Computing, May-June 2001.

[25] S. I. Gavrila , J. F. Barkley, "Formal Specification for Role Based Access Control User/role and Role/role Relationship Management", Proceedings of the third ACM workshop on Role-based access control, Fairfax, Virginia, United States, October 22-23, 1998.

[26] Purdue Reference Model for Computer Integrated Manufacturing http://iies.www.ecn.purdue.edu/IIES/PLAIC/PERA/ReferenceModel/index.html

[27] J. B. D. Joshi, Elisa Bertino, Arif Ghafoor, "Temporal Role Hierarchies in GTRBAC", Submitted to ACM Transactions on Information and System Security.

[28] J. B. D. Joshi, Basit Shafiq, Elisa Bertino, Arif Ghafoor, "Dependencies and Separation of Duty Constraints in GTRBAC", Accepted at Eighth ACM Symposium on Access Control Models and Technologies, 2003, Como, Italy.

[29] J. B. D. Joshi, A. Ghafoor, W. Aref, E. H. Spafford, "Digital Government Security Infrastructure Design Challenges", IEEE Computer, Volume 34, Issue 2, February 2001.

APPENDICES

APPENDIX A

XML SCHEMAS FOR RBAC ELEMENTS

(i) **User Credential**

```
<xs:schema>
  <xs:element name = “XUS”>
   <xs:complexType>
    <xs:attribute name = “xus_id” type=” xs:string” >
    <xs:element name = “user” type=” xs:string” maxOccurs=”unbounded”>
      <xs:complexType>
        <xs:sequence>
            <xs:attribute name = “user_id” type=” xs:string” use=”required” />
            <xs:element name = “user_name” type=” xs:string” minOccurs=”0” />
            <xs:element name = “cred_type” type=” cred_type”
  maxOccurs=”unbounded”/>
            <xs:element name = “max_roles” type=” xs:integer” minOccurs=”0” />
        <xs:/sequence>
      <xs:/complexType>
     <xs:/element>

    <xs:complexType name=”cred_type” >
     <xs:sequence>
      <xs:attribute name = “cred_type_id” type=” xs:string” use=”required” />
      <xs:attribute name =“type_name” type=”xs:string” />
      <xs:element name = “cred_expr” minOccurs=”0” />
       <xs:complexType>
          <!—each attribute from the schema(ii) is mapped to one element here →
            <xs:element name =
  “$[XCredTypeDef]/credential_type/attribute_list/attribute_name”
            type=”xs:string” />
        <xs:/complexType>
       <xs:/element>
      <xs:/sequence>
     <xs:/complexType>
   <xs:/complexType>
  <xs:/element>

   < !--refers to all the available credential types generated from the schema(ii) below -->

    <xs:key name="userID">
      <xs:selector xpath="user"/>
      <xs:field xpath="@user_id"/>
```

```
    </xs:key>
   <xs:key name="credTypeID">
     <xs:selector xpath="user/cred_type"/>
     <xs:field xpath="@cred_type_id"/>
    </xs:key>
   <xs:key name="typeName">
     <xs:selector xpath="user/cred_type"/>
     <xs:field xpath="type_name"/>
    </xs:key>
  <xs:keyref name="credTypeIdRef" refer="[XCredTypeDef]/credTypeId">
   <xs:selector xpath="user/cred_type"/>
   <xs:field xpath="@cred_type_id"/>
  </xs:keyref>
  <xs:keyref name="typeNameRef" refer="[XCredTypeDef]/typeName">
   <xs:selector xpath="user/cred_type"/>
   <xs:field xpath="@type_name"/>
  </xs:keyref>

 <xs:/schema>
```

(ii) **Credential Type Definition**

```
<xs:schema>
 <xs:element name =“XCredTypeDef”>
   <xs:complexType>
        <xs:attribute name =“xctd_id” type=”xs:string”/>
        <xs:element name =“credential_type” type=”xs:string”/>
          <xs:complexType>
            <xs:sequence>
                <xs:attribute name =“cred_type_id” type=”xs:string” use=”required”/>
                 <xs:attribute name =“type_name” type=”xs:string” use=”required”/>
                 <xs:element name = “attribute_list” minOccurs=”0”>
                   <xs:complexType>
                    <xs:element name = “attribute_name” maxOccurs=”unbounded”>
                          <xs:complexType>
                              <xs:sequence>
                               <xs:attribute name = “type” type=”xs:string” use=”required” />
                               <xs:attribute name = “usage” type=”xs:string” use=”required”>
                                        <xs:simpleType>
                                           <xs:restriction base="xs:string">
                                              <xs:enumeration value="mand"/>
                                              <xs:enumeration value="opt"/>
                                            <xs:/restriction>
                                       <xs:/simpleType>
                                 <xs:/attribute>
                               <xs:/sequence>
                              <xs:/complexType>
                         <xs:/element>
                        <xs:/complexType>
                        <xs:/element>
                    <xs:/sequence>
                  <xs:/complexType>
             <xs:/element>
<xs:/complexType>
```

```
      <xs:/element>

    <xs:key name="credTypeID">
       <xs:selector xpath="credential_type"/>
       <xs:field xpath="@cred_type_id"/>
     </xs:key>

     <xs:key name="typeName">
          <xs:selector xpath="credential_type"/>
          <xs:field xpath="@type_name"/>
      </xs:key>

   <xs:/schema>
```

(iii) **Role Definition**

```
<xs:schema>
<xs:element name =“XRS”>
   <xs:complexType>
       <xs:attribute name =“xrs_id” type=”xs:string”/>
        <xs:element name = “role” maxOccurs=”unbounded”>
         <xs:complexType>
           <xs:sequence>
            <xs:attribute name = “role_id” type=” xs:string” use=”required” />
            <xs:attribute name = “role_name” type=” xs:string” />
            <xs:element name = “SSD_Role_Set_id”  type=” xs:string” minOccurs=”0”/>
                  <xs:element name = “DSD_Role_Set_id” type=” xs:string” minOccurs=”0”/>
                  <xs:element name = “senior” type=” xs:string” minOccurs=”0”/>
                  <xs:element name = “junior” type=” xs:string” minOccurs=”0”/>
                  <xs:element name = “cardinality” type=” xs:integer” minOccurs=”0”/>
                 </xs:/sequence>
               <xs:/complexType>
             <xs:/element>
    </xs:complexType>
   <xs: /element>

<xs:key name="roleID">
     <xs:selector xpath="role"/>
     <xs:field xpath="@role_id"/>
   </xs:key>

<xs:key name="roleName">
  <xs:selector xpath="role"/>
  <xs:field xpath="@role_name"/>
</xs:key>

<xs:keyref name="roleNameSeniorRef" refer="roleName">
  <xs:selector xpath="role"/>
  <xs:field xpath="senior"/>
</xs:keyref>

<xs:keyref name="roleNameJuniorRef" refer="roleName">
  <xs:selector xpath="role"/>
```

```
  <xs:field xpath="junior"/>
</xs:keyref>

<xs:keyref name="SSDIdRef" refer="[XSoDDef]/SSDId">
  <xs:selector xpath="role"/>
  <xs:field xpath="SSD_Role_Set_id"/>
</xs:keyref>

<xs:keyref name="DSDIdRef" refer="[XSoDDef]/DSDId">
  <xs:selector xpath="role"/>
  <xs:field xpath="DSD_Role_Set_id"/>
</xs:keyref>

<xs:schema>
```

(iv) **Separation of Duty Definitions**

```
  <xs:schema>
    <xs:element name =“XSoDDef”>
      <xs:complexType>
           <xs:attribute name =“xsod_id” type=”xs:string”/>
           <xs:element name = “SSD_Role_Sets” >
             <xs:complexType>
               <xs:element name = “SSD_Role_Set” type=”SSD_Role_Set”  minOccurs=”0”/>
             <xs:/complexType>
           <xs:/element>
           <xs:element name = “DSD_Role_Sets” >
             <xs:complexType>
               <xs:element name = “DSD_Role_Set” type=”DSD_Role_Set” minOccurs=”0”/>
            <xs:/complexType>
           <xs:/element>
        </xs:complexType>
      <xs: /element>

     <xs:complexType name = “SSD_Role_Set”>
       <xs:sequence>
        <xs:attribute name = “SSD_Role_Set_id” type=” xs:string” use=”required” />
         <xs:attribute name = “SSD_cardinality” type=” xs:integer” use=”required” />
         <xs:element name = “SSD_Role” ”  type=” xs:IDREF”  maxOccurs=”unbounded” />
      <xs:/sequence>
     </xs:complexType>
    <xs:complexType name = “DSD_Role_Set”>
      <xs:sequence>
        <xs:attribute name = “DSD_Role_Set_id” type=” xs:string” use=”required” />
         <xs:attribute name = “DSD_cardinality” type=” xs:integer” use=”required” />
         <xs:element name = “DSD_Role” ”  type=” xs:string”  maxOccurs=”unbounded” />
      <xs:/sequence>
    </xs:complexType>

  <xs:key name="SSDId">
    <xs:selector xpath="SSD_Role_Set"/>
    <xs:field xpath="@SSD_Role_Set_id"/>
   </xs:key>
```

```
 <xs:key name="DSDId">
   <xs:selector xpath="DSD_Role_Set"/>
  <xs:field xpath="@DSD_Role_Set_id"/>
 </xs:key>

 <xs:keyref name="roleNameSSDRef" refer="[XRS]/roleName">
  <xs:selector xpath=" SSD_Role_Set"/>
  <xs:field xpath="SSD_Role"/>
 </xs:keyref>

 <xs:keyref name="roleNameDSDRef" refer="[XRS]/roleName">
  <xs:selector xpath="DSD_Role_Set"/>
  <xs:field xpath="DSD_Role"/>
 </xs:keyref>

<xs:schema>
```

(v) **Permission Definition**

```
<xs:schema>
<xs:element name =“XPS”>
   <xs:complexType>
            <xs:attribute name =“xps_id” type=”xs:string”/>
            < xs:element name = “permission” type=” xs:string” maxOccurs=”unbounded”>
              <xs:complexType>
                    <xs:sequence>
                       <xs:attribute name = “perm_id” type=” xs:string” />
                       <xs:element name = “object”>
                              <xs:complexType>
                                 <xs:sequence>
                                   <xs:attribute name = “object_type” >
                                    <xs:simpleType>
                                     <xs:restriction base="xs:string">
                                       <xs:enumeration value="Cluster"/>
                                       <xs:enumeration value="Schema"/>
                                       <xs:enumeration value=”Instance"/>
                                       <xs:enumeration value=”Element"/>
                                      <xs:/restriction>
                                     <xs:/simpleType>
                                    <xs:/attribute>
                                    <xs:attribute name = “object_id” type=” xs:string” />
                                 <xs:/sequence>
                              <xs:/complexType>
                       <xs:/element>
                       <xs:element name = “operation” >
                          <xs:simpleType>
                           <xs:restriction base="xs:string">
                             <xs:enumeration value="read"/>
                             <xs:enumeration value="write"/>
                             <xs:enumeration value="navigate"/>
                             <xs:enumeration value=”all"/>
                           <xs:/restriction>
                          <xs:/simpleType>
                         <xs:/element name>
                         <xs:element name = “prop” minOccurs=”0”/>
```

```
                    <xs:simpleType>
                     <xs:restriction base="xs:string">
                      <xs:enumeration value="no_prop"/>
                      <xs:enumeration value="first_level"/>
                      <xs:enumeration value="cascade"/>
                     </xs:restriction>
                    </xs:simpleType>
                   <xs:/element>
                  <xs:/sequence>
                 <xs:/complexType>
                <xs:/element>
  <xs:/complexType>
<xs:/element>

 <xs:key name="permName">
    <xs:selector xpath="permission"/>
    <xs:field xpath="perm_name"/>
  </xs:key>
 <xs:key name="objectID”>
    <xs:selector xpath="permission/object"/>
    <xs:field xpath="@object_id"/>
  </xs:key>

<xs:/schema>
```

Note: The areas in [ ] require XSL/XPath and XLink support and do not form valid schema constructs unless properly replaced by their exact syntactic expressions.

APPENDIX B

XML SCHEMAS FOR POLICY ADMINISTRATION DOCUMENTS

(i) **User to Role Assignment**

```
  <xs:schema>
    <xs:element  name=”XURAS”>
      <xs:complexType>
        <xs:attribute name =“xuras_id” type=”xs:string”/>
        <xs:element name = “ura”  maxOccurs=”unbounded”>
          <xs:complexType>
           <xs:sequence>
           <xs:attribute name = “ura_id” type=” xs:string” use=”required” />
            <xs:attribute name = “role_name” type=“xs:string” />
            <xs:element name = “assign_users”>
                <xs:complexType>
                    <xs:element name = “assign_user”>
                      <xs:complexType>
                        <xs:attribute name = “user_id” type=” xs:string” use=”required” />
                        <xs:element name = “assign_constraint” type = “assign_constraint_type”
minOccurs=”0”/>
                                <xs:/complexType>
                    <xs:/element>
                <xs:/complexType>
            <xs:/element>
            <xs:/sequence>
          <xs:/complexType>
        <xs:/element>
      <xs:/complexType>
    <xs:/element>
  <xs:schema>

  <xs:complexType name=”assign_constraint_type”>
        <xs:attribute name = “op” type=”xs:string” default=”AND”>
          <xs:simpleType>
           <xs:restriction base="xs:string">
             <xs:enumeration value="AND"/>
             <xs:enumeration value="OR"/>
             <xs:enumeration value=”NOT"/>
           </xs:restriction>
          <xs:/simpleType>
         <xs:/attribute>
         <xs:element name =“assign_condition” type=”assign_condition_type” maxOccurs=“unbounded” />
```

```
<xs:/complexType>

 <xs:complexType name=”assign_condition_type”>
    <xs:sequence>
      <xs:attribute name = “cred_type” type=“xs:string” use=”required”/>
      <xs:attribute name =“p_cond_id” type=”xs:string” />
      <xs:attribute name =“d_expr_id” type=” xs:string” />
      <xs:element name =“logical_expr” type=”logical_expr_type” minOccurs=”0”
maxOccurs=“unbounded” />
   <xs:/sequence>
<xs:/complexType>

  <xs:complexType name=”predicate_type”>
   <xs:sequence>
    <xs:element name = “operator” >
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:enumeration value="EQ"/>
             <xs:enumeration value="GT"/>
             <xs:enumeration value=”LT"/>
             <xs:enumeration value=”NEQ"/>
          <xs:/restriction>
        <xs:/simpleType>
       <xs:/element>
       <xs:element name = “name_param” type=“xs:string” />
       <xs:element name = “value_param” type=“xs:string” />
  <xs:/sequence>
 <xs:/complexType>

  <xs:complexType name=”logical_expr_type”>
   <xs:sequence>
    <xs:attribute name = “op” type=”xs:string” default=”AND”>
       <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:enumeration value="AND"/>
            <xs:enumeration value="OR"/>
            <xs:enumeration value=”NOT"/>
         </xs:restriction>
       <xs:/simpleType>
     <xs:/attribute>
     <xs:element name = “predicate”  minOccurs=“1” >
       <xs:complexType>
          <xs:choice>
               <xs:element name = “logical_expr” type=”logical_expr_type” />
                <xs:element name = “predicate” type=”predicate_type” />
         <xs:/choice>
       <xs:/complexType>
   <xs:/element>
  <xs:/sequence>
 <xs:/complexType>

<xs:key name="uraID">
   <xs:selector xpath="ura"/>
   <xs:field xpath="@ura_id"/>
```

```
   </xs:key>

 <xs:/schema>
```

(ii) **Permission to Role Assignment**

```
<xs:schema>
     <xs:element  name=”XPRAS”>
      <xs:complexType>
        <xs:attribute name =“xpras_id” type=”xs:string”/>
        <xs:element name = “pra”  maxOccurs=”unbounded”>
           <xs:complexType>
            <xs:sequence>
            <xs:attribute name = “pra_id” type=” xs:string” use=”required” />
           <xs:attribute name = “role_name” type=“xs:string” />
           <xs:element name = “assign_permissions”>
            <xs:complexType>
              <xs:element name = “assign_permission”>
                  <xs:complexType>
                     <xs:element name = “perm_id” type=“xs:string” maxOccurs=”unbounded”/>
                   <xs:/complexType>
              <xs:/element>
             <xs:/complexType>
          <xs:/element>
         <xs:/sequence>
        <xs:/complexType>
       <xs:/element>
      <xs:/complexType>
   <xs:/element>

              <xs:key name="praID">
                <xs:selector xpath="pra"/>
                <xs:field xpath="@pra_id"/>
               </xs:key>

           <xs:/schema>
```

(iii) **XML Access Sheet**

```
<xs:schema>
  <xs:element name = “login”  maxOccurs=”unbounded”>
   <xs:complexType>
    <xs:attribute name = “login_id” type=” xs:string” use=”required” />
    <xs:choice>
          <xs:element name = “user_id” type=“xs:string”/>
          <xs:element name = “cred_type” type=“cred_type”/>
     <xs:/choice>
   <xs:/complexType>
  <xs:/element>
  <xs:element name = “xar”  maxOccurs=”unbounded”>
   <xs:complexType>
    <xs:attribute name = “xar_id” type=” xs:string” use=”required” />
     <xs:element name = “object”>
        <xs:complexType>
          <xs:sequence>
```

```
            <xs:attribute name = “object_type” >
              <xs:simpleType>
                <xs:restriction base="xs:string">
                   <xs:enumeration value="Cluster"/>
                   <xs:enumeration value="Schema"/>
                   <xs:enumeration value=”Instance"/>
                  <xs:enumeration value=”Element"/>
                 <xs:/restriction>
                <xs:/simpleType>
               <xs:/attribute>
               <xs:attribute name = “object_id” type=” xs:string” />
            <xs:/sequence>
          <xs:/complexType>
        <xs:/element>
      <xs:/complexType>
   <xs:/element>

  <xs:key name="loginID">
     <xs:selector xpath="login"/>
     <xs:field xpath="@login_id"/>
   </xs:key>
  <xs:key name="xarID">
     <xs:selector xpath="xar"/>
     <xs:field xpath="@xar_id"/>
   </xs:key>

<xs:/schema>
```

(iii) **XML Sessions Sheet**

```
<xs:schema>
 <xs:sequence>
   <xs:element name = “session_id” type=“xs:string” />
   <xs:element name = “user_id” type=“xs:string” />
   <xs:element name = “role_name” type=“xs:string” />
   <xs:element name = “domain” type=“xs: anyURI” />
   <xs:element name = “login_time” type=“xs:time” />
   <xs:element name = “login_date” type=“xs:date” />
   <xs:element name = “duration” type=“xs:integer” />
   <xs:element name = “active” type=“xs:boolean” />
 <xs:/sequence>

  <xs:key name=”sessionID">
     <xs:selector xpath="."/>
     <xs:field xpath="session_id"/>
    </xs:key>

<xs:/schema>
```