CERIAS Tech Report 2006-13

A POLICY ENGINEERING FRAMEWORK FOR FEDERATED ACCESS MANAGEMENT

by Rafae Bhatti

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

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis Acceptance

This is to certify that the thesis prepared

By Rafae A. Bhatti

Entitled

A Policy Engineering Framework for Federated Access Management

Complies with University regulations and meets the standards of the Graduate School for originality and quality

For the degree of Doctor of Philosophy

Final examining committee members

A. Ghafoor , Chair

Charlie Hu

Elisa Bertino

Ninghui Li

Approved by Major Professor(s): A. Ghafoor

Approved by Head of Graduate Program: Mark J. T. Smith

Date of Graduate Program Head's Approval: 04/04/06

A POLICY ENGINEERING FRAMEWORK FOR FEDERATED ACCESS MANAGEMENT

A Thesis

Submitted to the Faculty

of

Purdue University

by

Rafae A. Bhatti

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2006

Purdue University

West Lafayette, Indiana

To my family, who have believed in me and supported me in every way throughout my journey of education which only culminates with this document but began more than 2 decades ago.

ACKNOWLEDGMENTS

Acknowledgement is due to my advisors Prof. Arif Ghafoor and Prof. Elisa Bertino, both of whom have been a tremendous source of guidance and inspiration, and their advice has been instrumental in the evolution of the work reported in this thesis. Also to be acknowledged are my committee members, Prof. Ninghui Li and Prof. Charlie Hu, for their valuable input and support in several phases of my graduate career. Last but not the least, the research presented in this thesis has been partially supported by the National Science Foundation under the NSF Grants # IIS-0209111 and IIS-0242419.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Bhatti, Rafae A. Ph.D., Purdue University, May, 2006. A Policy Engineering Framework for Federated Access Management. Major Professor: Arif Ghafoor.

Federated systems are an emerging paradigm for information sharing and integration. Such systems require access management policies that not only protect user privacy and resource security but also allow scalable and seamless interoperation. Current solutions to distributed access control generally fail to simultaneously address both dimensions of the problem. This work describes the design of a policy-engineering framework, called X-FEDERATE, for specification and enforcement of access management policies in federated systems. It has been designed from the perspectives of both security management and software engineering to not only allow specification of requirements for federated access management but also allow development of standardized policy definitions and constructs that facilitate policy deployment and enforcement in a federated system. The framework comprises of an access control language specification that is an extension of the well-accepted Role Based Access Control (RBAC) standard. The language extends RBAC to incorporate various essential features for federated access management. The framework also includes the design of an administrative model targeted at access control policy administration in a decentralized environment. The framework has been implemented as a research prototype that illustrates the use of X-FEDERATE as an enabling technology for secure Web-based federation with applications in federated digital libraries and federated electronic healthcare management.

1. INTRODUCTION

Access management is a key component of all information management systems. The recent evolution of information management architectures has resulted in the shift from centralized to decentralized control as evident in modern grid and peer-to-peer systems. These architectures have given rise to a collaborative landscape of information integration and federation enabled by the federated database technology connecting together heterogeneous information sources. Manifestations of such federated systems can be seen in the form of Web-services and multi-domain enterprise systems. Many military and commercial organizations have adopted the federated approach to online information management, be it for critical infrastructure protection such as the DoD NetCentric Directive [1] or wide dissemination of scholarly work such as the Federated Digital Library initiative [2].

At the same time, this federated paradigm of collaboration has placed increasing demands on the access management technologies to provide adequate support for secure dissemination of information. The overarching theme of the work reported in this thesis is to address the security requirements for federated information sharing, and to design approaches for distributed access management capable of fulfilling these requirements in modern day federated systems.

1.1 Policy-Based Access Management

The access management problem concerns itself with the design and administration of access management policies. Policies for distributed access management are particularly challenging because of the heterogeneity introduced by multiple administrative domains, which impacts both the design and administration of policies. The design of a policy-based management approach to satisfy the security requirements

for federated systems presents a two-fold challenge, where on the one hand the policy must incorporate the access management needs of the individual systems, while on the other hand the policies across multiple systems must be designed in a manner that they can be uniformly developed, deployed and integrated within the federated system.

An effective mechanism for access management in federated systems must take into consideration the access control requirements of protected resources as stated in the access control policies of each participating site. However, the requirements for federated information sharing are not only significantly more complex but also inherently heterogeneous across the multiple federated sites. The key challenges here include specification of authentication and authorization rules, design of interoperable federation protocols to allow Single Sign On (SSO), and specification and enforcement of authorization constraints. These requirements have been tabulated in Table 1.1. Addressing these challenges in the access control policies while overcoming the complexity and heterogeneity calls for developing a sophisticated policy engineering methodology that can be adopted for composing federated access management policies that can not only adequately capture the access management requirements but can also interoperate across the multiple federated sites. Additionally, given the scale and depth of modern-day distributed systems, it is imperative that the methodology be based on standardized constructs that can be readily integrated into existing systems. This requirement becomes increasingly significant when the deployment environment is a federated system.

Toward this end, the work presented in this thesis involves the design of a policy engineering framework for federated access management. We formulate a policy engineering methodology that incorporates principles from security management and software engineering. The methodology is based on the well-known Unified Modeling Language (UML). From a security management perspective, the goal is to meet the requirements for federated access management. From a software engineering perspective, the goal is to incorporate the well-known principles of software engineering

in the policy design to yield a specification that allows policies to be developed and managed in a standardized manner. This policy engineering methodology not only results in development of interoperable policies for federated information sharing but also facilitates usability and integration of the proposed framework in existing systems.

Table 1.1: Issues that impact policy design in federated information sharing

Issues	**Description**	**Impact on Policy Design**
Decentralized Administration	Federated systems are characterized by the principle of local autonomy which implies that each participating site retains local autonomy (i.e. administrative control over its resources). A centralized administration approach implies loss of autonomy for participating sites [3]. A decentralized approach lets the participating sites retain the authority for specifying authorization policies for federated users.	Decentralized administration requires that the policy be specified for an unseen user pool requiring differentiated access to a diverse set of resources located anywhere across the federation. This precludes the use of traditional approaches to distributed authorization that assume knowledge of user identities and resource locations. Even when this knowledge is available, the requirement of fine-grained access control would lead to rule-explosion in the policy given the size of federated population in open systems. To keep the rule set from becoming prohibitively large calls for a scalable approach.

Continued on next page

Issues	**Description**	**Impact on Policy Design**
Credential Federation	A federated access management policy must support a mechanism to transfer the credentials of the federated users across administrative boundaries for them to obtain federated resources according to the applicable authorization policies. This requirement also includes providing Single Sign On (SSO) to enable persistent authorization for federated users.	Credential federation demands interoperable protocols that can allow participating sites to communicate user credentials. Multiple policies may be necessary to evaluate the request of a federated user, which requires the support for combining rules from multiple policies. This entails that the policy be specified using a standardized vocabulary to allow multiple policies to seamlessly interoperate. The use of a standardized meta-model is essential for clear elicitation and integration of interoperable policy specification.
Constraint Specification	A federated system requires the specification of semantic and contextual constraints to ensure adequate protection of federated resources. Former includes high level integrity principles, such as Separation of Duty (SoD) [4]. Latter includes temporal or non-temporal attributes that must be evaluated to allow an access request.	The existing PBM approaches have tightly coupled constraints with policy rules which is an inflexible approach. Since the integrity requirements and contractual obligations within a federation might change on-demand, an access management mechanism must be flexible enough to facilitate such adaptation. The policy design must therefore be based on principles of modularity and reusability that allow independent yet interoperable specification of various components of the policy.

1.2 Contributions and Organization

To address the problem of policy-based management in federated systems, we present X-FEDERATE, a policy engineering framework for federated access management. It consists of an XML-based policy specification language, its UML-based meta-model, and an enforcement architecture. The UML modeling tool is an integral part of our policy engineering methodology, as it facilitates modeling, development, testing and integration of policies using standardized tools. The design of our policy specification language builds upon the Role Based Access Control (RBAC) model [5], and augments it with necessary extensions to support access management in a federated system.

Another contribution of this thesis is the design of an administration model for the X-FEDERATE policy framework. Administration is a key requirement for policy-based systems management, and becomes particularly important in the context of policies incorporating rich constraint specification, such as those for federated information sharing highlighted in Table 1.1. To meet this requirement, we present an administration model outlining the formal specifications of the administrative concepts for access management in our framework.

The remainder of the thesis is organized as follows. Chapter 2 covers related work and highlights the particular merits of our work with respect to the outlined challenges. Chapter 3 presents the X-FEDERATE policy engineering framework and the policy engineering methodology. The framework includes the details of the policy specification language, and a meta-model for our language based on the well-known UML specification. Chapter 4 presents the policy administration model for the X-FEDERATE framework. Chapter 5 presents the design of a federated architecture based on our framework. Chapter 6 presents a distributed systems application of the framework. It illustrates the use of the policy engineering methodology for federated healthcare information management. Chapter 7 concludes the thesis.

2. RELATED WORK

This chapter covers the related work in the policy-based access management approaches for distributed systems.

Policy-based approach to management of systems has been most widely reported in the literature in the context of network systems management (for e.g. [6]). One notable example of policy-based language for systems management is Ponder [7]. Ponder is a declarative policy language with the ability to support authorization and delegation policies, as well as obligation policies. However, authorization policies in Ponder are primarily aimed at allowing network users to manage network objects, with known user groups and object locations, and are therefore inadequate for a federated environment where users and resources are not identified in advance. It therefore does not support credential specification and federation requirements for access management in federated systems.

Ponder supports specification of contextual constraints, based on temporal and non-temporal parameters. However, contextual constraint specification is tied into the authorization policies, which reduces their modularity, and hence flexibility. Ponder also supports specification of SoD constraints through the use of meta-policies. However, the specification is at user-level, and is more complicated to maintain in a federated environment as opposed to a role-based SoD constraint. On the other hand, Ponder is well suited to the task that it is designed for, i.e. network services management. It has a management toolkit that allows policy specification, deployment, and dynamic adaptation, as well as support for off-line static analysis of policy conflicts [8].

The access control model for federated systems presented in [3] is based on a tightly coupled architecture. It concerns with defining principles for designing access

control policies in federated systems, and does not deal with policy-based management issues. It therefore does not address the particular issues related to credential specification, credential federation, or constraint specification highlighted in this thesis.

Various policy models have earlier been used for access control in centralized and traditional distributed systems, but not many approaches have been designed to meet the requirements for policy-based access management in federated systems as described in this thesis. Akenti [9] and Permis [10] are access control systems which use policies encoded in X.509 attribute certificates. Both assume authenticating credentials to be used for issuance of authorization certificates, much like our approach. Akenti supports discretionary access control (i.e. based on subject identities), leading to rule explosion in policy rule set. Permis uses role-based access control; it, however, does not provide support for specification and enforcement of user-to-role and role-to-permission assignment policies. Both these schemes do not support specification of semantic or contextual constraints.

Shibboleth [11] and Liberty Alliance [12] define protocols for attribute-based authentication in support of SSO in Web-based environments. The attributes of a user in Shibboleth are always acquired from his/her home site by the resource provider, whereas those in Liberty Alliance protocol can be provided by any identity provider on the Internet. Liberty Alliance protocol therefore establishes a circle of trust between identity providers and resource providers. Both schemes provide particular emphasis on user privacy, and the identity of the user is not known to the resource provider. However, the role of these schemes is limited to distributed authentication, and to providing attribute information for a user to be used in authorization decisions. They do not include mechanisms for specifying and enforcing authorization policies.

Security Assertions Markup Language (SAML) [13] and XML Access Control Access Control Language (XACML) [14] are emerging specifications aimed at addressing different aspects of distributed access management. As noted earlier, SAML

primarily provides a mechanism for credential federation, but does not provide any policies for use of those credentials. Also, SAML does not incorporate a way to establish trust between business partners exchanging credentials. XACML can be configured to support RBAC through its RBAC profile. XACML, however, does not support some of the essential features in the RBAC standard, such as specification of separation of duty, role hierarchy and cardinality constraints on the definition of roles. Although it does support the semantics of hierarchical RBAC (through permission aggregation), it does not allow hierarchies and hierarchical constraints to be defined directly on roles. Additionally, it does not support the administrative functions for session management as defined in the RBAC standard. Consequently, XACML primarily acts only as a PDP (Policy Decision Point) and lacks the infrastructure to enforce the access control policy. The X-FEDERATE framework presented in this thesis not only includes a specification language but also an enforcement architecture. It can therefore provide the functionality of both a PDP and a PEP (Policy Enforcement Point).

A key feature of our framework in comparison with all other related approaches is the policy engineering methodology that addresses the security management issues relevant to the design of policies for federated access management using the principles of software engineering. Based on this design methodology, we provide a UML-based meta-model for the policy specification language that allows the policy to be developed and managed in a standardized manner. While there exist UML-based specifications related to secure software design [15–17], there is none that captures the federated information sharing requirements outlined in this thesis. The work presented in [17] is a UML profile designed to capture the requirements for confidentiality, information flow, fair exchange, and secure communication in a distributed system, and also allows modeling of threat scenarios. It, however, cannot be used to specify access control policies. The approaches presented in [16, 17] have been designed to model access control policies, and both are based on RBAC model.

They, however, do not cater to the requirements of the federated paradigm, including the advanced constraint specification beyond the basic RBAC model.

The Resource Access Decision specification [18] by the Object Management Group is a very relevant effort in the area of providing a uniform policy specification for application systems to enforce resource-oriented access control policies. It, however, does not address the challenges outlined in this thesis with respect to federated information sharing.

From amongst all related approaches, we provide in Table 2.1 a comparison of X-FEDERATE with the two most closely related schemes, namely Ponder and XACML. It provides a clear illustration of the distinctive features of X-FEDERATE while contrasting them to these schemes based on the federated access management requirements outlined in this thesis.

Table 2.1: Comparison of X-FEDERATE & related work

Feature			**Ponder**	**X-FEDERATE**	**XACML**
Policy Modeling and Specification			YES (through an Interface Definition Language)	YES (through standardized UML-based meta model)	NO (no meta model-specification)
Conflict Resolution	Modality		YES (Precedence relationships)	YES (Rule combining modes)	YES (Rule combining modes)
	Semantic	Static SoD	YES (as constraint)	YES (as constraint)	NO
		Dynamic SoD	NO	YES (as constraint)	NO

Continued on next page

Feature			**Ponder**	**X-FEDERATE**	**XACML**
Support for RBAC [1]			Level 1	Level 3	Level 1
Support for policy management			YES (through a Management Interface)	YES (through an Admin Model)	NO
Parser & Interpreter			YES	YES	YES
Strong typing			YES	YES (through XML schema)	YES (through XML schema)
Constraint Specification	Semantic	Static SoD	YES (through meta policies)	YES (through SoD sets)	NO
		Dynamic SoD	NO	YES (through SoD sets)	NO
	Contextual	Temporal	NO	YES (through periodic time expressions)	NO
		Non - Temporal	YES(through parameter-value constraints)	YES (through logical expressions)	YES (through rule conditions)

[1]The following levels are defined by the authors to categorize the work reported on RBAC based on the RBAC standard [19] and its extensions reported in the literature.

- Level 1: Supports the notion of roles, and allows permissions to be associated with roles.
- Level 2: Level 1 + specification of constraints directly on the definition of roles.
- Level 3: Level 2 + role management functions as defined per the RBAC standard.

3. POLICY ENGINEERING FRAMEWORK

This chapter presents the design of X-FEDERATE, our policy engineering framework. The primary objective behind the development of X-FEDERATE is to outline a policy engineering methodology to provide support for policy-based access management in federated systems.

While policies for distributed access management have been developed, current approaches for policy-based management (PBM) do not transition well to the federated paradigm. Federated systems comprise of several distinct, potentially mutually untrusted administrative domains with heterogeneous policy specifications. Federated access management is therefore thwarted by the inherent incompatibility of multiple PBM approaches.

Additionally, designing policies for PBM in federated systems poses considerable challenges, as has been highlighted in Chapter 1. An effective mechanism for access management in federated systems must take into consideration the access control requirements of protected resources as stated in the access control policies of each participating site. However, the requirements for federated information sharing are not only significantly more complex but also inherently heterogeneous across the multiple federated sites. This complexity and heterogeneity of access control policies calls for developing a sophisticated policy engineering methodology that can be adopted for composing federated access management policies that can not only adequately capture the access management requirements but can also interoperate across the multiple federated sites. Additionally, given the scale and depth of modern-day distributed systems, it is imperative that the methodology be based on standardized constructs that can be readily integrated into existing systems. This requirement

becomes increasingly significant when the deployment environment is a federated system.

We shall first elaborate upon the design principles of our policy engineering methodology, and then present the details of the policy specification language.

3.1 Design Principles

We formulate a policy engineering methodology that incorporates principles from security management and software engineering. From a security management perspective, the goal is to meet the requirements for federated access management. From a software engineering perspective, the goal is to incorporate the well-known principles of software engineering in the policy design to yield a specification that allows policies to be developed and managed in a standardized manner. This policy engineering methodology not only results in development of interoperable policies for federated information sharing but also facilitates usability and integration of the proposed framework in existing systems.

As stated above, we formulate our policy engineering methodology for the X-FEDERATE framework by incorporating principles from security management and software engineering. We now elaborate upon the two principles involved in the design of our methodology.

3.1.1 Security Management Perspective

Recently, there has been a growing recognition of security problems in federated environments, and several emerging specifications in various stages of standardization have emerged [13, 14, 20]. But standards alone won't solve the problem. The answer lies in combining standards with policies that govern how shared information can be used. We believe that a PBM approach for access management provides a viable solution since it is flexible and adaptable enough to meet the unique requirements of a complex environment, such as a federated system. A key benefit of policies

for systems management is that policies are interpreted rather than compiled into program code, so can be changed dynamically without changes to the application code [6].

In Table 1.1, we have highlighted the key security management issues that impact the policy design for federated information sharing, and subsequently discuss the design goals of our framework. In response to these challenges, we have outlined a set of design goals to adequately capture the access management requirements for federated information sharing.

Among our design goals is to provide an interoperable specification for expressing access control policies that is compatible with emerging security standards for information federation. All notable emerging standards for Web-based federation are XML-based; we therefore use an XML-based policy specification language. As a consequence, our policy-based framework facilitates interoperability with complementary security protocols for federated systems.

The design of an access management policy should in principle not be strongly tied to any specific mechanism for provisioning of security attributes. Therefore, another design goal is to allow modular specification of authenticating and authorization credentials to provide support for pluggable authentication standards to be incorporated in a policy by a federating site. Being neutral to the authentication mechanism, we do not deal with the authentication system needed to generate the authenticator. In other words, we assume that the authentication information supplied to the system is already verified, and is encoded in an authenticating credential usable in our framework.

Our design is focused on specification of policies, and therefore we do not consider certain other auxiliary issues. For example, we do not deal with credential provisioning issues, which include deployment of credentials across multiple applications, typically through the use of directory services (such as LDAP). We also do not deal with identity aggregation issues involving multiple LDAP repositories for manipu-

lating composite credentials. Additionally, we assume that the channels used for network communication are secured by appropriate mechanisms (such as SSL/TLS).

3.1.2 Software Engineering Perspective

In addition to design issues relevant to security management, we also use software engineering principles to guide us in the design of our framework. We emphasize that software engineering is a broad discipline, and encompasses the various stages involved in the software development life cycle (SDLC). Our effort is primarily focused on the requirements specification (as related to federated access management), design, and implementation phases in the SDLC. Toward this end, we provide a UML-based meta-model of our language to facilitate policy specification. It facilitates capturing the requirements of the target application by including the appropriate UML components in the policy, and allows rapid translation of policy specification into implementation code through the use of state-of-the-art translation technologies [21].

The use of a standardized model to elicit policy specification has other advantages. A standardized vocabulary facilitates specification of various security policies, and allows them to be applied in an interoperable manner in a federated system since the policies in the system will be instances of a single meta-model. This encourages usage of our policy language for federated information sharing in an organization with minimal change to existing policy configuration. Our approach complements similar efforts in the industry, such as Common Information Model (CIM) specifications by the Distributed Management Task Force [22]. Recent research has also reported the use of UML for constructing middleware-aware system models to promote integration and interoperability with heterogeneous middleware components [23].

We now turn to the policy specification language of our X-FEDERATE framework.

3.2 Policy Language

This section describes the key features of X-GTRBAC (XML-based Generalized Temporal Role Based Access Control), our XML-based policy specification language. Our specification language is an extension of the RBAC model suitable for addressing the access management challenges in federated systems discussed in this thesis.

X-GTRBAC language specification is captured through a context-free grammar called X-Grammar, which follows the same notion of terminals and non-terminals as in Backus-Naur Form (BNF), but supports the tagging notation of XML which also allows expressing attributes within element tags. The use of attributes helps maintain compatibility with XML schema syntax, which serves as the type definition model for our language. 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. The data types of the values of elements or attributes are indicated inside parenthesis ˝( )˝symbol. Since it follows BNF convention, X-Grammar can be accepted by a well-defined automaton to allow formal analysis (since existing compiler tools for BNF grammars can be applied). Another reason that favors the use of X-Grammar syntax for policy specification instead of directly working with XML schemas is better readability and presentation. Examples of X-Grammar policies are given in following sections. The complete syntax of X-GTRBAC language specification appears in Appendix A.

3.3 Policy Components

We now describe the main components of our X-GTRBAC. While doing so, we motivate our design decision by evaluating existing approaches against our stated

requirements, and pointing out the merits of our design with respect to our objectives.

3.3.1 Credentials

Credentials are a key component of an access control language. A credential encodes the authentication and authorization information for the users. We have indicated in Table 1.1 that a heterogeneous and unfamiliar user and resource pool in a federated system complicates credential specification, since it precludes the use of traditional approaches to distributed authorization (such as X.509 based PKI) that assume knowledge of user identities and resource locations.

The approached presented in [24, 25] are well-known examples of distributed schemes that have used identity-based X.509 certificates for user authentication. The authentication information (i.e. public keys) is then used to construct an authorization credential that comprises of a set of resource-specific rules. The credentials are bound to user identities and therefore this approach to credential specification is not scalable. Even when knowledge of identities is available, the requirement of fine-grained access control would lead to rule-explosion in the access control policy given the size of federated population in open systems. Additionally, this approach tightly couples authentication with authorization, and is therefore inflexible, and violates one of our design principles.

X-GTRBAC addresses this problem through the use of attribute-based (as opposed to identity-based) credential specification. We adopt a modular approach and allow independent specification of credentials used in authentication and authorization. The authenticating credential comprises of authentication information expressed in terms of user attributes which are used by the access control processor for role assignment. This idea is similar to the one used in [26]. However, unlike the work in [26], we do not require reliance on X.509 identity-based certificates to encode user authentication information. Instead, the user attributes may be supplied in any

mutually agreed format, such as an Attribute Statement in the SAML standard [13]. This supports the requirement for credential federation (See Section 3.6.2).

An authorization credential comprises of information about role attributes, role hierarchy, and role constraints. Examples of role attributes are time of day, system load, etc. [27]. They are used by the access control processor for controlling assignment of users and permissions to roles. Role hierarchy provides a means of privilege inheritance, and hierarchical role definitions can be applied to extend or specialize existing policies. Role constraints restrict the enabling, activation and delegation (see Section 3.6.3) of roles to allow fine-grained access management.

The authenticating and authorization credentials used in X-GTRBAC are included in an XML User Sheet (XUS) and an XML Role Sheet (XRS) respectively. The definitions of the credential types used in the XUS are provided through the use of an XML Credential Type Definition (XCredTypeDef) sheet. The top-level X-Grammar syntax of XUS, XCredTypeDef, and XRS is shown in Table 3.1.

The credential specification in X-GTRBAC facilitates a combination of rule-based role assignment and role-based authorization (See Section 3.3.3). Our approach allows fine-grained access control while avoiding rule-explosion in the policy since users are assigned to roles and access rules are specified at per-role rather than per-user level.

Table 3.1: X-Grammar: User & Role Definition

```
<!-- XML User Sheet>     ::=
<XUS xus_id = (xs:id) >
 <User user_id = (xs:id) >
    <CredType cred_type_id=(xs:idref)
           cred_type_name= ( xs:name) >
    [<!Header>]
      <!-- Credential Expression>
  </CredType>
 </User>
</XUS>

<!-- Credential Type Definitions > ::=
  <XCredTypeDef xctd_id = (xs:id) >
    <CredTypeDef cred_type_id = (xs:id)
cred_type_name= (xs:name) >
<!-- Attribute List>
    </CredTypeDef >
 </XCredTypeDef>
```

```
<!-- XML Role Sheet>::=
<XRS xrs_id = ( xs:id) >
 <Role role_id = ( xs:id)
    role_name = ( xs:name)>
[<!-- Cred Type>]
    [<!(En|Dis)abling Constraint>]
    [<![De]Activation Constraint>]
[<!Delegation Constraint>]
    [<JuniorRoleId> ( xs:idref)
                   </JuniorRoleId>]
    [<SeniorRoleId> ( xs:idref)
                   </SeniorRoleId>]
    {<SSDRoleSetId> ( xs:idref)
                   </SSDRoleSetId>}*
    {<DSDRoleSetId> ( xs:idref)
                   </DSDRoleSetId>}*
{<!LinkedRoleID>}*
 </Role>
</XRS>
```

3.3.2 Constraints

Constraints are essential to the expressiveness of an access control language. Specification of semantic and contextual constraints is vital to support the enforcement of integrity principles and resource provisioning contracts in a federated system. As motivated earlier, enforcing expressive constraints in a decentralized manner involves maintaining prohibitive amounts of state information and introduces significant complexity. Additionally, adapting the constraints according to on-demand

changes in integrity requirements and contractual obligations within a federation requires a specification format that facilitates such adaptation.

Most well-known distributed authorization schemes [24–26, 28] do not cover the requirements of constraint specification and enforcement as required for access management in federated systems. The approaches described in [24, 25] tightly couple resource-specific authorization constraints with the identity-information. This method of constraint specification is clearly inflexible to allow on-demand adaptation of constraints; doing so would require issuance of a new credential for the affected users since their identity is bound to the authorization. Additionally, constraints in the work presented in [24,25] are inadequate to capture semantic integrity constraints, such as SoD, in a federated system since doing so at the user level would require prohibitive amount of state information to be maintained. In comparison, enforcing SoD at the granularity of role is more manageable and one has to include in the constraint definition only the roles, as opposed to all permissions, that the user may have access to. The support for contextual constraints based on temporal or other environmental attributes is also limited in the work presented in [24, 25], since these approaches do not have a formal temporal model, and rely on underlying operating system primitives to enforce temporal constraints. The approaches presented in [26, 28] are based on basic RBAC and do not support specification of contextual constraints.

X-GTRBAC supports a variety of temporal and non-temporal constraint categories to adequately capture the access management requirements in federated systems. The temporal constraint specification in X-GTRBAC is primarily based on Generalized Temporal Role Based Access Control (GTRBAC) model [29]. GTRBAC is a mechanism using temporal logic to express a diverse set of fine-grained temporal constraints in an RBAC environment. The temporal constraint categories supported by GTRBAC include periodicity, interval, and duration constraints which can be used to constrain the period, interval and duration, respectively, of user-to-role and permission-to-role assignments. Another category is that of trigger-based

constraints, which can be thought of as condition-action rules. As the name implies, trigger-based constraints are used to condition the occurrence of an event on another. Moreover, X-GTRBAC also captures the SoD constraint among roles to capture integrity requirements. Both static SoD (SSD) and dynamic SoD (DSD) constraints are supported. Capturing these constraints at the role level helps reduce state information needed to enforce the constraints. X-GTRBAC supports modular specification of the SoD and temporal constraints on roles [30]. The modular approach allows independent specification of SoD and temporal constraint definitions which can then be imported into the policy through the use of XML namespaces. Specification of constraints separate from the policy allows reusable constraint definitions that can be used across multiple policies. Additionally, constraint definitions may be changed at one place without requiring change to all dependent policies, facilitating flexible adaptation.

X-GTRBAC additionally supports the specification of contextual constraints based on non-temporal attributes, usually associated with a role [27]. Contextual constraints on role attributes can be used in addition to temporal constraints to support finer granularity of control on user-to-role and permission-to-role assignments. The top-level X-Grammar syntax of the SoD and the temporal constraint definitions are shown in Table 3.2, respectively. The SoD constraints are captured through the use of SSD and DSD role sets which include the roles in static and dynamic SoD respectively, and references to these sets are included with the role definition in XRS. The temporal constraints are captured through the use of temporal expressions of GTRBAC and references to these expressions are included in assignment policy (Table 3.3).

Temporal constraints are of particular relevance to federated resource provisioning because it requires a set of fine-grained temporal constraints to adequately ensure resource protection while also ensuring its availability per the contractual requirements. This set includes constraints that control the periodicity, interval and duration of resource accesses (i.e. permission assignments) during and across provisioning

sessions, in addition to trigger-based constraints that allow provisioning actions to be conditioned on related events. This represents a collection of stateful rules that can be configured in permission-to-role assignment policies. Doing so allows specification of usage-oriented resource protection policies to enforce usage control of federated resources.

Table 3.2: X-Grammar: Constraint Definitions

<!-- Separation of Duty Definitions> ::= <XSoDDef xsod_id = (xs:id) > <!SSDRoleSets> <!DSDRoleSets> </XSoDDef>	<!-- Definitions of Temporal Constraints>::= <XTempConstDef xtcd_id = (xs:id) > {<!Interval Expression>}* {<!-- Periodic Time Expression>}* {<!-- Duration Expression>}* </XTempConstDef>

3.3.3 Assignment Rules

An integral component of RBAC polices expressed using X-GTRBAC is the specification of rules for user-to-role and permission-to-role assignments. Rule-based assignment in RBAC policies provides a succinct declarative specification that is both scalable and flexible. It avoids the problem of rule-explosion since rules are specified at per-role (as opposed to per-user or per-resource) level. It is flexible since a declarative syntax allows rules to be modified without changing the application code.

As noted earlier, the authenticating credential contains user attributes which are used by the access control processor for role assignment to users, whereas the authorization credential contains role attributes which are used by the access control processor for permission assignment to roles. (Role attributes may be used in user-to-

role assignment too.) Additionally, a permission-to-role assignment policy may also include rules on resource attributes to allow specification of usage-oriented protection policy. Resource attributes capture semantic information (or meta-data) about types of resources, and avoid reliance on fixed resource locations. They also provide a mechanism for fine-grained permission assignment to roles based on precise resource characteristics.

The types and attributes of resources are defined through the use of an XML Resource Type Definition (XResTypeDef) sheet. The resources types available in the system are defined using XML Resource Type Sheet (XRTS). A permission in the system comprises of a resource belonging to a given type specified in terms of its attributes, and an associated operation, and is defined in an XML Permission Sheet (XPS). The resource type and its definitions must be imported by a domain (using XML namespaces) before using them in a permission definition. The top-level X-Grammar syntax of an XPS, XResTypeDef and XRTS is shown in Table 3.3.

Our assignment policy schema specifies a logical expression syntax for rule specification. It does not, however, impose any restriction on the attributes that may be used for composing these rules. The existence and type checking of the queried attribute shall be done in an application-specific manner. For instance, user attributes can be verified through appropriate attribute authorities stated in the authentication credential.

The assignment policies are specified in X-GTRBAC in an XML User to Role Assignment Sheet (XURAS) and XML Permission to Role Assignment Sheet (XPRAS). The top-level X-Grammar syntax of XURAS is shown in Table 3.3 (XPRAS is analogous). Note that these policies include references to temporal constraint definitions. For example, pt_expr_id references a periodic time expression defined using XTempConstDef in Table 3.2.

Table 3.3: X-Grammar: Permission & Assignment Rule

```
<!XML Permission Sheet>::=
<XPS xps_id = (xs:id) >
  <Permission perm_id = (xs:id)
  [prop= (noprop|first_level|cascade)] >
  <Object res_type_id=(xs:idref) >
          {<!Attribute >}*
  </Object>
  <Operation> (saml:Action) </Operation>
 </Permission>
</XPS>

<!-- XML Resource Type Definitions>::=
<XResTypeDef xrtd_id = (xs:id) >
    <ResTypeDef res_type_id = (xs:id)
              res_type_name = (xs:name)>
{<!Attribute List>}*
 </ResTypeDef>
</XResTypeDef>

<!-- XML Resource Type Sheet>::=
<XRTS xrts_id = (xs:id) >
    <ResType res_type_id = (xs:idref)
res_type_name = (xs:name)>
{<!Attribute>}*
 </ResType>
</XRTS>
```

```
<!-- XML Predicate Function
Definitions>::=
 <XPredFuncDef xpfd_id = (xs:id) >
    <Function func_id = (xs:id)
func_name= (xs:name)
return_type= (xs:anyType)>
<!Parameter List>
    </Function>
 </XPredFuncDef>

<!-- XML User-Role Assignment Sheet>::=
<XURAS xuras_id = ( xs:id) >
  <URA ura_id=(xs:id) role_id=(xs:idref)>
   <AssignUsers>
     <AssignUser user_id=(xs:idref)>
 <AssignConstraint[op =(AND|OR|NOT|XOR)]>
       //no opcode defaults to AND
 <AssignCondition cred_type_id=(xs:idref)
 [pt_expr_id=(xs:idref)
| d_expr_id=(xs:idref)] >
  <LogicalExpr [op = (AND|OR|NOT)]>
//no opcode defaults to AND
     {<!-- Predicate>}+
        </LogicalExpr>
 </AssignCondition>
     </AssignConstraint>
  </AssignUser>
 </AssignUsers>
 </URA>
</XURAS>
```

A key feature of our rule specification format is that it allows combing rules from multiple sources to facilitate evaluation of multiple credentials. An assignment rule consists of an assignment constraint, which comprises of multiple assignment conditions. Each assignment condition contains a set of logical expressions to encode rules on a given credential type. The assignment rules consist of Boolean predicate functions. A set of available predicate functions is defined through the use of an XML Predicate Function Definition (XPredFuncDef). The predicate definitions must be imported by a domain (using XML namespaces) before using a predicate function in an assignment rule. The top-level X-Grammar syntax of an XPredFuncDef sheet is shown in Table 3.3.

Our logical expression syntax allows multiple logical expressions to be combined together in an appropriate rule combining mode using Boolean connectives. The modes supported by the specification language are AND (all rules must be true), OR (at least one rule must be true), and NOT (no rule must be true). Mutiple levels of nesting are supported, each under a distinct mode, to allow a fine granularity of rule specification. An assignment condition is satisfied if all of its included logical expressions are satisfied according to the respective mode. The results of evaluating multiple assignment conditions within an assignment constraint are combined similarly. Role assignment occurs as a consequence of an assignment constraint being satisfied.

We note that the AND mode essentially implements deny-overrides, whereas OR mode implements permit-overrides. The NOT mode allows one to condition a role assignment based on negation. This is useful in instances when it is easier to express exclusion rather than inclusion criterion for membership in a role. Although negation is allowed in the body, it is not allowed in the consequence of a rule. This prevents contradictory rule sets to exist in the specification. This property is helpful when combining rules aimed at a given consequence, since one can always be sure that new rules will not clash with existing rules in the policy.

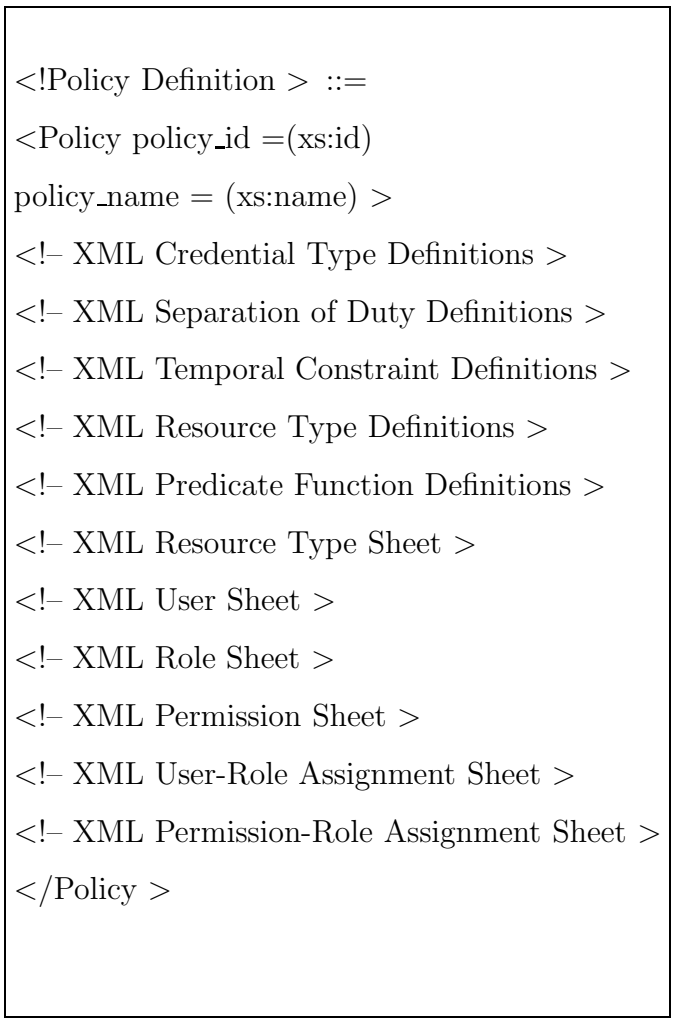

Fig. 3.1. The overall X-GTRBAC policy

3.4 Policy Composition

An overall X-GTRBAC policy is composed from these individual policy components as given in Figure 3.1. The complete policy grammar is provided in Appendix A.

3.5 Policy Meta-model

In this section, we provide a UML-based meta-model of our policy language. UML is a standardized modeling specification, and UML models for other types of security policies (such as IPSec [31] for network communication) have also been developed by the community [22]. A UML-based meta-model of our language thus is an important aspect of our policy engineering approach. On the one hand, it provides a set of standardized components to compose an access management policy which can readily be translated into implementation code using well-known UML-compatible tools [21]. On the other hand, it also allows the policy to be applied with other types of security policies in an interoperable manner in a federated system, since the policies in the system will be instances of a single meta-model.

Our meta-model is designed using a UML profile. A UML profile is an extension of the UML to include specifications not captured directly by UML. It has three key items: stereotypes, tags, and constraints. A profile provides the definition of these items, and explains how they extend the UML in a particular domain. A stereotype is assigned to a UML construct (class, attribute, association) that is modified by the extension profile. The stereotype can be further specified by adding tags (or properties) that refine its meaning or impact on the model within the domain of the profile. Since our target domain is XML, we require a UML profile for XML Schema that defines the stereotypes and tags needed for appropriate translation to XML Schema representation of our policy language.

Our task is facilitated by adopting an existing UML profile for XML Schema (UPX) [21]. The UPX allows the specification of stereotypes and tags for XML Schema during the UML design process and the model thus developed can be translated into the XML Schema syntax through the use of the XML Metadata Interchange (XMI) technology [21]. The translation to the XML Schema syntax allows support for validation of policy documents for type consistency. XML-based policy

documents can be validated against the XML Schema specification of our language using well-known schema validation tools [21].

3.5.1 UML Package

The meta-model for our policy language is represented using a UML package which consists of standard UML constructs including a set of extensions defined as stereotypes and tags in the UPX. Table 3.4 summarizes and explains the meaning of the UML constructs and extensions used to represent the various policy components defined in the X-Grammar specification presented in this chapter (consistent with the standard notation, we denote a UML stereotype in angled brackets <<>>).

The UML class diagrams for the top-level components of the X-GTRBAC policy given in Figure 3.1 are shown in Figure 3.2. The complete UML-based meta-model of the language is given in Appendix B.

Table 3.4: Summary and explanation of teh UML constructs and UPX stereotypes

Policy Component	**UML Construct**	**UPX Stereotype(s)**	Meaning
Policy root	UML Package	<<XSDschema>>	The root of the policy is represented using a UML package with a UPX stereotype.
X-Grammar Non-Terminal	UML Class	<<XSDcomplexType>>	An X-Grammar element that exists as a non-terminal defines a new XML Schema complex type. It is represented in UML using a class with a UPX stereotype defined on it.

Continued on next page

Policy Component	**UML Construct**	**UPX Stereotype(s)**	Meaning
X-Grammar Terminal Only	UML attribute	<<XSDelement>>	An X-Grammar element that exists only as a terminal defines an XML Schema element. It is represented in UML using a UML attribute with a UPX stereotype defined on it to map the UML attribute to an XML Schema element.
X-Grammar Attribute	UML attribute	<<XSDattribute>>	An X-Grammar attribute is represented in UML using a UML attribute with a UPX stereotype defined on it.
Inclusion within X-Grammar Non-Terminal	UML association	<<XSDcomplexType>> <<XSDelement>>	The inclusion of non-terminals and terminals within an X-Grammar non-terminal is represented in UML using a UML association with a UPX stereotype defined on it. An included non-terminal or terminal is mapped in XML Schema as an included complex type or element, respectively.

Continued on next page

Policy Component	**UML Construct**	**UPX Stereotype(s)**	Meaning
Constant list	UML enumeration	<<XSDsimpleType>>	A list of constants indicating valid values of an element defines a new XML Schema simple type. It is represented in UML using a UML enumeration with a UPX stereotype defined on it.

3.5.2 UML to XML Schema

The mapping of UML model (expressed through the UPX) to XML Schema subsequently allows the policy specification to be translated into implementation code using platform-specific binding mechanisms. In this context, the UML model serves two key purposes: it provides a layer of abstraction to present a simple interface to the system designer to construct policies, and it also encapsulates the implementation details from the system designer.

3.6 Policy Engineering with X-Federate

In this section, we discuss the application of our policy engineering methodology for federated access management. The methodology suggests the use of UML Package (see Table 3.4) which comprises the definitions of the federated schema for the various policy components that can be used by all federating sites to encode their access management requirements. In other words, each federating site represents an administrative domain (or domain for short), which is a unit of administrative authority. A federated system is then a multi-domain environment where each domain is responsible for managing the users and resources under its administrative

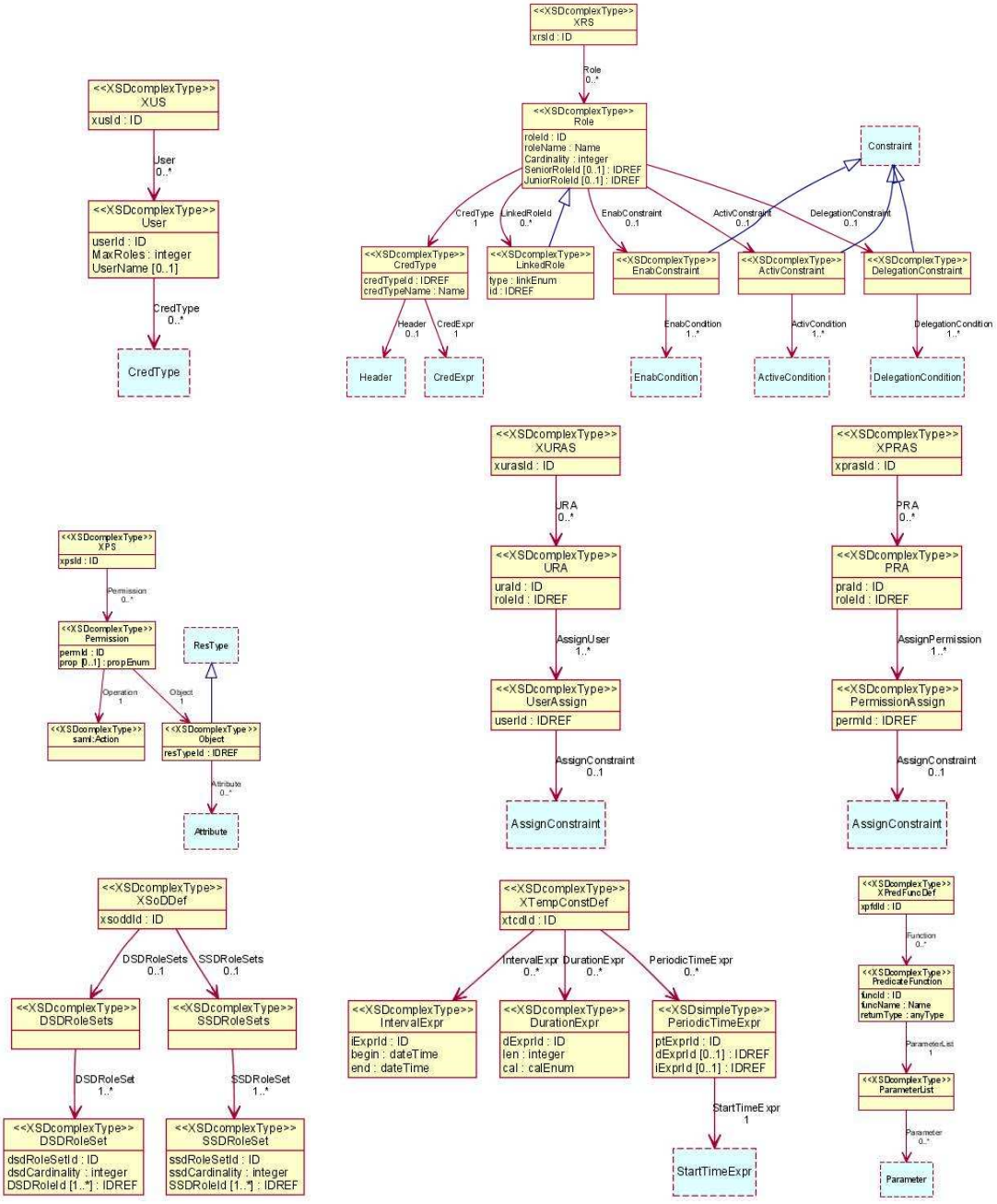

Fig. 3.2. The UML class diagrams top-level components of the X-GTRBAC policy (in a row major order): (a) XUS (b)XRS (c)XPS (d)XURAS (e)XPRAS (f)XSoDDef (g)XTempConstDef (h)XPredFuncDef

control [27]. Each domain adheres to the definitions provided by the UML Package, and domains can therefore compose interoperable policies while also specifying their own access control requirements on their resources.

Using the meta-model of our policy specification, we now highlight the salient features of X-FEDERATE that enable the solution to access management challenges in federated systems outlined in this thesis.

3.6.1 Decentralized Administration

The requirement for local autonomy, and hence decentralization of administrative control, in federated systems complicates developing access control policies because the policy needs to be specified in terms of a diverse, unseen user pool requiring granular and differentiated access to a diverse set of resources located anywhere across the federation. As discussed in Table 1.1, traditional approaches for policy design are inadequate for this purpose because the federating sites have no knowledge of user identities or resource locations. Additionally, the requirement for fine-grained access control would cause scalability concerns.

The policy engineering methodology outlined in this chapter helps alleviate this problem. In particular, the package contains the class CredType (see Figure 3.2(a)) which specifies the structure of authenticating and authorization credentials for users and roles based on their attributes, as discussed in Section 3.3.1. A credential thus defined is included in the definition of classes XUS or XRS (See Figure 3.2(a) and (b)). Rules can therefore be defined on these credential attributes to allow role assignment for users without knowledge of their identities. Additionally, the class XPS (See Figure 3.2(c)) includes specification of resource attributes which allows resource assignment rules to be defined in terms of attributes without knowledge of resource locations. Shared definitions of credential and resource attributes can thereby promote interoperability between multiple domains.

Delegation of responsibilities is essential to scalable decentralization. Delegation in federated systems is captured through some form of trust relationships [24]. In X-GTRBAC, the notion of delegation is elegantly captured through the use of role hierarchies: a junior role inherits all privileges of a senior role. An optional Delegation Constraint may be used in the role definition (See Table 3.1) to limit the extent of delegation (in terms of time and associated privileges); unrestricted delegation is otherwise assumed. This role-based delegation serves as the basis of trust in creating role mappings across multiple domains for federated information sharing. (Each domain in the federation publishes its role definitions to a well-known repository which may be imported for establishing appropriate role mappings using XML namespaces.) For instance, an Employee role from domain B that is mapped as junior to a Manager role in domain A would be allowed to exercise the Manager-level privileges in domain A per its delegation policy, without requiring explicit knowledge of domain Bs access control policy. The UML package of our meta-model defines the Junior Role ID, Senior Role ID, and Linked Role ID elements as part of class XRS to indicate the role delegation relationship. The latter element is used to establish delegation relationship between roles not related to each other through hierarchical relationship.

Delegated administration requires access to a local compliance checker that can compute correspondence between mapped roles with respect to the local domain policies. The use of a compliance checker to ensure compliance of federated requests with local policies is a recognized mechanism for preserving local autonomy in distributed systems [32]. In our X-FEDERATE prototype, the compliance checker is incorporated into an authorization engine residing in each domain. It internally maintains a domain-specific mapping from the foreign (i.e. federated) roles to local roles according to the delegation policies of the local domain.

3.6.2 Credential Federation

The credential specification in a federated system must support federation requirement, as outlined in Table 1.1. Many existing distributed authorization schemes [24–26, 28] do not address this requirement due to inherent limitation of their credential specification formats, as discussed in Section 3.3.1.

The credential specification in the UML package of our meta-model allows credential federation through the use of interoperable protocols. In addition to the attributes associated with a CredType, each credential also has an optional Header element (See Appendix B) to capture the security relevant attributes necessary to allow credential information to be exchanged between multiple federating sites. The package currently allows one to specify the structure of the credentials using security attributes defined as per the SAML standard [13] . SAML expresses authentication information from various sources, such as X.509 Attribute Certificates, Kerberos tickets or passwords, as assertions. We have developed a SAML profile of the X-GTRBAC langauge to provide a SAML-compliant format for authenticating credentials. We employ appropriate translation mechanisms for SAML assertions to be used with X-GTRBAC language syntax. Since our rule specification supports combing rules from multiple sources, this allows use of our specification in situations when multiple policies are necessary to evaluate the request of a federated user. Chapter 5 of this thesis is devoted to the detailed treatment of the SAML profile of X-GTRBAC and its use in credential federation in our framework.

Credential federation includes support for single sign on (SSO). SSO enables persistent authorization for federated users within a single login session. Our framework supports SSO through the use of digitally signed SAML statements that capture an authorization decision already issued by a domain corresponding to a user request. This decision statement can subsequently be reused by the user at a domain within the federation without getting re-authenticated, subject to the acceptance of the digital signature.

3.6.3 Constraint Specification

A PBM approach designed for federated systems must be flexible and adaptable enough to meet the integrity requirements and contractual obligations within a federation. This requires a modular mechanism for constraint specification that allows constraint definitions to be updated without affecting the remaining policy. Supporting semantic and contextual constraints in the access control policy specification is a daunting task. The UML package of our meta-model consists of two distinct mechanisms for specification of constraints, including the use of class XSoDDef (See Figure 3.2(g)) for defining SoD role sets and class XTempConstDef (See Figure 3.2(h)) for defining temporal constraint expressions. The SoD role sets are included in the definition of XRS class to apply SoD constraints, whereas the temporal constraint expressions may be associated with a set of constraint classes. The classes of constraints available in the package are all derived from a base Constraint class (See Appendix B). These classes include AssignConstraint (for constraint on assignments), ActivConstraint (for constraint on role activations), EnabConstraint (for constraint on role enabling), and DelegationConstraint (for constraint on role delegation). In addition to temporal constraint expressions, non-temporal constraint expressions can also be associated with a constraint by directly including an instance of the class LogicalExpression (See Appendix B) in the definition of a constraint. This mechanism of constraint specification allows modular construction of constraint definitions, and promotes reusability and flexibility in their use during policy specification. A LogicalExpression comprises of predicate functions defined as per the class XPredFuncDef (See Figure 3.2(f)). The AssignConstraint is included in the definition of the classes XURAS and XPRAS (See Figure 3.2(d) and (e)) which specify the rules for user-to-role and permission-to-role assignments respectively. The ActivConstraint, EnabConstraint, and DelegationConstraint are included with the definition of roles in XRS.

In this chapter, we presented X-FEDERATE, our policy engineering framework. We discussed the details of the policy specification language, the UML-based meta-model, and the policy engineering methodology. The next section will discuss the administrative model for the X-FEDERATE framework.

4. ADMINISTRATION MODEL

There has been a growing interest in administration models built on RBAC and related schemes [33, 34]. However, existing models are inadequate for administering policies for federated systems because they do not support the requirements for federated information sharing outlined in Chapter 1. In particular, they do not allow decentralizing administration tasks and specification of constraints during policy administration.

In this chapter, we present an administration model for the X-FEDERATE framework that is designed to address this need. The administration model, called X-GTRBAC Admin, is designed to allow decentralized administration of X-GTRBAC policies in the presence of constraints. We introduce the formal specifications of administrative concepts and constraints needed to provide the administrative support for the X-GTRBAC policies.

4.1 Background

In this section, we provide some background needed to discuss the administrative concepts related to the X-GTRBAC Admin model.

4.1.1 RBAC and GTRBAC

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

Definition 4.1.1 *(**RBAC Model**) [19] The RBAC model consists of the following components:*

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

A session $s_i \in$ Sessions has the permission of all roles r' junior to roles activated in the session, i.e. $\{p|((p,r) \in PA \vee (p,r') \in PA) \wedge r \in role(s_i) \wedge r \geq r'\}$

The RH relation is one of the most important aspects of RBAC for its use toward simplifying authorization management. The original RBAC model supports only inheritance or usage hierarchy, which allows the users of a senior role to inherit all permissions of junior roles. In order to preserve the principle of least privilege, RBAC model has been extended to include activation hierarchy which enables a user to activate one or more junior roles without activating senior roles [35]. An inheritance-activation hierarchy can be defined on roles by composing inheritance and activation hierarchies [36]. In this thesis, we do not concern ourselves with the advanced semantics of role hierarchies, and use the $\geq$ relation defined in the RBAC standard.

The GTRBAC model [29] incorporates a set of language constructs for the specification of various temporal constraints on roles, including constraints on role enabling, role activation, user-to-role assignments, and permission-to-role assignments. In particular, GTRBAC makes a clear distinction between role enabling and role activation. An enabled role indicates that a user can activate it, whereas an activated role indicates that at least one subject has activated a role in a session. The notion of separate activation conditions is particularly helpful in large enterprises, with several hundred users belonging to the same role, to selectively manage role activations at the individual user level.

As briefly noted in Chapter 3, the temporal framework in GTRBAC model allows the specification of the following constraints, events, and expressions:

1. *Temporal constraints on role enabling/disabling*: These constraints allow one to specify the time intervals during which a role is enabled. It is also possible to specify a role duration.

2. *Temporal constraints on user-to-role and permission-to-role assignments*: These are constructs to express either a specific interval or a duration in which a user or a permission is assigned to a role.

3. *Activation constraints*: These allow one to specify how a user should be restricted in activating a role. These include, for example, specifying the total duration for which a user is allowed to activate a role, or the number of users that can be allowed to activate a particular role.

4. *Run-time events*: A set of run-time events allows an administrator to dynamically initiate GTRBAC events, or a user to issue activation requests.

5. *Constraint enabling expressions*: GTRBAC includes events that enable or disable duration constraints and role activation constraints.

6. *Triggers*: Triggers allow one to express dependency among GTRBAC events as well as capture the past events and define future events based on them.

7. *Periodic Time Expression*: A periodic expression (PTE) is represented by pairs <[begin, end], P>, where P is a periodic expression denoting an infinite set of periodic time instants, and [begin, end] is a time interval I denoting the lower and upper bounds that are imposed on instants in P. Formally, P is expressed as follows:

   Definition 4.1.2 *(**Periodic Expression**) [29] Given calendars* $C_d, C_1, \cdots, Cn$, *and time occurrences* $O_1, \cdots, O_n$, *a periodic expression P is defined as:*

$$P = \sum_{i=1}^{n} O_i.Ci \rhd x.C_d$$

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

The formalism for periodic expressions is based on the notion of calendars. A calendar is defined as a countable set of contiguous intervals, numbered by integers called indexes of the intervals. Symbol $\rhd$ separates the first part of the periodic expression that identifies the set of starting points of the intervals it represents, from the specification of the duration D of each interval in terms of calendar C_d. For example, $all.Years + \{3, 7\}.Months \rhd 2.Months$ 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 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 $1.C_n$.

The temporal constraint expressions in GTRBAC are summarized in Table 4.1.

Table 4.1: Temporal constraint expressions of GTRBAC

Constraint Categories	**Events**	**Expression (E:event, C:constraint)**
Enabling	Role enabling	$(I, P, D, enable/disable\, r)$
Activation	Role activation	<!– only occurs as a run time event >
Assignment	User-to-role assignment	$([I, P, D], assign_u/deassign_u\, r\, to\, u)$
	Permission-to-role assignment	$([I, P, D], assign_p/deassign_p\, p\, to\, r)$
Trigger	<!– any triggering event >	$E_1, \cdots, E_n, C_1, \cdots, C_k \rightarrow E\, after \Delta t$
Run-time	User activation request	$(s : (de) activate\, r\, for\, u\, after \Delta t))$
	Administrator action	$(assign_u/de - assign_u\, r\, to\, u\, after \Delta t)$ $(enable/disable\, r\, after \Delta t)$ $(assign_p/de - assign_p\, p\, to\, r after \Delta t)$ $(enable/disable\, C\, after \Delta t)$

4.2 X-GTRBAC Admin

We now introduce X-GTRBAC Admin, the administrative extension to the GTRBAC model, and present the formal specification for its components. A key feature of model is that it is designed to handle not only the temporal constraints of the GTRBAC model, but also non-temporal constraints, which are represented in the model as logical expressions over Boolean predicates. Additionally, the model supports attribute-based user-to-role and permission-to-role assignments. These features of the model make it suitable to meet the constraint specification and decentralized policy administration requirements for federated information sharing, as outlined in Table 1.1.

In order to include the administrative concept, the model is extended to include the specification of an administrative domain (Admin Domain), an administrative role (Admin Role), and an administrative permission (Admin Permission).

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

Admin Role: An Admin Role is an administrative role authorized to manage policy administration tasks within a given Admin Domain. This authority is given by a set of associated Admin Permissions as defined in the model. A pool of administrative users is selected by the security administrator for assignment to Admin Roles, where such assignment may be based on evaluation of user attributes and applicable constraints, as in the case of regular roles, to allow fine-grained policy assignments. The Admin Roles are related according to a partial order which defines an administrative role hierarchy.

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

4.2.1 Formal Model

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

Definition 4.2.1 *(**Core Components**): The X-GTRBAC Admin model consists of the following core components:*

- *AD, a set of Admin Domains*
- *AU, a set of administrative users, $AU \subseteq Users$*
- *RR, a set of regular roles, $RR = Roles$*
- *ER, a set of enabled regular roles, $ER \subseteq RR$*
- *RP, a set of regular permissions, $RP = Permissions$*
- *AR, a set of Admin Roles*
- *AP, a set of Admin Permissions*
- *The association of regular roles with domains, called role instances[1], is defined as*

$$RR_D \subseteq AD \times RR = \{(ad, rr) | ad \in AD, rr \in RR\}$$

- *The association of regular permissions with domains, called permission instances, is defined as*

$$RP_D \subseteq AD \times RP = \{(ad, rp) | ad \in AD, rp \in RP\}$$

- *The association of an Admin Domain with an Admin Role is defined as*

$$AR_D \subseteq AD \times AR = \{(ad, ar) | ad \in AD, ar \in AR\}$$

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

- *The association of Admin Permissions with Admin Domains is defined as*

$$AP_D \subseteq AD \times AP = \{(ad, ap) | ad \in AD, ap \in AP\}$$

- *The set $ATTR_x$ of attribute-value pairs for a regular user, role or permission instance x; an attribute value pair AVP is defined as a tuple (name, value), where both name and value are constants.*

- *The set SOD of regular roles in Separation of Duty (SoD); a collection SODS of SOD sets may exist to define fine-grained SoD constraints.*

- *The set CR of constraints defined according to Definition 4.2.2. CR may be an empty set.*

- *The set CONST is a collection of constants of simple data types (as string, integer, etc.).*

- *$AUA \subseteq AU \times AR$, the user assignment relation, that assigns administrative users to Admin Roles*

- *$APA \subseteq AP \times AR$, the permission assignment relation, that assigns Admin Permissions to Admin Roles*

- *$RM \subseteq RR \times RR$, the role mapping relation, that maps a regular role from one domain to a regular role from another domain*

- *A partially ordered regular role hierarchy $RRH \subseteq RR \times RR$; $RRH = RH$*

- *A partially ordered Admin Role hierarchy, $ARH \subseteq AR \times AR$*

- *A partially ordered Admin Domain hierarchy $DH \subseteq AD \times AD$; $(ad_x \geq ad_y) \in DH$ implies that domain ad_x dominates domain ad_y in the hierarchy.* □

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

Definition 4.2.2 *(**Constraint Expression**): A constraint expression in X-GTRBAC Admin is one of the following two kinds:*

- *A periodic time expression PTE defined as per the GTRBAC model (See Definition 4.1.2).*
- *A logical expression using the usual $\vee$ and $\wedge$ operators on 3-tuples of the form $(y, \omega, \delta(p_1, .., p_n))$ where δ is a parameterized administrative function (as defined in Table 4.2), p_i and y are a member of the set ($RR \cup AR \cup RP \cup AP \cup AD \cup Users \cup CONST$), and $\omega = \{=, \neq, \geq, \leq, \in\}$* $\square$

Example 1: $PTE_{EX1} =< P, [2005.Years, 2005.Years] >, P = all.Years + 1, 4, 7, 10\text{-}.Months + 1.Weeks \rhd 1.Weeks$ represents a periodic time expression that defines a periodic time P which is a set of intervals starting at the same instant as the first week of every quarter of every year, and having a duration of one week. Additionally, the PTE is valid within the interval bounded by the start and end of year 2005. $\square$
Example 2: $LE_{EX2} = (NewYork, =, hasCredAttrValue(PermittedPhysicianPx, location))$ represents a logical expression that includes evaluation of a predicate function hasCredAttrValue (hCAV) that compares the value of the location attribute of the role PermittedPhysicianPx using the equality operator with the expected value of NewYork. $\square$

Based on these concepts, a PE and a policy rule are formally defined below.

Definition 4.2.3 *(**Predicate Expression**): A predicate expression (PE) is a Boolean expression involving a set P of predicates, such that every $p \in P$ is defined to evaluate a constraint expression, i.e., p: $\rightarrow$ CR true, false. A constraint associated with a predicate evaluates to true in one of the following two ways:*

- *It is a PTE, and the associated interval and periodicity conditions are satisfied, or*

- *It is a logical expression with clauses of the form* $(y, \omega, \delta(p_1, .., p_n))$*, and the expression is satisfied over the set of clauses. A clause evaluates to true if* y *compares with the return value of the function* δ *according to the comparison operator* ω*.* □

Example 3: $PE_{EX3} = p_1 \wedge p_2$ represents a predicate expression which evaluates to true if the constraint expression associated with p_1 is true and the one associated with p_2 is true. If p_1 is defined on PTE_{EX1} and p_2 is defined on LE_{EX2}, then PE_{EX3} is true if PTE_{EX1} evaluates to true and LE_{EX2} evaluates to true. □

4.2.2 Administrative Features

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

Table 4.2 summarizes the administrative functions provided by X-GTRBAC Admin. These functions can be used by the security administrators to obtain information about different components of the system, such as association of a role or permission instance with a domain, authority of an Admin Role over Admin Domains, association of an Admin Role with a regular role, values of role attributes, roles in separation of duty, users and permissions assigned to roles, and users who have activated certain roles. This information may be used as the basis of predicates in constraint expressions to restrict any administrative operation. An example of using review functions in constraints will be provided in Section 4.2.3.

Table 4.2: X-GTRBAC Admin Functions

Function	**Description**	**Formal Semantics**
$domain : (RR \cup AR \cup AP) \rightarrow AD$	Returns the domain of a role or permission instance.	$domain(x) = \{d \mid (d,x) \in RR_D \vee (d,x) \in AR_D \vee (d,x) \in AP_D\}$.
$has\_authority\_over : AR \rightarrow 2^{AD}$	Returns the set of all Admin Domains that an Admin Role has authority over. This set includes domains that are dominated by the domain of the Admin Role.	$has\_authoriy\_over(ar) = \{d \mid domain(ar) = d \vee domain(ar) \geq d\}$.
$administers : AR \rightarrow 2^{RR}$	Returns the set of all regular roles administered by an Admin Role. An Admin Role administers a regular role if both belong to the same domain.	$administers(ar) = \{rr \mid domain(ar) = domain(rr)\}$.
$has\_attribute\_value : (Users \cup RR \cup RP) \times CONST \rightarrow CONST$	For the user, role, or permission x, returns the value of the attribute identified by v.	$has\_attribute\_value(x,v) = \{AVP.value \mid AVP.name = v \wedge AVP \in ATTR_x\}$.
$in\_separation\_of\_duty : RR \rightarrow SODS$	For the role rr, returns the SOD set.	$in\_separation\_of\_duty(rr) = \{SOD \mid rr \in SOD\}$.
$assigned\_users : RR \rightarrow 2^{Users}$	For the role rr, returns the set of users assigned to the role.	$assigned\_users(rr) = \{u \mid (u,rr) \in UA\}$[2].

Continued on next page

[2]This function is modified to use qualified (i.e. regular) role instance. This definition supersedes that in Definition 4.1.1

Function	**Description**	**Formal Semantics**
$assigned\_permissions : RR \rightarrow 2^{Permissions}$	For the role rr, returns the set of permissions assigned to the role.	$assigned\_permissions(rr) = \{rp \mid (rr, rp) \in PA\}$[2].
$activated\_users : RR \rightarrow 2^{Users}$	For the role rr, returns the set of users active in the role.	$activated\_users(rr) = \{u \mid rr \in ER \wedge (\exists s_i \in Sessions \mid rr \in role(s_i) \wedge u \in user(s_i))\}$[3]

The administrative relations provided by X-GTRBAC Admin are given in Table 4.3. They are used to represent association between various components of the model, such as set of administrative users and Admin Roles, Admin Permissions and Admin Roles, and Admin Roles and regular roles. The relations involve the evaluation of a (possibly null) constraint expression that is used to determine the validity of the associated administrative operation, such as role assignment, enabling, or inter-domain role mapping.

Table 4.3: X-GTRBAC Admin Relations

Relation	**Description**	**Formal Semantics**
$can\_assign\_admin \subseteq AU \times AR_D \times CR$	The user assignment relation that a security administrator can use to assign administrative users to Admin Roles subject to the associated constraints.	$can\_assign\_admin(au, ar, c) \leftrightarrow (c = null \vee evaluate(c) = true)$.
$can\_deassign\_admin \subseteq AU \times AR_D \times CR$	analogous to can_assign_admin.	

Continued on next page

[3]Note that the roles that may have been activated belong to the enabled regular role set, i.e. only enabled roles can be activated

Relation	**Description**	**Formal Semantics**
can_assignp- *_admin-* $\subseteq AR_D \times AP_D \times CR$	The permission assignment relation that that a security administrator can use to assign Admin Permissions to Admin Roles if the permission and the role belong to the same domain, and subject to the associated constraints.	$can\_assignp\_admin(ar, ap, c) \leftrightarrow domain(ar) = domain(ap) \wedge (c = null \vee evaluate(c) = true)$.
can_deassignp- *_admin-* $\subseteq AR_D \times AP_D \times CR$	analogous to *can_assignp_admin*.	
can_assign $\subseteq$ $AR_D \times RR_D \times CR$	The user assignment relation, that authorizes an Admin Role (or its senior) to assign a regular user to a regular role (or its junior) if the Admin Role has the *assign* Admin Permission, and subject to the associated constraints.	$can\_assign(ar, rr, c) \leftrightarrow (rr \in administers(ar) \wedge (ar, assign) \in APA \wedge evaluate(c) = true) \vee (\exists ar' \in AR \mid ar \geq ar' \wedge can\_assign(ar', rr, c)) \vee (\exists rr' \in RR \mid rr' \geq rr \wedge can\_assign(ar-, rr', c))$.
can_deassign $\subseteq$ $AR_D \times RR_D \times CR$	analogous to *can_assign*.	
can_enable $\subseteq$ $AR_D \times RR_D \times CR$	The role enabling relation, that authorizes an Admin Role (or its senior) to enable a regular role (or its junior) if the Admin Role has the *enable* Admin Permission, and subject to the associated constraints.	$can\_enable(ar, rr, c) \leftrightarrow (rr \in administers(ar) \wedge (ar, enable) \in APA \wedge evaluate(c) = true) \vee (\exists ar' \in AR \mid ar \geq ar' \wedge can\_enable(ar', rr, c)) \vee (\exists rr' \in RR \mid rr' \geq rr \wedge can\_enable(ar, rr'-, c))$.
can_disable $\subseteq$ $AR_D \times RR_D \times CR$	analogous to *can_enable*.	

Continued on next page

Relation	**Description**	**Formal Semantics**
$can\_assignp \subseteq AR_D \times RR_D \times CR$	The permission assignment relation, that authorizes an Admin Role (or its senior) to assign a regular permission to a regular role (or its junior) if the Admin Role has the *assignp* Admin Permission, and subject to the associated constraints.	$can\_assignp(ar, rr-, c) \leftrightarrow (rr \in administers(ar) \wedge (ar, assignp) \in APA \wedge evaluate(c) = true) \vee (\exists ar' \in AR \mid ar \geq ar' \wedge can\_assignp-(ar', rr, c)) \vee (\exists rr' \in RR \mid rr' \geq rr \wedge can\_assignp(ar, rr', c))$.
$can\_deassignp \subseteq AR_D \times RR_D \times CR$	analogous to *can_assignp*.	
$can\_map \subseteq AR_D \times RR_D \times RR_D \times CR$	The role mapping relation, that allows an Admin Role to map a regular role from one domain to a regular role from another domain if the Admin Role has the *map* Admin Permission, and subject to the associated constraints	$can\_map(ar, r1, r2, c) \leftrightarrow (r1, r2 \in administers(ar) \wedge domain(r1) \neq domain(r2) \wedge (ar, map) \in APA \wedge evaluate(c) = true) \vee (\exists ar' \in AR \mid ar \geq ar' \wedge can\_map(ar', r1-, r2, c)) \vee (\exists r1', r2' \in RR \mid r1' \geq r1 \vee r2' \geq r2 \wedge can\_map(ar, r1'-, r2', c))$.
$can\_unmap \subseteq AR_D \times RR_D \times RR_D \times CR$	analogous to *can_map*.	

Table 4.4 summarizes the administrative operations provided by X-GTRBAC Admin. These include operations for role assignment, enabling, and inter-domain role mapping[4].

[4]The reverse operations (such as deassign_role) are obtained by using the corresponding predicate (such as can_deassign) from Table 4.3 and replacing with -.

Table 4.4: X-GTRBAC Admin Operations

Operation	**Description**	**Formal Semantics**
$assign\_admin\_role(au \in AU, ar \in AR, c \in CR)$	The user assignment operation that adds a new tuple to the AUA relation.	$if can\_assign\_admin(au, ar, c)$ $then AUA = AUA \cup (au, ar)$.
$assign\_admin\_permission(ar \in AR, ap \in AP, c \in CR)$	The permission assignment operation that adds a new tuple to the APA relation.	$if can\_assign\_adminp(ar, ap, c)$ $then APA = APA \cup (ar, ap)$.
$assign\_role(ar \in AR, rr \in RR, u \in Users, c \in CR)$	The user assignment operation that adds a new tuple to the UA relation.	$if can\_assign(ar, rr, c)$ $then UA = UA \cup (u, rr)$.
$enable\_role(ar \in AR, rr \in RR, c \in CR)$	The role enabling operation that adds a new element to the ER set.	$if can\_enable(ar, rr, c)$ $then ER = ER \cup (rr)$.
$assign\_permission(ar \in AR, rr \in RR, rp \in RP, c \in CR)$	The permission assignment operation that adds a new tuple to the PA relation.	$if can\_assignp(ar, rr, c)$ $then PA = PA \cup (rr, rp)$.
$map\_role(ar \in AR, r1 \in RR, r2 \in RR, c \in CR)$	The role mapping operation that adds a new tuple to the RM relation.	$if can\_map(ar, r1, r2, c)$ $then RM = RM \cup (r1, r2)$

Definition 4.2.4 *(**Policy Rule**): A policy rule is a statement involving an administrative operation as defined in Table 4.4), the execution of which is subject to a predicate* $p \in P$. □

Example 4: Let pm = PBobCD, r = PermittedPhysicianPBob (with a valid role credential), and let p be defined as the predicate expression PE_{EX3} in Example 3. Then, the permission-to-role assignment policy is represented as

$$prassign(pm, r)\ iff\ p.$$

□

The set of administrative restrictions used in X-GTRBAC Admin are given in Table 4.5. These restrictions are not a fundamental component of the model, and have the status of recommendations. The use of these restrictions is not only intuitive but also keeps the administration concept simple in practice.

Table 4.5: X-GTRBAC Admin Restrictions

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

Continued on next page

Restriction	Description	Formal Semantics
AUA User cardinality	The cardinality of administrative users associated with Admin Roles through the AUA relation is 1, i.e. an administrative user can be assigned to only one Admin Role at any given time.	$\forall ar, ar' \in AR, ar \neq ar', (u, ar) \in AUA \Rightarrow (u, ar') \notin AUA$.

4.2.3 Administration Process

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

- **Creation of core component sets:** The first step in the administration process is the one-time creation of the core component sets of the model by the security administrator as defined in Definition 4.2.1. A key concept here is the creation of set of role (permission) instances, which allows different instances of the same role (permission) in different domains to adhere to the same role (permission) definition. This allows enforcement of uniform administration policies across the enterprise. Another significant feature is the introduction of partial order on the administrative role and administrative domain sets which captures the hierarchical organizational structure in the enterprise. We recommend the use of membership constraints defined in Table 4.5 on creation of RR, AD and ARD sets. These constraints imply that the association of Admin Roles and Admin Domains should be unique, and any authority over multiple Admin Domains can be transferred to an Admin Role only through semantics of the administrative role and domain hierarchies.

- **Assignment of administrative users to Admin Roles, and that of Admin Permissions to Admin Roles:** This is done by the security administrator using the assign_admin_role and assign_admin_permission operations

defined in Table 4.4. We recommend the use of cardinality constraints defined in Table 4.5 on these assignments. These constraints are natural to impose given the distinction between administrative roles and regular roles; administrative tasks would typically not require multiple users to be assigned to them, and vice versa.

- **Assignment of regular users to regular roles, and that of regular permissions to regular roles:** This is done by the Admin Roles using the assign_role and assign_permission operations defined in Table 4.4.

An example of policy administration process involving the use of these operations is presented in Figure 4.1; an administration process involving other administrative operations will be similar.

The example begins by creation of the core components and the constraints used in policy administration. This step is performed by the security administrator. After this step, the Admin Roles and Admin Permissions in the system have been designated through the AUA and APA relations. In the next step, assign_role and assign_permission operations are requested, involving assignment of user u2 and permission p2a to role r1a by ARa subject to the constraints c1 and c2 respectively. The pre-condition of these operations involves evaluation of the authorization relations and associated constraints, and, the post-condition (after a successful evaluation) is the addition of new tuples in the UA and PA relations.

In this chapter, we presented the administration model for the X-FEDERATE framework. The next chapter will discuss a federated architecture that allows the use of X-FEDERATE for federated access management in Web-based distributed systems.

Let

AD = {a, b};

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

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

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

APA = {(ARa, *assign*), (ARa, *assignp*)}

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

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

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

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

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

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

CR = $\{c_1 = (u1, \in, (assigned\_users(r1a)) \wedge (high, \neq, has\_attribute\_value(r1_a, priority))),$

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

Do

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

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

Pre-condition:

O1: $r1_a \in administers(ARa) \wedge (ARa, assign) \in APA \wedge evaluate(c_1) \leftrightarrow can\_assign(ARa, r1_a, c_1)$

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

Post-condition:

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

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

Fig. 4.1. An example of policy administration process involving *assign_role* and *assign_permission* operations

5. FEDERATION ARCHITECTURE

In this section, we present a federation architecture that allows the use of our X-FEDERATE framework for federated access management in Web-based distributed systems. The cornerstones of this architecture are the underlying federated database schema, an interoperable profile of our language for Web-based federation, and a federation protocol.

5.1 Federated Database Architecture

The X-FEDERATE framework is designed to be implemented in any distributed architecture that supports uniform schema definition. In our current work, we have based our federation architecture on the well-established Federated Database System (FDBS) described in literature [37–42]. To summarize, such an architecture consists of a multi-level schema, consisting of a local schema, a component schema, an export schema, and the overall federated schema. This schema architecture allows the resulting database system to support distribution, heterogeneity and autonomy, which are the three cardinal requirements of a federated database system. A local schema is the conceptual schema of a component database system expressed in the native data model of the component DBMS, and hence different local schemas may be expressed in different data models. A component schema is derived by translating local schemas into a data model called the canonical or common data model (CDM) of the FDBS. Two reasons for defining component schemas in a CDM are (i) they describe the divergent local schemas using a single representation and (ii) semantics that are missing in a local schema can be added to its component schema. Thus they facilitate negotiation and integration tasks performed when developing a tightly coupled FDBS. Similarly, they facilitate negotiation and specification of views and

multi-database queries in a loosely coupled FDBS. The process of schema translation from a local schema to a component schema generates the mappings between component schema objects and local schema objects. An export schema represents a subset of a component schema that is available to the FDBS. The purpose of defining export schemas is to facilitate control and management of association autonomy. A federated schema is an integration of multiple export schemas. A federated schema also includes the information on data distribution that is generated when integrating export schemas.

Of particular relevance to us in the FDBS architecture are the concepts of CDM, and the federated schema. The policy definitions in X-FEDERATE framework act as the CDM throughout the federated system, and will be used by each participating site to encode their export schemas. The integration of these schemas will then constitute the federated schema. The policy framework that we have designed can be adapted to work with any implementations of FDBS that abides by the FDBS architecture. Notable among the implementations reported in the literature are the Mermaid [41], IRO-DB [38], Disco [42], and MIRO-Web [37].

5.2 SAML profile

To adapt the X-FEDERATE framework for Web-based federation, we present a profile of our X-GTRBAC policy language based on the SAML [13] federated identity protocol. To develop the SAML profile of X-GTRBAC, we outline the configuration shown in Figure 5.1.

The interface to the system has been designed so that it should support, and not duplicate, the functionalities available in the SAML standard. SAML provides a message exchange protocol between autonomous business entities, and can be used to encode security attributes and decisions called assertions. However, SAML is not a self-sufficient mechanism to support Web-based federation and SSO as it does not provide any authentication or authorization support; it does the important

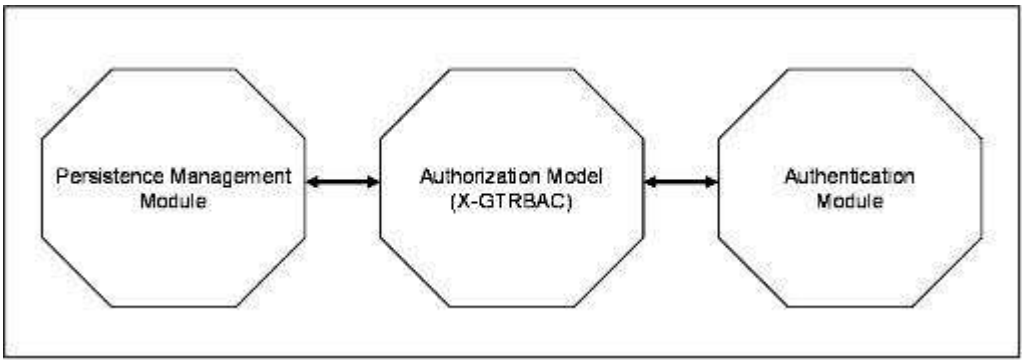

Fig. 5.1. Design configuration of the SAML profile of X-GTRBAC.

task of allowing the communicating entities to exchange security information in a decentralized manner but does not establish, check or revoke any information on its own. Therefore, a mechanism is needed that SAML can tie in to. Our specification provides one such mechanism, and is designed so as to accept SAML-encoded assertions as an acceptable form of credential. However, that alone is not sufficient for our purposes- SAML assertions are inherently subject to the same name-binding problem that exists in the protocols it is designed to work with, such as Kerberos and X.509. Therefore, we have designed the SAML profile of X-GTRBAC that works with attribute-based X-GTRBAC credentials. The SAML profile involves the feature set from latest SAML specification (v2.0). The use of SAML profile in X-GTRBAC system requires a translation from SAML encoding to X-GTRBAC format, and vice versa, using XSLT.

We now focus on precise policy configuration semantics of our proposed specification. Table 5.1 provides the credential configuration using SAML profile for X-GTRBAC in the context of this example. It uses features from SAML standard v2.0 which allows this credential configuration to be adopted by all entities that are already using SAML-compliant protocols.

Table 5.1: Credential Configuration in SAML profile

SAML Credential	**X-GTRBAC Instance**	**Mapping Rules**
<Assertion id= ″XXX-MAA-001″> <Issuer format= ″entity″> ... </Issuer > <AuthnStatement / > <AttributeStatement > <Subject > <NameID format= ″persistent″> Bobs public key </NameID > </Subject > <Conditions > <NotBefore > 2005:01:30 </NotBefore > <NotOnOrAfter > 2006:12:31 </NotOnOrAfter > </Conditions > <Attribute name = ″DOB″> <AttributeValue > 1978:05:21 </AttributeValue > </Attribute > <Attribute name = ″DLN″> <AttributeValue > 0991-09-0991 </AttributeValue > </Attribute > </AttributeStatement > <ds:Signature / > </Assertion >	<XUS xus_id= ″LibElseXUS″> <User user_id = ″any″> <UserName/ > <CredType cred_type_id= ″ LibElseResL2SAML″ cred_type_name= ″ LibElseResL2SAML″> <Header > <Issuer >... </Issuer > <Principal format= ″persistent″> Bobs public key </Principal > <Validity > <NotBefore > 2005:01:30 </NotBefore > <NotOnOrAfter > 2006:12:31 </NotOnOrAfter > </Validity > <DSig > </DSig > </Header > <CredExpr > <Attribute name= ″DOB″ value= ″1978:05:21″/> <Attribute name= ″DLN″ value= ″0991-09-0991″/> </CredExpr > </CredType> </User > </XUS >	• User@user_id = auto generated • NameID → UserName • CredType@cred-_type_id = auto generated • CredType@cred-_type_name = auto generated • Issuer → Issuer • NameID → Principal • NameID@format → Principal@format • NotBefore → NotBefore • NotOnOrAfter → NotOnOrAfter • ds:Signature → DSig • Attribute@name → Attribute@name • AttributeValue → Attribute@value

The credential in Table 5.1 is represented by a SAML assertion. We have only included the attributes and elements relevant for this discussion, and also omitted the namespace prefixes for compactness. The mapping rules used to translate a SAML assertion to X-GTRBAC format have been provided in the table. The X-GTRBAC credential is represented as an XUS document in our system. We now discuss the noteworthy features of the credential configuration:

- **Attribute Based Credential:** Of particular interest is the configuration of TM credentials in attribute-based mode which allows authentication for unknown users since identity is not assumed to be known. It can be observed that if a user name is not provided in the SAML credential, the corresponding credential in X-GTRBAC is constructed using the reserved word ˝any˝which represents anonymous users. In this case, the credential used is non-name-bound, and defines the identity of the subject in terms of a public key (or hash of it). This kind of binding is indicated by the value of ˝persistent˝for the format attribute of NameID element in SAML assertion. Persistent is a format for NameIDs in SAML standard that allows opaque values (such as random hashes) to be used in place of subject names in support of anonymity and privacy. Note that name-binding credentials can still be used if desired, which will be indicated by the appropriate value of the format attribute of NameID element as per the SAML standard (for e.g. X.509 Subject Name or Kerberos Principal Name).

- **Authenticating Attributes:** The AuthnStatement element in the SAML assertion contains the authentication context used to generate the authenticator (i.e. credential) for the subject. The attribute information contained in the credential is not necessarily owned by a centralized entity, and can be collected from multiple attribute authorities. The authentication statement for a subject can in practice be obtained by invoking the SAML Authentication Request protocol on an identity provider. The latter responds with the

authentication statement, and optionally also including attribute statements. This protocol includes the specification of a metadata repository from where required resource attributes may be learnt, and subsequently obtained using the attribute authorities indicated in the resource metadata. We maintain that our focus is not on attribute collection and credential generation. Instead, the SAML profile is designed to work with SAML assertions that already include such credentials generated through prior means.

In addition to attribute-based credential configuration as specified by SAML profile for X-GTRBAC, there are additional requirements on the use of credentials within the X-GTRBAC system to allow the access control capabilities of X-GTRBAC system to be integrated with Web-based federation and SSO features of SAML.

- **Role Assignment:** The attribute-based credentials from Table 5.1 are used by X-GTRBAC for attribute-based role assignment for unknown users. An appropriate X-GTRBAC policy configuration (see Table 5.2) allows a user Bob at a library (LibBob) to access PUThesis_Vol2006_No5 at a federated site (LibElse) using only his certified attributes. The assignment policy is represented as an XURAS document.

- **Delegation:** The requirement for delegation of authority is the key to decentralization, and is captured elegantly through the use of role hierarchy in our RBAC mechanism: a junior role inherits all privileges of a senior role. At present, we only support delegation within the role hierarchy (i.e. delegation always occurs from a senior role to a junior role). An optional Delegation Constraint may be used in the role definition (See Table 5.2) to limit the extent of delegation (in terms of time and associated privileges); unrestricted delegation is otherwise assumed. The role definition is given in an XRS document (see Table 3.1).

Table 5.2: Constraint Specification in X-GTRBAC

Constraint	**X-GTRBAC Instance**	**Mapping Rules**
Role Assignment	<XURAS xuras_id= "LibElseXURAS"> <URA ura_id= "uraBorrowerL2" role_name= "BorrowerL2"> <AssignUser user_id= "any"> <AssignConstraint > <AssignCondition cred_type_id= "LibElseResL2SAML" d_expr_id= "TwoDays"> <LogicalExpr > <Predicate > <Operator >neq </Operator > <FuncName >hasValue </FuncName > <ParamName >DLN </ParamName > <RetValue >null </RetValue > </Predicate > <Predicate > <Operator >neq </Operator > <FuncName >hasValue </FuncName > <ParamName >DOB </ParamName > <RetValue >null </RetValue > </Predicate > </LogicalExpr > </AssignCondition > </AssignConstraint > </AssignUser > </URA > <XURAS >	The role BorrowerL2 can only be assigned to a user who possesses the credential LibElseResL2SAML. This refers to the credential defined in XUS document in Table 5.1. The assignment condition includes rules on credential attributes. It asserts the existence of the DLN and DOB attributes. The assignment condition also refers to a duration expression which implements the restriction that the resource can be borrowed only for 2 days. The duration expression is defined in an XTempConstDef document (see Table 3.2).

Continued on next page

Constraint	X-GTRBAC Instance	Mapping Rules
Role Delegation	<XRS xrs_id= ″xrsBorrowL2″> <Role role_id= ″rBorrowerL2″ role_name= ″BorrowerL2″> <Junior >BorrowerL1 </Junior > <DelegationConstraint > <DelegationCondition d_expr_id= ″OneWeek″/ > </DelegationConstraint > </Role > </XRS >	The role BorrowerL2 can only be delegated if the delegation constraint is satisfied. The delegation condition on the role refers to a duration expression which imposes a restriction on the duration of the delegation. The duration expression is defined in an XTempConstDef document (see Table 3.2).

- **Digital Signatures:** An effective SSO solution depends on the persistence of the authentication and authorization assertions across enterprise domains. Toward this end, the Header element of an X-GTRBAC credential includes support for digital signatures. The support for digital signatures in SAML allows signed assertions to be exchanged between all SAML-compliant entities.

The proposed mechanism for using the SAML profile of X-GTRBAC with the X-FEDERATE framework is depicted in Figure 5.2. The authentication module is responsible for generating the attribute and authentication statements included in the SAML assertion. The use of standardized protocols allows us to leverage existing mechanisms for these tasks. The SAML Authentication Request protocol discussed earlier is now implemented by stand-alone SAML-aware Web server software (e.g.: http://www.pingidentity.com/products/pingfederate.html), and may be deployed by SAML authorities to create and exchange SAML-compliant attribute and authentication statements. The persistence management module is responsible for creation of digitally signed authorization credentials. We outsource the creden-

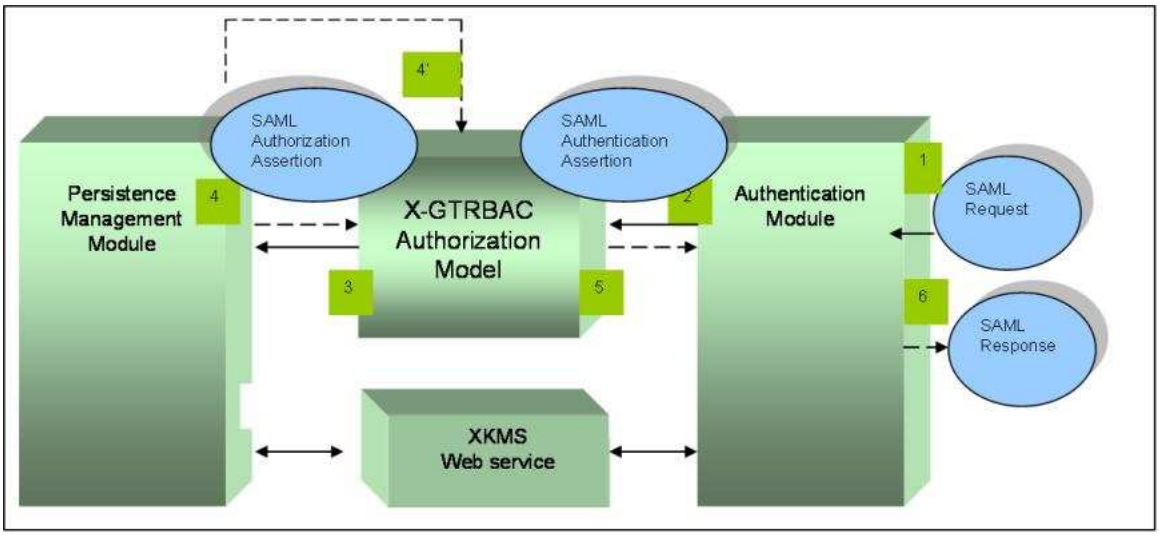

Fig. 5.2. Software architecture for the SAML profile of X-GTRBAC.

tial management to the well-known XML Key Management Specification (XKMS - http://www.w3.org/TR/xkms/). XKMS is a Web-based service that can be invoked from a client application, and supports PKI-based key generation, registration, revocation, and verification. SOAP binding is used for message exchange. XML Encryption and XML Digital Signature standards are used to provide message confidentiality and authenticity, respectively. The end-to-end communication is assumed to be secured using mechanisms such as SSL/TLS.

5.3 Federation Protocol

Using the FDBS architecture and the SAML profile, we now describe a federation protocol for information sharing in a federated system. The protocol guides the use of the policy documents defined at each federating site, and assumes that the policy definition sheets that govern the format of credentials, attributes and predicates used in the policy are accessible to the federating sites at a well-known location. This would typically be the federation administrator site which bootstraps the federation.

The following steps are included in the federation protocol (we continue the use of the digital library example for illustration):

1. The protocol is initiated with a user Bob submitting an authorization query from his home site (LibBob) for accessing a Web resource available at a federating site (LibElse). The request includes the requesting subject (Bob), the requested resource (PUThesis_Vol2006_No5) and the requested permissions (http:GET) on the resource. It may optionally include an authentication or authorization credential to assist in the authorization decision. The credentials are collected and supplied by the home site (LibBob) of the requesting user. We use a SAML-compliant format for all access requests.

2. Upon receiving an access request, the X-GTRBAC access control module at the federating site (LibElse) extracts the policy information from the policy base to enforce the authorization constraints on the release of the request resource. The access request may either be from a local or a federated user (Bob is a federated user from LibBob). If the requested resource is not available within the system, the access control module simply returns (or appropriately redirects) the request. Otherwise, it proceeds as follows.

3. As an initial step, the access control module processes the request (expressed as a SAML Authorization Decision Query). This includes translating the credential included in the request from the SAML format to X-GTRBAC format. Based on the kind of credential, it does the following.

    - If it is an authentication credential (expressed as a SAML Attribute Statement included in the query), the access control module assigns the user (Bob) to an appropriate role (Borrower_L2) within the system according to the user-to-role assignment policy of the federating site (LibElse). This process requires (successful) evaluation of temporal and non-temporal contextual constraints before the role assignment takes place. The non-temporal constraints not only involve rules on user attributes but may

also involve rules on attributes of the role for a fine-grained access control. Following a successful role assignment, the user (Bob) is issued an authorization credential by the system.

- If it is an authorization credential (expressed as a previously issued SAML Authorization Decision Statement included in the query), the access control module inspects if the role information in the credential corresponds to a local role. If it doesnt, it means that it is an SSO request and includes the role of the federated user (Bob) in his original domain. In this case, the access control module invokes a role mapping routine to map the user to a local role according to the delegation policy of the system. After this step, the user acquires the privileges of the assigned or mapped role in the local system.

4. The next step after establishing the role (Borrower_L2) of the user (Bob) is to determine the authorization of the user to access the requested resource (PUThesis_Vol2006_No5). At this stage, the access control module evaluates the authorization credential of the user (Bob) according to the permission-to-role assignment policy of the federating site (LibElse). This process requires (successful) evaluation of temporal and non-temporal contextual constraints before the permission assignment takes place. The non-temporal constraints may not only involve rules on role attributes but also on attributes of the resource for a fine-grained access control.

5. The result of the evaluation is generated by the access control module (expressed as a SAML Authorization Decision Statement), including any applicable resource provisioning constraints.

6. As a final step, the access control module forwards the authorization credential (i.e. SAML Authorization Decision Statement) for the user (Bob) to a document generating routine, which consults the access rights of the requesting user on the requested XML document (PUThesis_Vol2006_No5), and accord-

ingly generates XML views in response to the request. A session management routine within the access control module is repsonsible for monitoring the provisioning and de-provisioning constraints associated with the requested document.

7. The SAML Authorization Decision Statement issued by the federating site (LibElse) is digitally signed so that it can be subsequently reused at another site that accepts this authorization credential. This will allow the user (Bob) to be authorized at the latter site without having to go through an authentication process. As mentioned earlier, such a mechanism effectively provides support for Single Sign On (SSO).

A reference implementation of the federation architecture described in this chapter, along with the related policy documents, can be accessed at http://web.ics.purdue.-edu/ bhattir/project/sso. It involves the use of SAML decision queries and responses to evaluate the requests for accessing Web resources, as described in the protocol.

This chapter described the federated architecture for the use of our X-FEDERATE framework for federated access management in Web-based distributed systems. In the next chapter, we will provide a distributed system application of the framework in the healthcare domain.

6. DISTRIBUTED SYSTEM APPLICATION: HEALTHCARE INFORMATION MANAGEMENT

In this chapter, we demonstrate the use of X-FEDERATE for federated healthcare information management. This illustrates the application of our policy engineering methodology for the design and administration of access management policies for secure sharing of XML-based clinical documents in a federated healthcare system.

We consider a HealthCareFederation (HCF) clinical system. It is assumed in the example that all role names, clinical document types, and patients are part of the HCF database. The design of the HCF database will be based on the federated database architecture (FDBS) discussed in Chapter 5. The X-GTRBAC policy language used in our framework comprises the definitions of the various policy components used to encode the access management policies of all federating organizations within the HCF. The policy definitions, therefore, serve as the Common Data Model (CDM) for the HCF database and the policy documents are used to define an export schema, which collectively form the federated schema of the system.

We will now focus on the policy design and the application of our policy engineering methodology for federated healthcare information management.

6.1 Policy Design

To adequately capture the access management requirements of clinical resources in a healthcare system, we use the well-known Clinical Document Architecture (CDA) standard [43] issued by the HL7 organization to represent the protected clinical resources. Consequently, we base the design of our policy on a set of use cases proposed by the community for the CDA standard [44, 45]. We specifically focus

on the use cases involving the use of ˜layered constraints˝within the CDA-encoded clinical documents, which we will refer to as Electronic Health Records (EHRs). The HL7 Template proposal [44] which is intended to provide a format for defining constraints against an HL7 specification is a recent effort in this direction. The proposal allows for a template to be used to constrain the values of static assertions regarding an EHR. This includes constraints on allowable attribute values, comparison of attribute values, nesting of assertions, and logical evaluation of assertions. As the HL7 proposal suggests, the use of such a template will allow formulation of expressive constraints on the document contents, which we believe can be used in the design of disclosure and privacy policies for the EHRs.

Below we present the set of use cases based on which we design the policies used in our framework.

Actors: The creators and readers of EHRs (such as physicians and healthcare givers), patients associated with them who have access privileges, payers (insurance), and institutions (HMOs, government bodies, enforcers of legislations such as HIPAA) who have been permitted to access EHRs.

- **Use Case 1:** Access policy for an EHR must have granularity at the level of (i) the medical and administrative (such as address, phone no.) data, (ii) the category of medical record as defined per the ClinicalDocument type system in CDA, and (iii) the type of requestor within the actor populations.

- **Use Case 2:** The portion of EHR retrieved by a permitted requestor depends upon the requestor and the privacy conditions defined on the EHR.

- **Use Case 3:** The restrictions on accessing an EHR extend beyond the originating organization.

- **Use Case 4:** A user needs to get through the protections in case of emergencies.

- **Use Case 5:** A user may be denied access to certain sensitive and damaging diagnosis information.

We now outline the design of our policy with respect to the use cases described above. We consider a HealthCareFederation (HCF) clinical system. It is assumed in the example that all role names, document types, and patients are part of the HCF database. We do not concern ourselves with setting up the federated database, and only describe the design and enforcement of policies. The rules in the example policy, along with the use cases to which they relate, are given below.

- **Granularity of Access** [UseCase1]

  R1 A US board-certified physician can access medical data in any ClinicalDocument.

  R2 Locally certified physicians can only access clinical documents of type Discharge Summary.

  R3 A billing clerk can access administrative data in any ClinicalDocument of any patient.

- **Disclosure Rules** [UseCase2]

  R4 The disclosure policy for a billing clerk accessing an EHR of category "ClinicalDocument"requires the access to be restricted during the first week of any quarter in year 2005 and for a duration of 1 week.

  R5 For a ClinicalDocument of type Discharge Summary, the resource access is restricted to only occur from within the state of NewYork.

- **Privacy Rules** [UseCase2]

  R6 The privacy policy of patient Bob allows a physician to access its records only if they have attributes board_certified_id with value NY and fellowship_field_cd with value GeneralMedicine in their CDA-encoded credential.

- **Emergency Defaults** [UseCase4]

  R7 The applicable privacy or disclosure policy for an EHR may be overridden if the access has to occur from near or inside the EmergencyRoom.

- **Information Hiding** [UseCase5]

  R8 The patient may not be able to view a ClinicalDocument of type Psychiatry Report.

6.2 Example Policy

We shall now discuss an example policy involving the use of the constraint expressions in our framework to capture the requirements use cases outlined in Section 6.1. The rules in the example policy, along with the use cases to which they relate, are given below:

We will use the following categories of resources as defined by CDA:

1. **Clinical Document (CD):** All EHRs belong to this category
2. **Discharge Summary (DS):** An EHR belonging to this specialized category of CD.
3. **Psychiatry Report (PR):** An EHR belonging to this specialized category of CD.
4. **CDAD:** The administrative portion of the CD. (All types except CDAD mean medical portion.)

While we have already indicated the assignment rules that allow physicians to access the EHRs of patients, it is also necessary to encode rules that allow patients to view their EHRs. For this purpose, we use the convention that a user x is assigned to a role Px in order to access its own EHRs. A role Px by default has the permissions to access all its records, but manual overrides may be encoded by system administrator in special circumstances.

6.2.1 Formal Specification

We now encode the policy rules using the administrative concepts discussed in Chapter 4, and use where applicable the predicate expression formalism.

- **Rule 1:** Let $pm_{Rule1} = PxCD$, $r_{Rule1} = PermittedPhysicianPx$ (having a valid role credential), and let $pRule1$ be defined on $LE_{Rule1} = (US, =, hasCredAttrValue(PermittedPhysicianPx, board\_certified\_id))$. Then, the policy rule is represented as

$$prassign_{Rule1}(pm_{Rule1}, r_{Rule1}) \, iff \, p_{Rule1}.$$

- **Rule 2:** Let $pm_{Rule2} = PxDS$, $r_{Rule2} = PermittedPhysicianPx$ (having a valid role credential), and let p_{Rule2} be defined on $LE_{Rule2} = (NY, =, hasCredAttrValue(PermittedPhysicianPx, board\_certified\_id))$. Then, the policy rule is represented as

$$prassign_{Rule2}(pm_{Rule2}, r_{Rule2}) \, iff \, (p_{Rule2} \vee p_{Rule1}).$$

- **Rule 3:** Let $pm_{Rule3} = PxCDAD$, $r_{Rule3} = BillingClerk$ (having a valid role credential). Then, the policy rule is represented as

$$prassign_{Rule3}(pm_{Rule3}, r_{Rule3}).$$

  Note that this policy rule has no associated constraint.

- **Rule 4:** Let $pm_{Rule4} = PxCDAD$, $r_{Rule4} = BillingClerk$ (having a valid role credential), and let p_{Rule4} be defined on $PTERule4 = PTE_{EX1} =< P, [2005.Years, 2005.Years] >, P = all.Years{+}1, 4, 7, 10.Months{+}1.Weeks \triangleright 1.Weeks$. Then, the policy rule is represented as

$$prassign_{Rule4}(pm_{Rule4}, r_{Rule4}) \, iff \, p_{Rule4}.$$

- **Rule 5:** Let $pm_{Rule5} = PxDS$, $r_{Rule5} = PermittedPhysicianPx$ (having a valid role credential), and let p_{Rule5} be defined on $LE_{Rule5} = LE_{EX2} =$

$(NewYork, =, hasCredAttrValue(PermittedPhysicianPx, location))$. Then, the policy rule is represented as

$$pr_assign_{Rule5}(pm_{Rule5}, r_{Rule5}) \, iff \, p_{Rule5}.$$

- **Rule 6:** Let $u_{Rule6} = any$, $r_{Rule6} = PermittedPhysicianPx$ (having a valid user credential). Let p_{Rule6a} be defined on $LE_{Rule6a} = (NY, =, hasCredAttr-Value(Px, board\_certified\_id))$ and let p_{Rule6b} be defined on $LE_{Rule6b} = (G-eneralMedicine, =, hasCredAttrValue(Px, fellowship\_field\_cd))$. Then, the policy rule is represented as

$$urassign_{Rule6}(u_{Rule6}, r_{Rule6}) \, iff \, (p_{Rule6a} \wedge p_{Rule6b}).$$

- **Rule 7:** Let $pm_{Rule7} = PxCD$, $r_{Rule7} = PermittedPhysicianPx$ (having a valid role credential), and let p_{Rule7} be defined on $LE_{Rule7} = (EmergencyRoom, =, hasCredAttrValue(PermittedPhysicianPx, location))$. Then, the policy rule is represented as

$$prassign_{Rule7}(pm_{Rule7}, r_{Rule7}) \, iff \, p_{Rule7}.$$

- **Rule 8:** Let $pm_{Rule8} = PxPR$, $r_{Rule8} = Px$ (having a possibly null role credential). Then, the policy rule is represented as

$$pr_deassign_{Rule8}(pm_{Rule8}, r_{Rule8}).$$

We note that in practice, a combination of these rules may be needed in an assignment policy. We recall that our rule specification supports combing rules from multiple sources, which allows combining multiple predicate expressions in a policy rule (as shown in Example 4 in Chapter 4). For instance, Rule3 and Rule4 may be combined in a permission-to-role assignment policy by using the AND rule-combing mode to ensure that both rules are satisfied before the policy returns true, whereas Rule 7 used for emergency defaults may be combined with any existing rule using

the OR rule-combing mode so that the policy always returns true when an emergency context has been detected (See Section 3.3.3 for discussion on rule-combining modes). Also, with particular reference to Rule 8, it is an example of manual overriding of default privileges of a patient to access its own records. We note that the semantics of deassign are opposite to that of assign, and it removes the assignment of a permission from a role. To resolve conflicts, we associate a higher priority with deassign operation.

6.2.2 UML model creation

UML model for the specification is created by the system designer by following the policy engineering methodology presented in the thesis. As indicated in Chapter 3, the methodology suggests the use of UML Package (see Table 3.4) which comprises the definitions of the federated schema for the various policy components to encode the access management rules outlined in the example policy.

The UML model is then used by the system designer to integrate the policy with the system. It may be mentioned that at no point does the system designer actually have to deal with low-level X-Grammar syntax of the underlying policy language because the semantics of the language have already been incorporated in the high-level UML model. The mapping of UML model to XML Schema subsequently allows the policy specification to be translated into implementation code without having to deal with the complexity of the policy design. In our framework we use Java as the implementation language, and translate the XML Schema elements into Java classes using custom parsing and processing routines. This is the task of the policy enforcement architecture.

6.3 Enforcement Architecture

In this section, we present the policy enforcement architecture of our HCF clinical database system. The architecture comprises of two components: the integrated

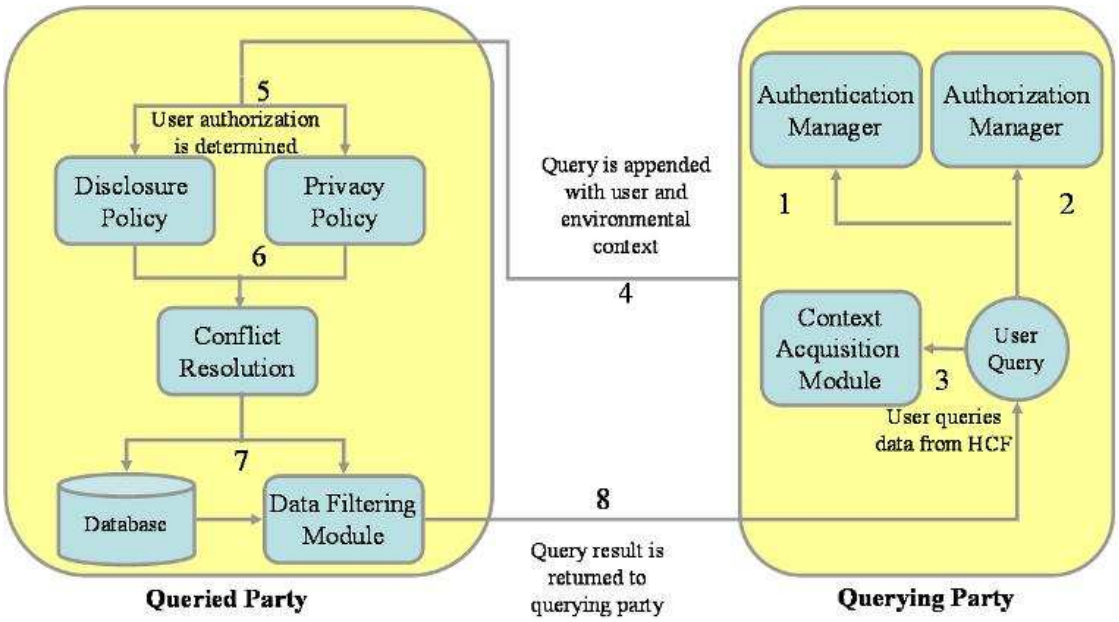

Fig. 6.1. The integrated architecture for federated healthcare database prototype.

architecture (IA) is shown in Figure 6.1, whereas the subsystem architecture (SA) for each individual domain is shown in Figure 6.2. The enforcement architecture has been implemented within a research prototype that implements our example policy.

6.3.1 Subsystem Architecture

The SA is employed at all parties in the HCF, and each party uses the definitions of the various policy components described in Section 3.3 to encode the disclosure and privacy policies. As indicated earlier, the policy documents constructed using these policy definitions form the federated schema of the system. We now provide an overview of the key components of the SA.

The XML Document Composition Module (XDCM) is used by each participating site to compose policy documents. Each site first creates the UML classes of its policy definitions using the UML package described in Section 3.5. The UML classes are then translated into XML schemas using a UML to XML Schema translation

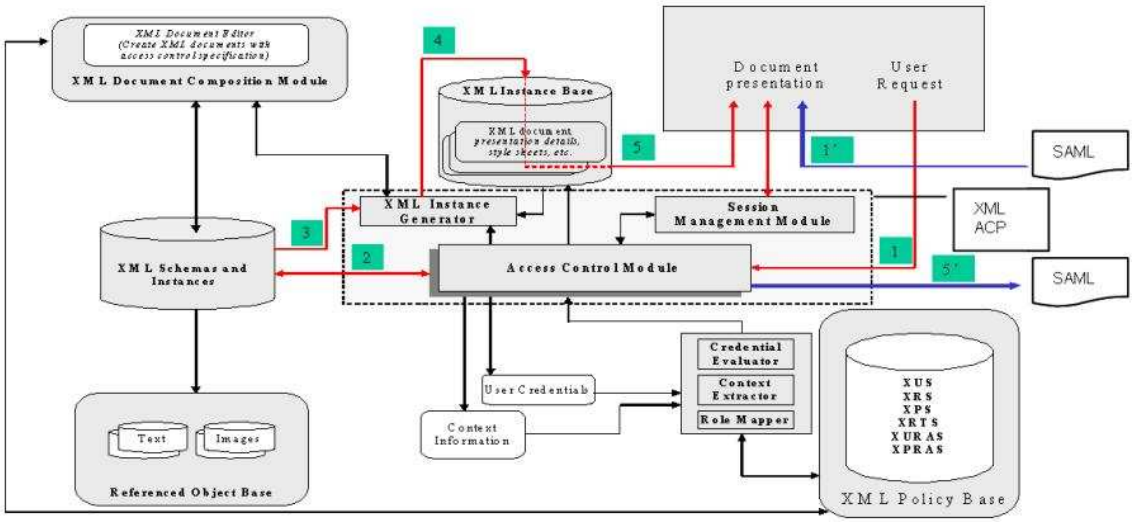

Fig. 6.2. The subsystem architecture for federated healthcare database prototype.

mechanism and exported to XDCM. The policy documents are then composed in XML inside the XDCM, and verified against the imported schema definitions.

The XML Policy Base (XPB) contains all policy related XML documents composed through XDCM. These documents collectively form the federated schema of the system. These include XML User Sheet (XUS), XML Role Sheet (XRS), XML Permission Sheet (XPS), XML Resource Type Sheet (XRTS), XML User to Role Assignment Sheet (XURAS), and XML Permission to Role Assignment Sheet (XPRAS). Also stored in XPB are the policy definitions, including XCredTypeDef, XResTypeDef, XSoDDef, XTempConstDef and XPredFuncDef.

Each individual resource (EHR) in the system belongs to a category (resource type) as defined per the CDA standard which is captured through the use of a type tag. Examples for the possible types are: Clinical Document, Discharge Summary, Psychiatry Report, Diet, Disability, and Observation. Note that binary encoding allows objects to be embedded within XML-based EHRs. Such objects may include multimedia data, such as medical images, which can be accessed by users who have

authorization to view the containing document. These objects are stored in the Referenced Object Base.

6.3.2 Integrated Architecture

The IA ties together the individual SAs together with externally available modules to provide support for federated access management. Examples of externally available modules may be authentication services and context acquisition services which may not be a part of the federated database. We now provide an overview of the key components of the IA.

- **Authentication and Authorization:** A user (any actor in our use cases) wishing to request clinical data from the HCF needs to provide credentials defined per the federated schema. To provide a scalable identity and authorization management infrastructure, the architecture employs an Authentication Manager and an Authorization Manager. The Authentication Manager is not directly a component of our authorization infrastructure, but is used to issue an authenticating credential to the user (encoded as an XUS in our framework). Subsequently, this authenticating credential is presented to the Authorization Manager. The Authorization Manager is then responsible for role assignment of the user request based on the attributes encoded in the user credential. Following a successful role assignment, the Authorization Manager issues an authorization credential to the user (encoded as an XRS in our framework). Since the Authorization Manager issues the credential defined per the federated schema, hence the authorization credential issued by it is accepted at all federating sites within the HCF. The credential evaluation for both the authenticating and authorization credentials is performed by the Credential Evaluator component of the SA of the federating site (see Figure 6.2).

- **Context Acquisition:** We have motivated that access to sensitive clinical records can be based on contextual conditions. This includes the user context

(which is included in the credential attributes), and the environmental context (such as location information). Both the user context and environmental context needs to be appended to the access request before it is submitted to the queried party. Based on our requirements for location-sensitive access control in personal healthcare information management, we provide a location capture mechanism which is part of the more general Context Acquisition Module (CAM). We envision the following function of the location capture service: It will capture location information through the IP address of the nearest registered access point (AP) through which the device of the requesting user communicates (we do not rely on the using the IP address of the device itself because the devices in a pervasive computing environment will typically be mobile devices having temporary IP addresses not sufficient to track location). This mechanism will be used to satisfy the proximity requirement in our system, and also to detect the loss of proximity to appropriately disengage the resource provisioning session. The design of a CAM providing such a service is outside the scope of our work, and is the subject of ongoing research [46]. We, however, for the purposes of our prototype system assume the existence of a software routine that simulates its effect. The location data is received by the system through a GUI which allows the location of a device to be entered for the purposes of simulated testing. The context acquisition module works in tandem with the Context Extractor component of the SA of the federating site (see Figure 6.2).

- **Disclosure and Privacy Policies:** The query embedded with the context information is evaluated by the access control module of the queried party. The query is checked against the policy documents for the federating site defined per the federated schema. The query evaluation is performed by the XML ACP component of the SA of the federating site (see Figure 6.2). It consists of two phases: (i) first, the privacy policy of the patient is checked for the requisite authorizations of the requesting user based on his/her supplied credentials, and

(ii) second, the disclosure policy of the queried party is checked for any restrictions on the release of the requested content based on the user authorization. As we indicated earlier, it is possible that the possibly rules may be conflicting (such as Rule 8 in the example policy of Section 6.2 conflicts with the default access of patients to their own records), and a conflict resolution mechanism is provided. The current strategy employed in our prototype is simply denial-takes-precedence, which has been implemented by assigning higher priority to de-assignment rules.

- **EHR Retrieval:** Based on the privacy and disclosure policy, the information violating user preferences or organizational rules can be omitted from the returned view of the data. The requested EHR content is retrieved from the HCF database (which essentially comprises of the individual data repositories of the federating sites as shown in the SA). The retrieved content is then sent to the Data Filtering Module, where appropriate view of the content is generated using the XML Instance Generator component of the SA (see Figure 6.2). Finally, the resulting view of the requested EHR content together with applicable resource provisioning constraints is returned to the querying party. The Session Management Module component of the SA (see Figure 6.2) is repsonsible for monitoring the provisioning and de-provisioning constraints associated with the requested EHR.

6.4 Implementation

In this section, we discuss the implementation of the example policy. The policy has been implemented in our prototype system that has been designed based on the architecture described in the previous section

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<Patient id= "xxx">

    <PersonalInformation>
        ....
        ....
    </PersonalInformation>

    <MedicalServiceProvider>
        ....
        ....
    </MedicalServiceProvider>

    <FamilyMedicalHistory>
        ....
        ....
    </FamilyMedicalHistory>

    <PatientMedicalRecords>
        ....
        ....
    </PatientMedicalRecords>

</Patient>
```

Fig. 6.3. XML-based EHR layout

6.4.1 Policy and Records Database

The policy documents for the HCF clinical database are stored in an XML Policy Base (XPB) at each federating site. As indicated earlier, the XML schema for the example policy is generated using the UPX and instance documents are composed and validated against it. These documents provide information on users, roles, and resources, and assignment rules needed for providing access management in our federated healthcare database system. The set of XML policy documents for the example policy implemented in our prototype are provided in Appendix C.

The actual resource type instances (i.e. the XML-based EHRs) at a participating site to which the users of the HCF will be requesting access are stored in an XML database shown by the XML Schemas and Instances component in the SA. Our current implementation stores them as XML-type tables in Oracle DBMS.

In Figure 6.3, we give the layout of the EHRs. Based on the requirement for fine-grained access based on the portion (administrative vs. medical) and type (Discharge Summary, Psychiatry Report, etc.) of the Clinical Document, we have structured the

EHR in a manner that allows retrieval of only the authorized content. The Personal-Information tag contains patients personal information, such as name, address, date of birth, identification numbers, and phones. This tag is of type ˝administrative˝and hence an element having this tag name can be accessed only by a user who has been authorized to access a resource of type CDAD (recall that this type has been defined in Section 6.2 to refer to the administrative portion of the Clinical Document). All the other tags in the document are of type ˝medical˝and hence an element having any of those tag names can be accessed only by a user who has been authorized to access a resource of type CD or any sub-category of it (recall that all types other than CDAD have been defined in Section 6.2 to refer to the medical portions of a Clinical Document). These tags include MedicalServiceProvider tag, the Family-MedicalHistory tag, and the PatientMedicalRecords tag.

6.4.2 Policy Enforcement

We now illustrate the enforcement of the example policy implemented in our current prototype. The idea is as follows. Access requests to retrieve patients EHR are composed using credentials obtained from the Authentication and Authorization Manager. The Context Acquisition Module appends the contextual information to the request, and it is then submitted to the queried party. At the latter end, they are first checked against the disclosure and privacy rules maintained in the XPB of the queried party. Based on these rules, queries are reformulated such that users can only access what they have authorization to. The reformulated query stores information about the patient and the authorized EHR types in a well-known format. The reformulated query is then passed to a procedure that parses the query and retrieves the required content from the EHR database. The answer to the query is stored in an XML document that contains only the required text and multimedia data. This result document is composed *on the fly* and is then returned to the querying party.

We now illustrate some scenarios of evaluation of access requests using our example policy, where the requests are representative of the requirements use-cases outlined earlier. In particular, we cover rules 1 thru 6 mentioned in Section 6.1. The scenarios involving remaining rules will be similarly handled by making simple extensions to the example policy to handle exceptions and denial, as already described.

- **Scenario 1:** Physician Smith wishes to access Clinical Document of patient Bob. Smith has a US board certification.

  Smith presents to the system a CDAIndividualHealthCarePractitioner credential encoded as an XUS (See Table C.1(b)). The credential contains the attributes board_certified_id having a value US and fellowship_field_cd having a value GeneralMedicine for Smith. The privacy policy of patient Bob is encoded as an XURAS (See Table C.2(a)) which requires that any physician having a board certification of NY is permitted to view EHRs of Bob. Since Smith has a US board certification, and assuming the US certification subsumes local certification, he is assigned the PermittedPhysicianPBob role (which is defined in XRS in Table C.1(e)). Following this role assignment, Smith is then evaluated for permission assignments against the constraints defined on credential attributes to enforce the disclosure policy. The assignment of CP_PBob_CPrCD_GET permission (which is defined in XPS in Table C.1(f)) defined to access any Clinical Document of patient Bob requires the user to have a board_certified_id having a value US (see Table C.2(b)), and hence Smith is eligible to be assigned the requisite permission.

- **Scenario 2:** Physician Carla wishes to access Clinical Document of patient Bob. Carla has a NY-board certification. Carla is accessing from New York.

  Carla presents to the system a CDAIndividualHealthCarePractitioner credential encoded as an XUS (See Table C.1(b)). The credential contains the attributes board_certified_id having a value NY and fellowship_field_cd having a

value GeneralMedicine for Carla. The privacy policy of patient Bob is encoded as an XURAS (See Table C.2(a)) which requires that any physician having a board certification of NY is permitted to view EHRs of Bob. Since Carla has a NY board certification, he is assigned the PermittedPhysicianPBob role (which is defined in XRS in Table C.1(e)). Following this role assignment, Carla is then evaluated for permission assignments against the constraints defined on credential attributes to enforce the disclosure policy. The assignment of C-P_PBob_CPrCD_GET permission (which is defined in XPS in Table C.1(f)) defined to access any Clinical Document of patient Bob requires the user to have a board_certified_id having a value US (see Table C.2(b)), and hence Carla is not eligible to be assigned the requisite permission.

- **Scenario 3:** Physician Carla wishes to access Discharge Summary of patient Bob. Carla has a NY-board certification. Carla is accessing from New York.
  Similar as above, except that now the permission requested is CP_PBob_C-PrDS_GET. The assignment of CP_PBob_CPrDS_GET permission (which is defined in XPS in Table C.1(f)) defined to access Discharge Summary of patient Bob requires the user to have a board_certified_id having a value of either US or NY, and the role to have a location having a value of NewYork (see Table C.2(b)). Since these conditions are satisfied in this instance, Carla is eligible to be assigned the requisite permission.

- **Scenario 4:** Billing clerk John wishes to access Clinical Document of patient Bob. John is accessing in second week of February in 2005.
  John presents to the system a ClinicPurdueBillingClerk credential encoded as an XUS (See Table C.1(b)). The assignment policy for BillingClerk role (which is defined in XRS in Table C.1(e)) is encoded as an XURAS (See Table C.2(a)) which does not require any user credential to be presented, but has an associated temporal constraint PTQuarterWeekOne (which is defined in XTempConstDef in Table C.1(c)). This constraint states that the assignment

is only allowed beginning the first week of every quarter of year 2005. Since John is accessing in second week of February, which is not the first week of any quarter in 2005, hence John is not eligible to be assigned the requisite role.

- **Scenario 5:** Billing clerk John wishes to access Clinical Document of patient Bob. John is accessing in first week of April in 2005.
  Similar as above, except that now the time of access is within the range of the temporal constraint. Since John is accessing in first week of April, which is the first week of the second quarter of year 2005, hence John is eligible to be assigned the requisite role.

The implementation of the above scenarios, along with the related policy documents, can be accessed at http://web.ics.purdue.edu/ bhattir/project/hcf. Our current prototype uses SAML to encode the access requests as SAML decision queries, and to encode the authorization decision as SAML decision statements.

This chapter provided a distributed system application of X-FEDERATE, and demonstrated the use of our policy engineering methodology for federated healthcare information management. The next chapter will conclude the thesis.

7. CONCLUSION AND FUTURE WORK

In this thesis, we have presented X-FEDERATE, a policy engineering framework for federated access management. It has been developed to manage the task of designing policies for federated information sharing, and to address the inherent incompatibility of multiple policy-based management approaches when applied to a federated system. We have analyzed the impact of security management challenges on policy design, and formulated a policy engineering methodology based on principles of software engineering to develop X-FEDERATE as a policy-based access management solution for federated systems. It consists of an XML-based policy specification language, its UML-based meta-model, and an enforcement architecture. The policy language is designed to meet the requirements for federated access management, while the use of UML-based meta-model allows policies to be developed and managed in a standardized manner. The framework also consists of an administration model targeted at decentralized policy administration in the presence of constraints. Our model provides a formal specification of administrative concepts and constraints to facilitate the administration of context-aware RBAC policies. We have provided a comprehensive comparison of our approach with related approaches, and highlighted its significance for federated access management. We have also presented a federation architecture and protocol. We have also discussed a distributed systems application of our framework in the area of healthcare information management.

The UML support for policy engineering provided in our framework can be enhanced to provide many more useful features from software engineering discipline. In addition to the use of a UML profile in the meta-model, other constructs such as use cases, activity diagrams, and sequence diagram in UML can also be used to model security relevant behavior of the system by allowing one to express appropri-

ate security constraints using well-known modeling notations. Recent research has also shown that UML classes and sequence diagrams can be analyzed using techniques based on typed graphs [47]. The specification of policy using UML notations can therefore provide support for consistency and verification analysis. These can be promising extensions to the UML-based meta-model developed in this work.

The use of UML model in our framework can also be leveraged to provide support for model-based security testing of RBAC policies. We have conducted some preliminary work in this regard [48] that deals with model-based strategy for testing implementation of an access control system that employs the RBAC policy specification. The approach has been based on the construction of a structural and behavioral model of the corresponding RBAC specification. The model is then used to generate static and dynamic test suites for the corresponding implementation. We plan to extend this work to incorporate security testing of access control systems that employ the complete X-GTRBAC policy specification language.

With regards to security management, there are several future directions that this work can be extended in, including both potential applications and design enhancements. We have already investigated the use of X-GTRBAC policy specification in Web-services acccess control, and have also proposed a WS-Policy profile of X-GTRBAC [49]. WS-Policy [50] is an emerging specification for expressing Web-services usage policies. The WS-Policy profile would allow the use of X-GTRBAC policies for access control in Web services using policies expressed in WSPolicy syntax. Other application areas include pervasive and mobile computing applications, where location-sensitive context-aware access control is absolutely a critical requirement. We have recently proposed a GEO-RBAC profile of X-GTRBAC policy [51] to address the requirement for spatial access control in such systems. GEO-RBAC [52] is a spatially-aware extension of RBAC designed to allow fine-garined and expressive location-based access control. We plan to continue work in this direction and integrate our system with practical mobile and wireless applications.

In terms of design enhancements, it is desirable to incorporate many additional aspects in our policy framework. Our current approach for role mapping abides by the local autonomy principle, and hence no form of external access mediation is needed. In a more general case, this may on one hand be overly-restrictive, and on the other hand lead to security breaches due to transitive establishment of undesirable delegation links. Therefore, a mediation mechanism is necessary to fairly regulate federated information sharing while ensuring security of federated resources. Composing an access mediation policy in a federated system poses considerable challenge since participating sites do not have a-priori knowledge of each others access control polices. We envision a decentralized mediation mechanism, where user attributes combined with declarative rules can form a criterion for automated trust establishment. This mechanism would require developing an ontological vocabulary (such as OWL-DL [53] or RDF [54]) for our policy definitions to allow automated reasoning of policies. We view this as a promising direction for the future of policy-based access control on the emerging policy-aware Web [55].

LIST OF REFERENCES

LIST OF REFERENCES

[1] P. Wolfowritz, "Data sharing in a net-centric department of defense," Tech. Rep. d8320.2, Department of Defense (DoD), DEC 2004.

[2] D. Seamon, "Deep sharing: A case for federated digital library," Tech. Rep. erm0348, EDUCAUSE, AUG 2003.

[3] S. D. C. di Vimercati and P. Samarati, "Access control in federated systems," in *Proceedings of the ACM New Security Paradigm Workshop, Lake Arrowhead, CA*, pp. 87–99, ACM Press, 1996.

[4] D. D. Clark and D. R. Wilson, "A comparison of commercial and military computer security policies," in *Proceedings of the Symposium on Security and Privacy, Oakland, CA*, pp. 184–194, IEEE Press, APR 1987.

[5] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[6] L. Lymberopoulos, E. C. Lupu, and M. Sloman, "An adaptive policy based management framework for network services management," *Journal of Networks and Systems Management*, vol. 11, no. 3, pp. 277–303, 2003.

[7] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The ponder specification language," in *Proceedings of the Workshop on Policies for Distributed Systems and Networks (Policy), LNCS 1995, Bristol, UK*, pp. 18–39, Springer, JAN 2001.

[8] E. C. Lupu and M. Sloman, "Conflicts in policy-based distributed systems management," *IEEE Transactions on Software Engineering*, vol. 25, no. 6, pp. 852–869, 1999.

[9] M. Thompson, A. Essiari, and S. Mudumbai, "Certificate-based authorization policy in a pki environment," *ACM Transactions on Information and System Security*, vol. 6, no. 4, pp. 566–588, 2003.

[10] D. Chadwick and A. Otenko, "The permis x.509 role based privilege management infrastructure," in *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT), Monterey, CA*, pp. 135–140, ACM Press, JUN 2002.

[11] "Shibboleth - specification - draft v1.0." http://shibboleth.internet2.edu/docs/-draft-internet2-shibboleth-specification-00.html, 2001.

[12] "Liberty alliance project specifications." http://www.projectliberty.org/resources/specifications.php, 2005.

[13] "Security assertion markup language (saml)." http://xml.coverpages.org/saml.html, 2004.

[14] "extensible access control markup language (xacml)." http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml, 2005.

[15] T. Fink, M. Koch, and K. Pauls, "An mda approach to access control specifications using mof and profiles," in *Proceedings of the 1st International Workshop on Views On Designing Complex Architectures (VODCA), Electronic Notes in Theoretical Computer Science 142, Bertinoro, Italy*, pp. 161–179, Elsevier, SEP 2004.

[16] J. Jurjens, "Umlsec: Extending uml for secure systems development," in *Proceedings of the 5th International Conference on The Unified Modeling Language, London, UK*, pp. 412–425, Springer, OCT 2002.

[17] T. Lodderstedt, D. A. Basin, and J. Doser, "Secureuml: A umlbased modeling language for model-driven security," in *Proceedings of the 5th International Conference on The Unified Modeling Language, London, UK*, pp. 426–4441, Springer, OCT 2002.

[18] "Resource access decision (rad), version 1.0." http://www.omg.org/technology/documents/formal/resource_access_decision.htm, 2001.

[19] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed nist standard for role-based access control," *ACM Transactions on Information and System Security*, vol. 4, no. 3, pp. 224–274, 2001.

[20] "Security in a web services world: A proposed architecture and roadmap." http://www-128.ibm.com/developerworks/library/specification/ws-secmap/, 2002.

[21] D. Carlson, *Modeling XML Applications with UML: Practical e-Business Applications*. Boston, MA: Addison-Wesley, 2001.

[22] "Common information model (cim) schema: Version 2.11." http://www.dmtf.org/standards/cim/cim_schema_v211, 2005.

[23] T. Verdickt, B. Dhoedt, F. Gielen, and P. Demeester, "Automatic inclusion of middleware performance attributes into architectural uml software models," *IEEE Transactions on Software Engineering*, vol. 31, no. 8, pp. 695–711, 2005.

[24] M. Blaze, J. Feigenbaum, and A. D. Keromytis, "Keynote: Trust management for public-key infrastructures," in *Proceedings of the 6th International Workshop on Security Protocols, LNCS 1550, Cambridge, UK*, pp. 59–63, Springer, APR 1998.

[25] C. Ellison, "Spki requirements," Tech. Rep. RFC 2692, Internet Engineering Task Force Draft (IETF), SEP 1999.

[26] A. Herzberg, Y. Mass, J. Mihaeli, D. Naor, and Y. Ravid, "Access control meets public key infrastructure, or: Assigning roles to strangers," in *Proceedings of the Symposium on Security and Privacy, Oakland, CA*, pp. 2–14, IEEE Press, MAY 2000.

[27] J. B. D. Joshi, R. Bhatti, E. Bertino, and A. Ghafoor, "Access control language for multidomain environments," *IEEE Internet Computing*, vol. 8, no. 6, pp. 40–50, 2004.

[28] N. Li, J. C. Mitchell, and W. H. Winsborough, "Design of a role-based trust management framework," in *Proceedings of the Symposium on Security and Privacy, Oakland, CA*, pp. 114–130, IEEE Press, MAY 2002.

[29] J. B. D. Joshi, E. Bertino, U. Latif, and A. Ghafoor, "A generalized temporal role based access control model," *IEEE Transaction on Knowledge and Data Engineering*, vol. 17, no. 1, pp. 4–23, 2005.

[30] R. Bhatti, J. B. D. Joshi, E. Bertino, and A. Ghafoor, "X-gtrbac: An xml-based policy specification framework and architecture for enterprise-wide access control," *ACM Transactions on Information and System Security*, vol. 8, no. 2, pp. 187–227, 2005.

[31] S. Kent and R. Atkinson, "Security architecture for the internet protocol," Tech. Rep. RFC 2401, Internet Engineering Task Force Draft (IETF), NOV 1998.

[32] A. D. Keromytis, S. Ioannidis, M. B. Greenwald, and J. M. Smith, "The strongman architecture," in *Proceedings of the 3rd DARPA Information Survivability Conference and Exposition (DISCEX), Washington D.C.*, p. 178, IEEE Press, APR 2003.

[33] R. Sandhu and Q. Munawer, "The arbac99 model for administration of roles," in *Proceedings of the 15th Annual Computer Security Applications Conference (ACSAC), Scottsdale, AZ*, pp. 229–240, ACM Press, DEC 1999.

[34] J. Crampton and G. Loizou, "Administrative scope and role hierarchy operations," in *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT), Monterey, CA*, pp. 145–154, ACM Press, JUN 2002.

[35] R. Sandhu, "Role activation hierarchies," in *Proceedings of the 3rd ACM Workshop on Role Based Access Control, Fairfax, VA*, pp. 33–40, ACM Press, OCT 1998.

[36] J. B. D. Joshi, E. Bertino, and A. Ghafoor, "Temporal hierarchies and inheritance semantics for gtrbac," in *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT), Monterey, CA*, pp. 74–83, ACM Press, JUN 2002.

[37] P. Fankhauser, G. Gardarin, M. Lopez, J. Munoz, and A. Tomasic, "Experiences in federated databases: From iro-db to miro-web," in *Proceedings of the 24th International Conf. on Very Large Databases (VLDB), New York, NY*, pp. 655–658, Morgan Kaufmann, AUG 1998.

[38] G. Gardarin, S. Gannouni, and B. Finance, "Iro-db: A distributed system federating object and relational databases," in *Object-Oriented Multi-database system: A solution for advanced applications*, Englewood Cliffs, NJ: Prentice Hall, 1995.

[39] D. Heimbigner and D. McLeod, "A federated architecture for information management," *ACM Transactions on Information Systems (TOIS)*, vol. 3, no. 3, pp. 253–278, 1985.

[40] M. R. andBandreddi E. Prasad andP.G.Reddy andAmar Gupta, "A methodology for integration of heterogeneous databases," *IEEE Transaction on Knowledge and Data Engineering*, vol. 6, no. 6, pp. 920–933, 1994.

[41] M. Tempelton, D. Brill, A. Chen, S. Dao, and E. Lund, "Mermaid: Experiences with network operation," in *Proceedings of the 2nd International Conference on Data Engineering (ICDE), Los Angeles, CA*, pp. 292–300, IEEE Press, FEB 1986.

[42] A. Tomasic, L. Raschid, and P. Valduriez, "Scaling access to heterogeneous data sources with disco," *IEEE Transaction on Knowledge and Data Engineering*, vol. 10, no. 5, pp. 808–823, 1998.

[43] R. H. Dolin, L. Alschuler, S. Boyer, C. Beebe, F. M. Behlen, and P. V. Biron, "Hl7 clinical document architecture, release 2.0," Tech. Rep. CDA-20040830v3, CDA, AUG 2004.

[44] L. Alschuler, R. H. Dolin, S. Boyer, C. Mead, and P. Elkin, "Layered constraints: The proposal for hl7 healthcare templates," in *Proceedings of the XML Conference and Exposition, Baltimore, MD*, DEC 2002.

[45] F. Moss, "Clinical record use cases, v0.1," Tech. Rep. draft-xacml-usecase-01a, OASIS XACML Technical Committee, SEP 2001.

[46] V. Kumar and S. Zidonik, "Workshop report," in *NSF Workshop on Context Aware Mobile and Sensor Information Management, Providence, RI*, JAN 2002.

[47] A. Tsiolakis and H. Ehrig, "Consistency analysis of uml class and sequence diagrams based on attributed typed graphs and their transformation," in *Proceedings of the Joint APPLIGRAPH/GETGRATS Workshop on Graph Transformation Systems (GRATRA), Berlin, Germany*, MAR 2000.

[48] A. Masood, R. Bhatti, A. Ghafoor, and A. Mathur, "Model-based testing of access control systems that employ rbac policies," Tech. Rep. 2005-62, Center for Education and Research in Information Assurance and Security (CERIAS), 2005.

[49] R. Bhatti, D. Sanz, E. Bertino, and A. Ghafoor, "A policy-based authorization system for web services:integrating x-gtrbac and ws-policy," Tech. Rep. 2006-03, Center for Education and Research in Information Assurance and Security (CERIAS), 2006.

[50] "Web services policy framework web services policy framework." http://www-128.ibm.com/developerworks/library/specification/ws-polfram/, 2006.

[51] R. Bhatti, M. Damiani, D. W. Bettis, E. Bertino, and A. Ghafoor, "A modular framework for administering spatial constraints in context-aware rbac," Tech. Rep. 2006-04, Center for Education and Research in Information Assurance and Security (CERIAS), 2006.

[52] E. Bertino, B. Catania, M. L. Damiani, and P. Perlasca, "Geo-rbac: a spatially aware rbac," in *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies (SACMAT), Stockholm, Sweden*, pp. 29–37, ACM Press, JUN 2005.

[53] "Owl web ontology language." http://www.w3.org/TR/owl-features/, 2004.

[54] "Resource description framework (rdf)." http://www.w3.org/TR/rdf-concepts/, 2004.

[55] D. Weitzner, J. Hendler, T. Berners-Lee, and D. Connolly, "Creating a policy-aware web: Discretionary, rule-based access for the world wide web," in *Web and Information Security*, Hershey, PA: Idea Group, (forthcoming).

APPENDICES

Appendix A: X-GTRBAC Grammar

[Basic Definitions]

```
<!-- Policy Definition>::=
<Policy policy_id=(xs:id)
        policy_name=(xs:name)>
  <!-- XML Credential Type Definitions>
  <!-- XML Separation of Duty Definitions>
  <!-- XML Temporal Constraint Definitions>
  <!-- XML Predicate Function Definitions>
  <!-- XML Resource Type Definitions>
  <!-- XML Resource Type Sheet>
  <!-- XML User Sheet>
  <!-- XML Role Sheet>
  <!-- XML Permission Sheet>
  <!-- XML User-Role Assignment Sheet>
  <!-- XML Permission-Role Assignment Sheet>
 </Policy>
<!-- XML Credential Type Definitions >
::= <XCredTypeDef xctd_id = (xs:id) >
{<!-- Credential Type Definition>}*
</XCredTypeDef>
<!-- Credential Type Definition> ::=
<CredTypeDef cred_type_id = (xs:id)
cred_type_name= (xs:name) >
<!Attribute List>
</CredTypeDef>
<!Attribute List >  ::= <AttributeList>
    {<!-- Attribute Definition>}*
</AttributeList >
<!-- Attribute Definition>::
<AttributeDef name=(xs: name) usage= mand | opt
type = xs:dateTime | xs:string | xs:integer />
<!-- XML User Sheet>     ::=
  <XUS xus_id = (xs:id) >
 {<!-- User Definition>}*
  </XUS>
  <!-- User Definition> ::=
  <User user_id = (xs:id)>
[<UserName><!-- NameID></UserName>]
<!CredType>
<MaxRoles>(xs:integer)</MaxRoles>
</User>
<!CredType >  ::=
<CredType cred_type_id = (xs:idref)
cred_type_name= (xs:name) >
[<!Header>]
<!-- Credential Expression>
</CredType>
<!-- Credential Expression >  ::= <CredExpr>
   {<!-- Attribute >}*
</CredExpr>
<!-- Attribute>::= <Attribute name= (xs:name)>
value= (xs:dateTime | xs:string | xs:integer) />
<!-- XML Role Sheet>::=
<XRS xrs_id = (xs:id)>
   {<!-- Role Definition>}*
</XRS>
<!-- Role Definition>::=
<Role role_id = (xs:id)
  role_name = (xs:name)>
    [<!-- Cred Type>]
[<!(En|Dis)abling Constraint>]
[<![De]Activation Constraint>]
{<SSDRoleSetId> (xs:idref)
</SSDRoleSetId>}*
    {<DSDRoleSetId> (xs:idref)
```

```
</DSDRoleSetId>}*
[<JuniorRoleId>(xs:idref)
</JuniorRoleId>]
    [<SeniorRoleId>(xs:idref)
</SeniorRoleId>]
{<!Linked Role ID>}*
   [<!Delegation Constraint>]
   [<Cardinality> (xs:integer)
</Cardinality>]
</Role>
<!Linked Role ID> ::=
<LinkedRoleId
id= (xs:idref)
type=delegator | delegatee />
<!-- XML Separation of Duty Definitions>
::=      <XSoDDef xsod_id = (xs:id) >
  [<!SSDRoleSets>]
  [<!DSDRoleSets>]
      </XSoDDef>
<!-- SSDRoleSets > ::=
<SSDRoleSets>
{<!SSDRoleSet>}+
</SSDRoleSets>
<!SSDRoleSet> ::=
<SSDRoleSet
ssd_role_set_id = (xs:id)
ssd_cardinality = (xs:integer)>
{<SSDRoleId>(xs:idref)
</SSDRoleId>}+
</SSDRoleSet>
<!-- DSDRoleSets >  ::=
<DSDRoleSets>
{<!DSDRoleSet>}+
</DSDRoleSets>
<!DSDRoleSet>::=
<DSDRoleSet
dsd_role_set_id =(xs:id)
dsd_cardinality = (xs:integer)>
{<DSDRoleId>(xs:idref)
</DSDRoleId>}+
</DSDRoleSet>
<!-- XML Permission Sheet>::=
<XPS xps_id = (xs:id) >
{<!-- Permission Definition>}+
</XPS>
<!-- Permission Definition>::=
<Permission
perm_id =(xs:id)
[prop= noprop|first_level|cascade ] >
<Object
res_type_id=(xs:idref) />
{<!Attribute >}*
</Object>
<!-- Operation>
</Permission>
<!-- Resource Type Definitions > ::=
  <XResTypeDef
xrtd_id = (xs:id) >
{<!Resource Type Definition>}*
</XResTypeDef>
<!-- Resource Type Definition> ::=
<ResTypeDef res_type_id = (xs:id)
res_type_name= (xs:name) >
<!-- Attribute List>
</ResTypeDef >
<!-- XML Resource Type Sheet>::=
<XRTS xrts_id = (xs:id) >
{<!-- Resource Type>}*
</XRTS>
<!-- Resource Type>::=
```

```
    <ResType res_type_id = (xs:idref)
res_type_name = (xs:name)>
{<!Attribute>}*
 </ResType>
<!-- Operation> ::= <Operation>
(saml:Action)</Operation>
<!-- XML User-Role Assignment Sheet>::=
<XURAS xuras_id = (xs:id) >
{<!-- User-role Assignment>}*
</XURAS>
<!-- User-role Assignment>::=
<URA ura_id=(xs:id) role_id=(xs:idref)>
{< ![De]Assign User>}+
</URA>
< ![De]Assign User>      ::=
<[De]AssignUser user_id=(xs:idref)>
 [<![De]Assign Constraint >]
</[De]AssignUser>
<!-- XML Permission-Role Assignment Sheet>
<XPRAS xpras_id = (xs:id) >
{<!-- Permission-Role Assignment>}*
</XPRAS>
<!-- Permission-Role Assignment>::=
<PRA pra_id=(xs:id)   role_id=(xs:idref)>
{< ![De]Assign Permission>}+
</PRA>
< ![De]Assign Permission>      ::=
<[De]AssignPermission
perm_id=(xs:idref)>
[<![De]Assign Constraint >]
</[De]AssignPermission>
<![De]Assign Constraint>::=
<[De]AssignConstraint
[op = AND|OR|NOT]>
// no opcode defaults to AND
{<![De] Assign Condition>}+
</[De]AssignConstraint>
<![De]Assign Condition>::=
<[De]AssignCondition
cred_type_id=(xs:idref)
[pt_expr_id=(xs:idref)|
d_expr_id=(xs:idref)]>
  [<!-- Logical Expression>]
</[De]AssignCondition>
<!(En|Dis)abling Constraint> ::=
<(En|Dis)abConstraint
[op = AND|OR|NOT]>
 // no opcode defaults to AND
{<!-- (En|Dis)abling Condition>}+
</(En|Dis)abConstraint>
<!(En|Dis)abling Condition>::=
<(En|Dis)abCondition
[pt_expr_id=(xs:idref) |
d_expr_id=(xs:idref)] >
   [<!-- Logical Expression>]
</(En|Dis)abCondition>
<![De]Activation Constraint> ::=
<[De] ActivConstraint
[op =  AND|OR|NOT]>
  // no opcode defaults to AND
{<![De]ActivationCondition>}+
</[De]ActivConstraint>
<![De]Activation Condition>::=
<[De]ActivCondition
[d_expr_id=(xs:idref)]>
 <!-- Logical Expression>]
</[De]ActivCondition >
<!-- Logical Expression>::=
<LogicalExpr [op = AND|OR|NOT]>
// no opcode defaults to AND
```

```
{<!-- Predicate>}+
</LogicalExpr>
<!-- Predicate>::=
<Predicate>
      <!-- PredicateBlock> |
< !--LogicalExpression>
</Predicate>
<!-- PredicateBlock>::=
<PredicateBlock>
    <Operator> gt|lt|eq|neq </Operator>
[<FuncId>(xs:idref)</FuncId>]
{<ParamName>(xs:name)</ParamName>}+
<RetValue>(xs:anyType)</RetValue>
</PredicateBlock>
<!-- XML Predicate Function Definitions>::=
<XPredFuncDef xpfd_id = (xs:id) >
{<!-- Function Definition>}*
</XPredFuncDef>
<!Function Definition>::=
<Function func_id = (xs:id)
func_name= (xs:name)
return_type= xs:anyType>
<!Parameter List>
    </Function>
<!Parameter List>::=
<ParameterList>
{<!-- Parameter>}*
   </ParameterList >
<!-- Parameter>::=
<Parameter order= (xs:int)
type = xs:string|xs:int|xs:date / >
```

[Temporal Definitions]

```
<!-- XML Temporal Constraint Definitions >::=
<XTempConstDef xtcd_id = (xs:id) >
      {<!Interval Expression>}*
      {<!-- Periodic Time Expression>}*
      {<!-- Duration Expression>}*
</XTempConstDef>
<!-- Periodic Time Expression>   ::=
<PeriodicTimeExpr pt_expr_id = (xs:id)
[d_expr_id = (xs:id)]
i_expr_id = (xs:id) ] >
 [d_expr_id = (xs:idref)]
[i_expr_id = (xs:idref)] >
<!-- Start Time Expression>
</PeriodicTimeExpr>
<!Interval Expression>::=
<IntervalExpr i_expr_id = (xs:id)>
  <begin> (xs:dateTime)</begin>
  <end>(xs:dateTime)</end>
</IntervalExpr>
<!-- Start Time Expression>::=
<StartTimeExpr
[pt_expr_id_ref = (xs:idref)]>
[<Year>all|odd|even</Year>]
[<!--MonthSet>]
[<!--WeekSet>]
[<!--DaySet>]
</StartTimeExpr>
<!--MonthSet>::=<MonthSet>
{<Month>1|..|12</Month>}1-12
(represents # of months from
the start of current Year)
</MonthSet >
<!--WeekSet>::=
<WeekSet>
{<Week>1|..|5</Week>}1-5
   (represents # of weeks from
the start of current Month)
</WeekSet >
```

```
<!--DaySet>::=
<DaySet>
{<Day>1|..|7</Day>}1-7
(represents # of days from
the start of current Week)
</DaySet >
<!-- Duration Expression>::=
<DurationExpr  d_expr_id = (xs:id)>
<cal>(Years|Months|Weeks|Days)</cal>
<len> (xs:integer)</len>
</DurationExpr>
```

[Credential Definitions]

```
<!--Header>            ::=
<Header>
 <Principal><!-- NameID></Principal>
  <Issuer><!-- NameID></Issuer>
  <!--  Validity>
  [<DSig> <!-- Signature ></DSig>]
  </Header>
<!-- NameID>::= (saml:NameID)
<!-- Validity> ::=
<Validity>
 <IssueTime>(xs:dateTime)</IssueTime>
  [<NotBefore>(xs:dateTime)</NotBefore>]
  [<NotOnOrAfter>(xs:dateTime)
</NotOnOrAfter>]
</Validity>
<!--  Signature >   ::= (ds:Signature)
<!Delegation Constraint>    ::=
<DelegationConstraint [op = AND|OR|NOT]>
  // no opcode defaults to AND
{<!-- Delegation Condition>}+
</DelegationConstraint>
<!Delegation Condition>          ::=
<DelegationCondition [pt_expr_id=(xs:idref) |
            d_expr_id=(xs:idref)] >
   [<!-- Logical Expression>]
</DelegationCondition>
```

Appendix B: UML Meta Model for X-GTRBAC Policy

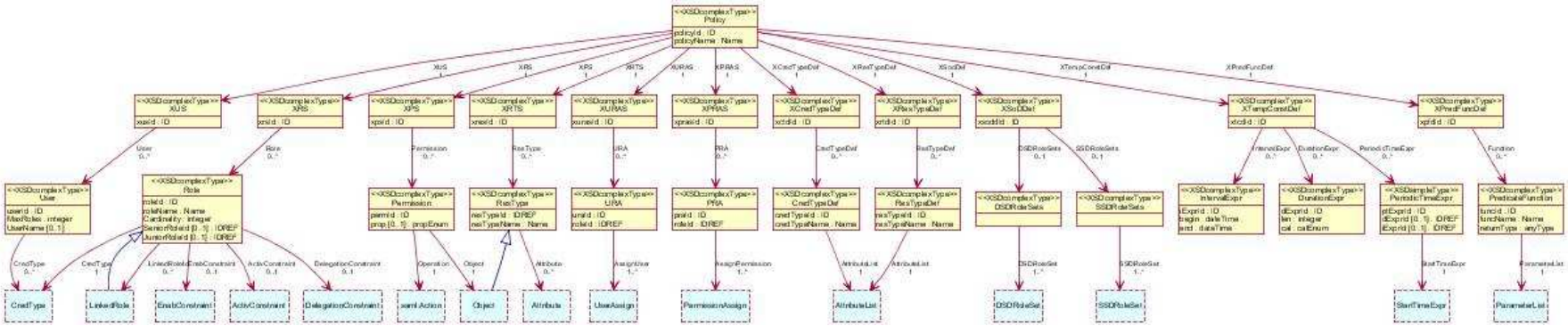

Fig. B.1. Top level UML model for XPolicy

Appendix C: Example Policy

Table C.1: XML documents for example policy definitions (in row major order): (a)XCredTypeDef (b)XUS (c)XTempConstDef(d)XPredFuncDef (e)XRS (f)XPS (g)XRedTypeDef (h)XReS

```
<?xml version="1.0" encoding="UTF-8"?>
<XCredTypeDef xctd_id = HCF_XCTD >
 <CredTypeDef cred_type_id = CDA_IHP
cred_type_name=
CDAIndividualHealthCarePractitioner >
 <AttributeList>
    <Attribute
name="board_certified_id" type="string" />
    <Attribute
name="fellowship_field_cd" type="string" />
  </AttributeList>
 </CredTypeDef >
<CredTypeDef cred_type_id = CP_PP
cred_type_name=
ClinicPurduePermittedPhysician >
 <AttributeList>
    <Attribute
name="location" type="string" />
  </AttributeList>
 </CredTypeDef >
<CredTypeDef cred_type_id = CP_BC
cred_type_name=
 ClinicPurdueBillingClerk />
</XCredTypeDef>
```

(a) The definition of the credentials CDAIndividualHealthCarePractitioner, ClinicPurduePermittedPhysician and ClinicPurdueBillingClerk.

```
<?xml version="1.0" encoding="UTF-8"?>
<XUS xus_id=HCF_XUS>
 <User user_id =any>
  <UserName/>
 <CredType cred_type_id=CDA_IHP
cred_type_name=
CDAIndividualHealthCarePractitioner>
  <CredExpr >
   <Attribute
name=board_certified_id
    value= />
   <Attribute
name=fellowship_field_cd
    value=/>
  </CredExpr>
 </CredType>
 </User>
 <User user_id =any>
  <UserName/>
 <CredType cred_type_id=CD_BC
cred_type_name=
ClinicPurdueBillingClerk />
 </User>
</XUS>
```

(b) The definition of a particular instance of the credentials defined in (a).

Continued on next page

```
<?xml version="1.0" encoding="UTF-8"?>
<XTempConstDef xtcd_id="HCF_XTCD">
  <IntervalExpr i_expr_id="Year2005">
        <begin>1/1/2005</begin>
        <end>12/31/2005</end>
  </IntervalExpr>
  <DurationExpr d_expr_id="OneWeek">
        <cal>Weeks</cal>
        <len>1</len>
  </DurationExpr>
<PeriodicTimeExpr
pt_expr_id="PTQuarterWeekOne"
i_expr_id="Year2005" d_expr_id="OneWeek">
    <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>
```

(c) This temporal constraint definition includes a periodic time expression (PTE) which states that the access is allowed beginning the first week of every quarter of year 2005. Note that duration expression and/or interval expression are referenced inside a PTE.

```
<?xml version="1.0" encoding="UTF-8"?>
<XPredFuncDef xpfd_id="HCF_XPFD">
 <Function func_id="fhCAV"
       func_name="hasCredAttributeValue"
return_type="xs:AnyType">
  <ParameterList>
  <Parameter order="1" type="xs:string" />
   </ParameterList>
 </Function>
</XPredFuncDef >
```

(d) This is the definition of predicate function hasCredAttributeValue. It has one parameter of type xs:string, and a return type of xs:anyType.

Continued on next page

```
<?xml version="1.0" encoding="UTF-8"?>
<XRS xrs_id="HCF_XRS">
 <Role role_id="rPhysicianPBob"
       role_name="PermittedPhysicianPBob" >
  <CredType
    cred_type_id="CP_PP"
  cred_type_name=
"ClinicPurduePermittedPhysician">
   <CredExpr >
    <Attribute name="name" value="" />
    <Attribute name="location" value="" />
   </CredExpr>
  </CredType>
 </Role>
 <Role role_id="rBillingClerk"
       role_name="BillingClerk" />
</XRS>
```

(e) The definition of the role PermittedPhysicianPbob and BillingClerk. The credential contains authorization attributes for the role that are used in the assignment policy of (j) for permission-role-assignment. Note that the value of role attributes is captured dynamically by the system and hence is not explicitly stated in the role definition.

```
<?xml version="1.0" encoding="UTF-8"?>
<XPS xps_id="HCF_XPS">
 <Permission perm_id="CP_PBob_CPrCD_GET">
  <Object res_type_id = "CPrCD" />
  <Operation namespace="saml:ghpp">
          GET</Operation>
 </Permission>
<Permission perm_id="CP_PBob_CPrDS_GET">
  <Object res_type_id = "CPrDS" />
  <Operation namespace="saml:ghpp">
          GET</Operation>
 </Permission>
 <Permission perm_id="CP_CPrCD_GET">
  <Object res_type_id = "CPrCD" />
  <Operation namespace="saml:ghpp">
          GET</Operation>
 </Permission>
</XPS>
```

(f) The definition of the permissions CP_PBob_CPrCD_GET, CP_PBob_CPrDS_GET and CP_CPrCD_GET. They define the permissions to perform GET action belonging to the saml:ghpp namespace on an EHR belonging to the categories CPrCD and CPrDS defined in (f). The first two permissions are specific to EHRs of patient Bob whereas the last one applies to EHR of any patient.

Continued on next page

```
<?xml version="1.0" encoding="UTF-8"?>
<XResTypeDef xrtd_id = "HCF_XRTD" >
    <ResTypeDef res_type_id = "CPrCD"
          res_type_name =
"ClinicPurdueResClinicalDocument">
<AttributeList>
   <Attribute name="id" type="anyURI"/>
    <Attribute
  name="change_reason_cd" type="string" />
    <Attribute
  name="completion_cd" type="string" />
    <Attribute
  name="confidentiality_cd" type="string" />
   <Attribute
  name="version_number" type="string" />
  </AttributeList>
 </ResTypeDef>
</XResTypeDef>
```

(g) The definition of a resource type ClinicalDocument. It declares a mandatory id attribute of the type anyURI, and a set of other ttributes of type string.

```
<?xml version="1.0" encoding="UTF-8"?>
<XReS xres_id = "HCF_XRES" >
    <Resource res_type_id = "CPrCD"
         res_type_name =
ClinicPurdueResClinicalDocument>
    <Attribute
name="id"  value= "CD_PBob_01.xml" />
 </Resource>
</XReS>
```

(h): The definition of an instance of the resource type ClinicPurdueResClinicalDocument. The resource is identified using the url value of the id attribute, which points to a resource instance belonging to patient Bob.

Table C.2: XML documents for example policy rules: (a)XURAS (b)XPRAS

```
<?xml version="1.0" encoding="UTF-8"?>
<XURAS xuras_id="HCF_XURAS">
 <URA ura_id="uraCDPatientPBob" role_id="rPhysicianCDPBob">
  <AssignUsers>
   <AssignUser user_id="any">
    <AssignConstraint>
      <AssignCondition cred_type_id= "CDA_IHP">
       <LogicalExpr op="AND">
        <Predicate>
         <Operator>eq</Operator>
         <FuncID>fhCAV</FuncID>
         <ParamName>board_certified_id</ParamName>
         <RetValue>NY</RetValue>
        </Predicate>
        <Predicate>
         <Operator>eq</Operator>
         <FuncID>fhCAV</FuncID>
         <ParamName>fellowship_field_cd
          </ParamName>
         <RetValue>GeneralMedicine</RetValue>
        </Predicate>
       </LogicalExpr>
      </AssignCondition>
    </AssignConstraint>
   </AssignUser>
  </AssignUsers>
 </URA>
 <URA ura_id="uraBillingClerk" role_id="rBillingClerk">
  <AssignUsers>
   <AssignUser user_id="any">
    <AssignConstraint>
      <AssignCondition cred_type_id= "nill"
      pt_expr_id="PTQuarterWeekOne" />
    </AssignConstraint>
   </AssignUser>
  </AssignUsers>
 </URA>
</XURAS>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<XPRAS xpras_id="HCF_XPRAS">
 <PRA pra_id="praPermittedPhysicianPBob" role_id="rPhysicianPBob">
  <AssignPermissions>
   <AssignPermission  perm_id="CP_PBob_CPrCD_GET">
    <AssignConstraint>
     <AssignCondition cred_type_id= "CDA_IHP">
      <LogicalExpr>
       <Predicate>
        <Operator>eq</Operator>
        <FuncID>fhCAV</FuncID>
        <ParamName>board_certified_id</ParamName>
        <RetValue>US</RetValue>
       </Predicate>
      </LogicalExpr>
     </AssignCondition>
    </AssignConstraint>
   </AssignPermission>
   <AssignPermission perm_id="CP_PBob_CPrDS_GET">
    <AssignConstraint>
     <AssignCondition cred_type_id= "ClinicPurduePermittedPhysician">
     <LogicalExpr op="AND">
      <Predicate>
       <LogicalExpr op="OR">
        <Predicate>
         <Operator>eq</Operator>
         <FuncID>fhCAV</FuncID>
         <ParamName>board_certified_id</ParamName>
         <RetValue>US</RetValue>
        </Predicate>
        <Predicate>
         <Operator>eq</Operator>
         <FuncID>fhCAV</FuncID>
         <ParamName>board_certified_id</ParamName>
         <RetValue>NY</RetValue>
        </Predicate>
        </LogicalExpr>
       <Predicate>
       <Predicate>
        <Operator>eq</Operator>
        <FuncID>fhCAV</FuncID>
        <ParamName>location</ParamName>
        <RetValue>NewYork</RetValue>
       </Predicate>
      </LogicalExpr>
     </AssignCondition>
    </AssignConstraint>
   </AssignPermission>
 </AssignPermissions>
</PRA>
 <PRA pra_id="praBillingClerk" role_id="rBillingClerk">
  <AssignPermissions>
   <AssignPermission  perm_id="CP_CPrCD_GET" />
 </AssignPermissions>
</PRA>
</XPRAS>
```

Continued on next page

(i) This is a role assignment policy for the PermittedPhysicianPBob and BillingClerk roles defined in (e). The policy for PermittedPhysicianPBob role states that any user can be assigned to this role if he/she supplies a CDA-encoded credential CDA_IHP. The predicate function defined in (d) is used to define conditions on credential attributes which correspond to Rule 6 in Section 5.3. The policy for BillingClerk role states that any user can be assigned to this role if he/she supplies a credential ClinicPurdueBillingClerk, and subject to the temporal constraint as defined in (c) which corresponds to Rule 3 in Section 5.3.	(j) This is a permission assignment policy for the roles defined in (e). It gives the conditions under which the permissions defined in (f) can be assigned to these roles. The respective constraints are defined on the credential for the role. The constraints correspond to Rules 1, 2, and 5 in Section 5.3.

VITA

VITA

Rafae Bhatti is a PhD candidate in the Department of Electrical and Computer Engineering and affiliated with the Center for Education and Reserach in Information Assurance and Security (CERIAS) at Purdue University. He received his B.S. degree in Electronics Engineering from GIK Institute, Pakistan in 1999 and his M.S. degree in Computer Engineering from Purdue University in 2003. His research interests include information systems security, with emphasis on design and administration of access management policies in distributed systems. In his M.S. thesis research at Purdue, he developed an XML-based policy specification framework for distributed access control. His PhD research focuses on the access management problems posed by the emerging federated paradigm of information sharing and collaboration, and on specification of XML-based security protocols for Web-based information systems. His work on XML-based access control framework for the Role Based Access Control (RBAC) model has recently been cited by the OASIS consortium in their official announcement of the RBAC standard.

Rafae will assume a position as an Assistant Professor at Florida International University effective Fall 2006. He is a student member of the ACM, IEEE and the IEEE Computer Society.