Managing Security Policies in a Distributed Environment Using eXtensible Markup Language (XML)*

Nathan N. Vuong, Geoffrey S. Smith
Florida International University
School of Computer Science
Miami, Florida USA
{nvuong01, smithg}@cs.fiu.edu

Yi Deng
University of Texas at Dallas
School of Engineering and Computer Science
Embedded Software Center
Richardson, Texas USA
yideng@utdallas.edu

ABSTRACT

In light of the growth of the Internet, much research has been done on application-level distributed authorization systems. Another area of research that is just as important, but has received little attention, is the management of security policies in a distributed environment. This paper describes practical concepts that can be employed in an enterprise environment for managing security policies using eXtensible Markup Language (XML). An example is given using our proposed concepts with Java[1] and Role-Based Access Control (RBAC) policies.

Keywords

Managing security policies, RBAC, XML, Java, distributed authorization, meta-language.

1. INTRODUCTION

An authorization system regulates and enforces access of principals to computing resources according to a prescribed policy. Its services are critical in ensuring the confidentiality, integrity, and accountability of shared data in a computing environment. It is useful to separate authorization into two sub-categories, *policy* and *mechanism* [4] as depicted in Figure 1. An access control *policy* specifies the authorized accesses of a principal whereas an access control *mechanism* implements or enforces the policy. The advantages of this separation are:

(1) It allows researchers to address each sub-category independently.

(2) A security policy can be enforced by different protection mechanisms.

(3) A single protection mechanism can enforce multiple security policies.

Historically, practitioners' approach toward providing authorization has been to code the authorization logic as part of the application. The reasons behind this approach are:

(1) Most operating systems, which the application resides on, provide minimal support for business authorization logics.

(2) The lack of a distributed authorization framework.

(3) Developers failed to abstract business logics from authorization logics, and to clearly separate authorization policy from enforcement mechanisms.

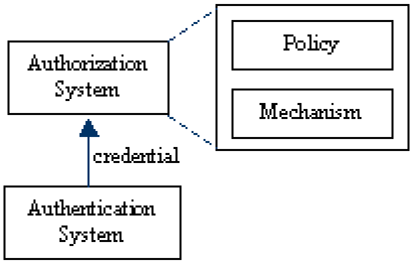


Figure 1. Authorization Model

But the above practices give rise to obvious problems in interoperability and scalability.

Realizing the need for a distributed authorization system that is interoperable, flexible, and manageable, researchers have proposed frameworks such as OMG Resource Authorization Decision (RAD) Specification [12], CORBA Security Services [11], and Secure European System in A Multi-vendor Environment (SESAME) [1] as a solution. These proposed frameworks provide a means to architect distributed authorization systems that separate security logics from application logics. Other research such as [3][8][16] provides notations, logics, and calculi for expressing and reasoning about security policies. But these works mainly concern modeling policies and enforcement mechanisms and put little emphasis on managing security policies.

The little research [2][5][7] that addresses managing security policies employs translating agents or software modules for communicating between disparate policies to achieve a cohesive corporate-wide security policy implementation. This concept requires a translating agent for each different access control implementation. Here, the granularity of an access control

SAC 2001, Las Vegas, NV
© 2001 ACM 1-58113-324-3/01/02...$5.00

*This work was supported in part by NSF under grant No. HDR-9707076.

[1] Java is a trademark of Sun Microsystems.

system depends on the native implementations, which implies that the lowest common denominator of all the systems must be adopted.

Our research proposes concepts that facilitate managing security policies in a distributed environment and can complement any distributed authorization framework. Our concepts explore a structured language model for expressing security policies. Our observation is that a structured language is more expressive than a traditional access control list (ACL). We employ a *meta-language* to define a grammar that can precisely and effectively represent the desired security policies. While the developed grammar provides the syntactic representation of security policies, a separate application programmer interface (API) is used to provide semantics to the grammar, and is used to maintain the integrity and consistency of the *authority state* [9]. This separation allows the semantics of security policies to be independently defined and to be separated from policy representations. The concept of a standardized policy *schema* is adopted for enforcing consistent policy representation. This requires all participating systems to validate the security policy structure against the specified schema. Additionally, it will allow a single administration tool, built according to the standardized schema, to be able to load, manage, and administrate security policies for the entire enterprise. With these proposals, an organization could effectively model, implement, and manage its security policies with confidence that interoperability, flexibility, and manageability are achieved.

The remainder of this paper is organized as follows. Section 2 provides the necessary background on Role-Based Access Control (RBAC) and eXtensible Markup Language (XML) to understand our work. Section 3 provides details on our proposed concepts to include designs and implementation. Finally, Section 4 concludes the paper.

2. BACKGROUND

We present here some basic concepts of Role-Based Access Control reference models, and then a brief overview of eXtensible Markup Language (XML) technology.

2.1 Role-Based Access Control (RBAC)

Role-Based Access Control (RBAC), a policy neutral access control mechanism, is widely known as being an inherently easier and less error-prone way of administrating access control policies. The basic principle of RBAC is the separation of *permission assignments* (PA) and *user assignments* (UA). With RBAC, permissions are assigned to roles and roles are assigned to users. A user thereby acquires the permissions assigned to that specific role. A user's permissions are limited to the roles in which he or she is authorized to function. The separation facilitates the administration of security policy, where each process can then be administered independently. Since permissions are de-coupled from users, changes to permission or user assignments have minimal isolated impact on administration.

The RBAC security model is abstract and general. This is indicated by the many interpretations of the RBAC model provided by researchers. Of the existing interpretations, Sandhu et al. [13] provide the most comprehensive and intuitive interpretation, capturing the vital and salient features of RBAC. They identify a family of RBAC reference models: RBAC$_0$ – base model, RBAC$_1$ – hierarchical model, RBAC$_2$ – constraints model, and RBAC$_3$ – all-inclusive model. RBAC$_0$ is the base reference model consisting of the basic essential elements for providing an RBAC service, with RBAC$_1$ through RBAC$_3$ built on RBAC$_0$ with added functionality such as role hierarchies and constraints. Figure 2 shows the RBAC models hierarchy.

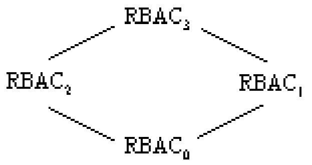


Figure 2. RBAC Models Hierarchy

Sandhu et al. formalize RBAC as follows:

- *U, R, P,* and *S* represent the finite set of users, roles, permissions, and sessions respectively within the system.
- $PA \subseteq P\ x\ R$ represents the finite set of permission to role assignments. This is a many-to-many relationship.
- $UA \subseteq U\ x\ R$ represents the finite set of user to role assignments. This is a many-to-many relationship.
- *user:* $S \rightarrow U$, a function that maps a session s_i to a user.
- $RH \subseteq R\ x\ R$ is a partial order on *R*, called the role hierarchy or role dominance relation, also written as ≥.
- For RBAC$_0$, *roles:* $S \rightarrow 2^R$, a function that maps a session s_i to a set of roles, where $roles(s_i) \subseteq \{r \mid (user(s_i), r) \in UA\}$, and each session s_i has the permissions $\cup_{r \in roles(si)} \{p \mid (p, r) \in PA\}$; that is, the permissions available to the user are the union of permissions from all roles activated in that session.
- For RBAC$_1$, *roles:* $S \rightarrow 2^R$ is modified from RBAC$_0$ to require $roles(s_i) \subseteq \{r \mid (\exists r' \geq r)[(user(s_i), r') \in UA]\}$ and each session has the permissions $\cup_{r \in roles(si)} \{p \mid (\exists r' \leq r)[(p, r') \in PA]\}$.
- RBAC$_2$ adds constraints in the form of restrictive functions that operate on RBAC components to meet the specific needs of an organization's protection policies. Typical constraints include *separation of duties* (also known as mutually exclusive roles) and *cardinalities* to limit the number of authorized roles.

Figure 3 provides a graphical depiction of the RBAC reference models. An abstract representation, *permission* is commonly understood as an approval for a particular mode of access to one or more objects in the system. Terms such as authorization, access right, privilege, and transaction have also been used in related literature to denote permission. From the RBAC perspective, the exact nature of permissions in a system is left open to implementation.

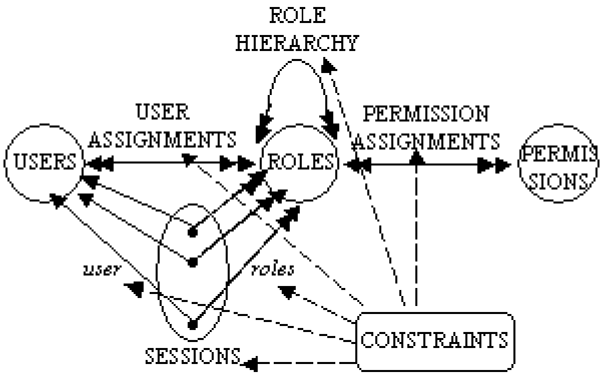


Figure 3. RBAC Reference Models

2.2 eXtensible Markup Language (XML)

The XML specification [17] is the work of the World Wide Web Consortium (W3C) Standard Generalized Markup Language (SGML) Working Group. It is designed as a meta-language for Internet use. Its objectives are to overcome the rigid HyperText Markup Language (HTML) tagging scheme while providing Web users with a means for defining their own domain specific tags and attributes.

2.2.1 XML Documents

An XML document has both a *logical* and a *physical* structure. The logical structure is composed of declarations, elements, comments, character references, and processing instructions, all of which are indicated in the document by explicit markup. The physical structure is composed of storage units called *entities*. An entity may reference other entities to cause their inclusion in the document. A document begins in a "*root*" or document entity, and all the logical and physical structures can be nested properly inside the document entity. An XML document may also be a *well-formed* and/or *valid* document. A *well-formed* document, in its entirety and expanded state, must conform to the production labeled *document*. A *valid* document must conform to the referenced schema, *Document Type Definition* (DTD). These two features can be employed to enforce the conformity and consistency of documents.

2.2.1.1 Logical Structure

Each XML document contains one or more *elements*, the boundaries of which are either delimited by start-tags and end-tags, or an empty-element tag. Each element has a type identified by name and may have a set of attribute specifications. Each attribute specification is made up of a name-value pair. The element structure of an XML document may be constrained by using element type and attribute-list declarations. An example element type declaration is:

```
<!ELEMENT memo (header, text) >
```

Here the `memo` element is composed a `header` and a `text` element. Element type declarations dictate which element types can appear as children of the element. Element declarations are logically grouped inside a DTD. An example of attribute-list constraints is:

```
<!ELEMENT header EMPTY>
<!ATTLIST header
  from    CDATA  #REQUIRED
  to      CDATA  #REQUIRED
  subject CDATA  #REQUIRED>
```

Here a `<header>` tag must provide attribute value for the `from`, `to`, and `name` attributes.

2.2.1.2 Physical Structure

An XML document may consist of one or more storage units called *entities*. Each entity consists of a *content* and a *name*. All XML documents have the *document entity* that serves as the starting point for the XML parser and may contain the whole document. A *parsed entity's* contents are referred to as its replacement text and are considered an integral part of the document. Entities are commonly used for physical modeling. An example of an entity definition is as follows:

```
<!ENTITY % text_type " TYPE (PARAGRAPH |
  SUMMARY | ABSTRACT) #REQUIRED ">
```

where `text_type` denotes the entity's name, and `"TYPE (PARAGRAPH | SUMMARY | ABSTRACT) #REQUIRED"` denotes the entity's content. Then the `text_type` entity can be referenced as follow:

```
<!ELEMENT text EMPTY>
<!ATTLIST text %text_type;>
```

When the XML parser parses `%text_type`, it replaces the referenced entity with the actual `text_type` content.

3. OUR WORK

Our research provides concepts for managing security policies in a distributed environment to include representation and administrative evaluation. Our concept for representing security policy is to use a structured language model. A structured language is more expressive and flexible than a traditional access control list (ACL). Properly designed, a structured language is closer to natural language than any other method for representing security policies. For evaluating and ensuring a consistent authority state, we propose an administrative application program interface (API) as an interface between an administrative application and the authority state. For consistent policy representation and ease of policy administration, we propose a standardized policy schema.

3.1 Implementing RBAC Policy Using Java and XML Technologies

Our research employs XML for syntactic representation of knowledge. XML, being a meta-language, provides accessible notations and means for us to describe an RBAC conceptual model. The developed RBAC grammar, based on Sandhu's RBAC reference models [13], is a domain specific grammar that can effectively represent various RBAC policies.

3.1.1 XML Logical Model

We model each RBAC component as an XML *element*:
A *User* is represented as

```
<!ELEMENT USER EMPTY>
<!ATTLIST USER
```

```
  NAME    ID    #REQUIRED>
```

The above syntax defines a new XML tag of type USER with a required NAME attribute of type ID that by default is unique.

A *Role* is represented as

```
<!ELEMENT ROLE EMPTY>
<!ATTLIST ROLE
  TITLE   ID    #REQUIRED>
```

A *Permission* is implementation-specific; therefore, we model it as an abstract representation that requires definition when defining policies. A Permission is represented as

```
<!ELEMENT PERMISSION EMPTY>
<!ATTLIST PERMISSION %DEFINITION;>
```

A *Permission Assignment* assigns a set of permissions to a role; it is represented as

```
<!ELEMENT PERMISSION_ASSIGNMENT EMPTY>
<!ATTLIST PERMISSION_ASSIGNMENT
  ROLE           IDREF  #REQUIRED
  PERMISSIONS    IDREFS #REQUIRED>
```

Using attribute constraints, the above definition requires that both ROLE and PERMISSIONS attribute must reference predefined values.

A *Role Assignment* assigns a set of users to a role; it is represented as

```
<!ELEMENT ROLE_ASSIGNMENT EMPTY>
<!ATTLIST ROLE_ASSIGNMENT
  ROLE    IDREF  #REQUIRED
  USERS   IDREFS #REQUIRED>
```

A *Role Hierarchy* is represented as a set of INHERITS elements, each of which associates a set of junior roles to a senior role:

```
<!ELEMENT INHERITS EMPTY>
<!ATTLIST INHERITS
  FROM    IDREFS #REQUIRED
  TO      IDREF  #REQUIRED>
```

With the defined RBAC components as XML elements, then an instance of an RBAC model is the composition of various RBAC components. For example, an $RBAC_1$ security model is represented as a production rule.

```
<!ELEMENT RBAC1_MODEL (USER+, ROLE+,
  INHERITS*,PERMISSION+,
  PERMISSION_ASSIGNMENT*,
  ROLE_ASSIGNMENT*)>
```

Here, the RBAC1_MODEL production is composed of one or more USER elements, one or more ROLE elements, zero or more INHERITS elements, one or more PERMISSION elements, zero or more PERMISSION_ASSIGNMENT elements, and zero or more ROLE_ASSIGNMENT elements. The above declarations constitute the $RBAC_1$ model grammar.

3.1.2 Representing a Hypothetical RBAC Policy

To demonstrate the practicality of our concepts, we've developed a hypothetical RBAC policy for a health care institution. For clarity and due to the lack of RBAC notations for expressing RBAC policies, we employed tables and graphs to informally describe the RBAC policy. Figure 4, Table 1, Table 2, and Table 3 depicts the hypothetical RBAC policy.

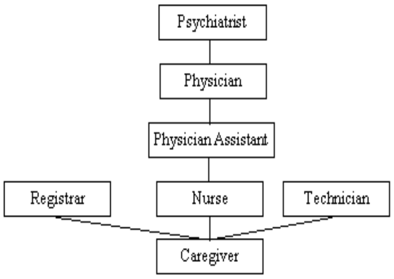
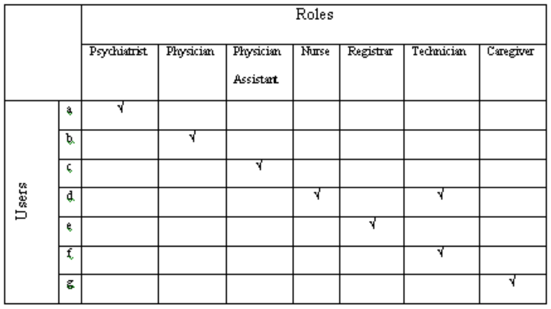
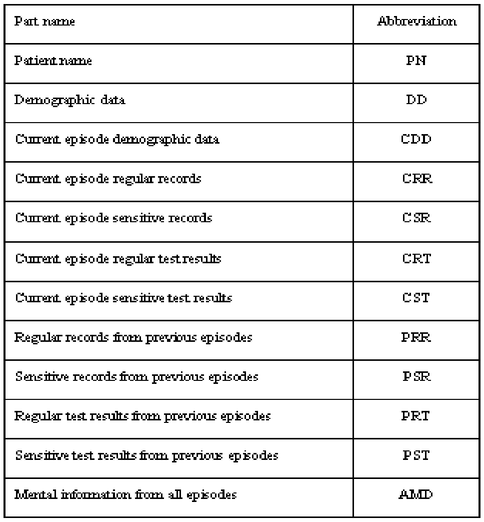


Figure 4. Role Hierarchy Relation



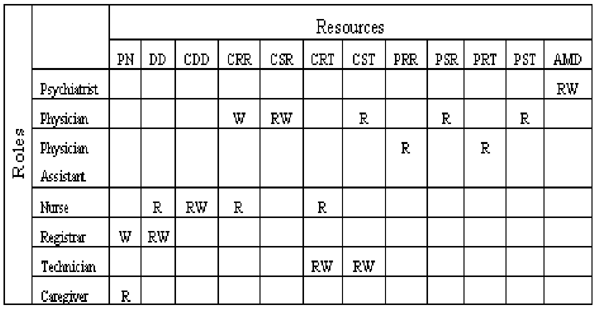
		Roles						
		Psychiatrist	Physician	Physician Assistant	Nurse	Registrar	Technician	Caregiver
Users	a	√						
	b		√					
	c			√				
	d				√		√	
	e					√		
	f						√	
	g							√

Table 1. User to Role Assignment (UA) Relation



Part name	Abbreviation
Patient name	PN
Demographic data	DD
Current episode demographic data	CDD
Current episode regular records	CRR
Current episode sensitive records	CSR
Current episode regular test results	CRT
Current episode sensitive test results	CST
Regular records from previous episodes	PRR
Sensitive records from previous episodes	PSR
Regular test results from previous episodes	PRT
Sensitive test results from previous episodes	PST
Mental information from all episodes	AMD

Table 2. Resource Description



		Resources											
		PN	DD	CDD	CRR	CSR	CRT	CST	PRR	PSR	PRT	PST	AMD
Roles	Psychiatrist												RW
	Physician				W	RW		R		R		R	
	Physician Assistant								R		R		
	Nurse		R	RW	R		R						
	Registrar	W	RW										
	Technician						RW	RW					
	Caregiver	R											

Table 3. Permission to Role Assignment (PA) Relation

Using the developed RBAC grammar, the abbreviated XML representation is as follows:

```
<?xml version="1.0" encoding="UTF-8"
standalone="no" ?>

<!DOCTYPE RBAC1_MODEL SYSTEM
"htpp://www.cs.fiu.edu/~nvuong01/
 RBAC1_MODEL.dtd">

<RBAC1_MODEL TYPE_NAME="RBAC1_POLICY">

    <!-- User set definition -->
    <USER NAME="a"></USER>
    <USER NAME="b"></USER>
    ...

    <!-- Role set definition -->
    <ROLE TITLE="Caregiver"></ROLE>
    <ROLE TITLE="Nurse"></ROLE>
    ...

    <!-- Role hierarchy definition -->
    <INHERITS FROM="Caregiver"
     TO="Registrar"></INHERITS>
    <INHERITS FROM="Caregiver"
     TO="Nurse"></INHERITS>
    ...

    <!-- Permission set definition -->
    <PERMISSION PERMID="P1"
     OPERATION="RW" RESOURCE="AMD">
    </PERMISSION>
    <PERMISSION OPERATION="R" PERMID="P2"
     RESOURCE="PST"></PERMISSION>
    ...

    <!-- Permission assignment -->
    <PERMISSION_ASSIGNMENT
     ROLE="Psychiatrist"
     PERMISSIONS="P1">
    </PERMISSION_ASSIGNMENT>
    <PERMISSION_ASSIGNMENT
     PERMISSIONS="P2 P4 P6 P10 P11"
     ROLE="Physician">
    ...

    <!-- Role assignment -->
    <ROLE_ASSIGNMENT ROLE="Psychiatrist"
     USERS="a"></ROLE_ASSIGNMENT>
    <ROLE_ASSIGNMENT ROLE="Technician"
     USERS="d f"></ROLE_ASSIGNMENT>
    ...

</RBAC1_MODEL>
```

3.1.3 Standardized Schema

For consistent policy representation, we propose a standardized policy schema. This ensures that the represented policy used by participating systems is consistent across the enterprise; therefore interoperability can be assured. Additionally, the standardized schema allows us to maintain a single administration tool. This tool, built in accordance to the schema, is able to load, manage, and administrate security policies for the entire enterprise.

We employed *valid* and *well-formed* features of XML to implement the proposed concept. This requires the participating XML document to conform to a specified DTD or schema located at the specified Uniform Resource Identifiers (URI) [10]. At runtime, the XML parser validates the policy's structure against the schema and reports any violations confronted. For our RBAC representation, the schema was referenced using a secured HyperText Transport Protocol (HTTPS).

3.1.4 Semantics Through an Application Program Interface (API)

XML was used to develop the RBAC grammar that provides the syntactic representation of knowledge. To provide semantics to the represented data, we propose a separate application program interface (API). This layered approach allows us to separate data representation from its semantics, and allows us to independently modify each layer's implementation without affecting the other layer. Additionally, the API also implements a list of administrative operations that serves as the interface for an external administrative application. The concept of *pre-conditions* and *post-conditions* were used to ensure that the integrity and consistency of the RBAC *authority state* [9] are maintained. Through the interface, the administration tool can safely and transparently manipulate the represented information. Our API implementation is an extension of NIST work [6] that includes operations such as *addRole*(), *removeRole*(), *assignRole*(), *removeAssignedRole*(), etc.

Our prototype of the API is in Java; therefore the implementation can run on any platform that supports the Java 2 virtual machine (VM). Figure 4 is an architectural depiction of our current implementation; Figure 5 provides a closer look at our prototyped Administration Tool.

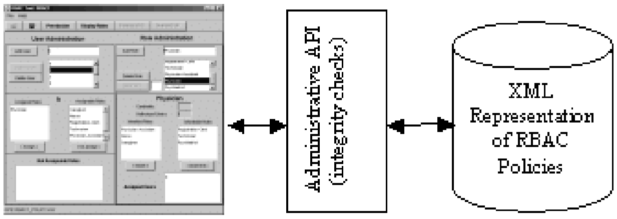


Figure 4. Administration Tool Architecture

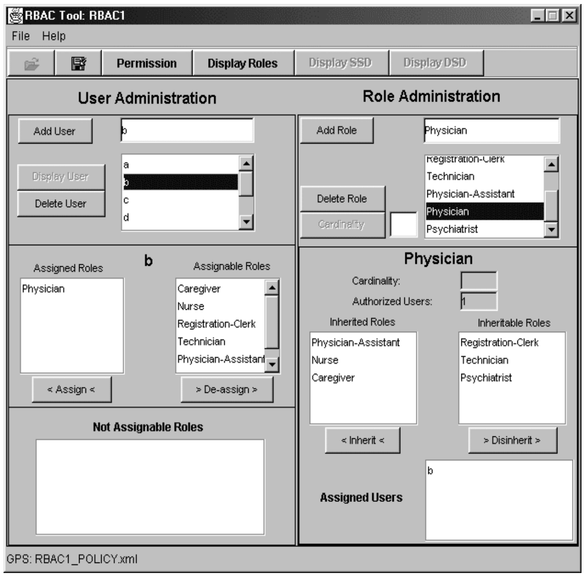


Figure 5. Administration Tool Interface

3.2 Managing Security Policies

By employing code and data mobility from Java and XML respectively, the administration tool is capable of loading an RBAC policy, located by a URI, from across a network. This enables a centrally located administrator to administer security policies for the entire enterprise. The underlying XML representation is transparent to the administrator; this allows for various administrative views to be built on top of the represented data. Additionally, the tool provides a less error prone and ease of administration by enforcing consistent policy administration. These are ideal attributes for an enterprise that requires large and complex security policies.

4. CONCLUSION

We have presented a new approach to managing security policies in a distributed environment. We claim that by adopting a structured language, a separate semantics API, and a standardized policy schema model to represent and implement security polices, we can achieve properties such as interoperability, flexibility, and manageability. Unlike most existing implementations, with our approach the semantics of authorization is independently defined and is separated from policy representation and from implementation mechanisms. We have demonstrated our concepts using XML and Java. XML, a meta-language, provides a very accessible notation for expressing the key elements in a conceptual model for an application domain. The flexibility and simplicity of the XML format allows researchers to design new domain-specific markup languages.

We believe that our concept can be applied to develop a generalized security language for expressing any security policy for a distributed environment, similar to [14][15]. With the proliferation of XML in the industry, there is a high probability that future systems will be equipped with an XML parser. This will help in realizing our views and concepts.

To further extend the proposed concept, we are experimenting with Java's ability to load Java *class* files across a network. This will allow us to maintain a centralized semantic implementation of the semantics API, which is in line with the concept of mandatory access control (MAC).

5. REFERENCES

[1] P. Ashley and B. Broom. 1997. An Implementation of the SESAME Security Architecture for Linux. *Australian Unix and Open Systems Group Technical Conference*.

[2] Roland Awischus. 1998. Access Control with the Security Administration Manager (SAM). *2nd ACM Workshop on Role-Based Access Control*. Fairfax, Va.

[3] Y. Bai and V. Varadharajan. 1997. A Logic for State Transformations in Authorization Policies. *Proceedings of the IEEE Computer Security Foundations Workshop*. June 1997.

[4] Dorothy E. Denning. 1982. Cryptography and Data Security. Addison-Wesley Publishing Company, 191-259.

[5] W. Essmayr, E. Kapsammer, R. R. Wagner, G. Pernul, A. M. Tjoa. 1998. Enterprise-Wide Security Administration. *Annual IEEE Computer Application Security Conference*.

[6] Serban I. Gavrila and John F. Barkley. 1998. Formal Specification for Role Based Access Control User/Role and Role/Role Relationship Management. *3rd ACM Workshop on Role-Based Access Control*. Fairfax, Va.

[7] John Hale, Pablo Galiasso, Mauricio Papa, and Sujeet Shenoi. 1999. Security Policy Coordination for Hetergeneous Information Systems. *Annual IEEE Computer Application Security Conference*.

[8] S. Jajodia, P. Samarati, and V.S. Subrahmanian. 1997. A Logical Language for Expressing Authorizations. *Proceedings of the IEEE Symposium on Security and Privacy*. May 1997.

[9] Jonathan D. Moffett and Emil C. Lupu. 1999. The Uses of Role Hierarchies in Access Control. *4th ACM Workshop on Role-Based Access Control*. Fairfax, Va.

[10] Internet Engineering Task Force, Network Working Group. RFC 2396 – Uniform Resource Identifiers (URI): Generic Syntax. 1998. http://www.ietf.org.

[11] OMG CORBAservices Common Object Services Specification: CORBA Security Services v1.2. December 1998.

[12] OMG CORBAmed DTF. Resource Access Decision (RAD), Revised Submission. 26 April 1999.

[13] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. 1996. Role-Based Access Control Models. *IEEE Computer*, 29(2):38-47.

[14] Mark Vandenwauver, René Govaerts, Joos Vandewalle. 1997. How Role Based Access Control is implemented in SESAME. *Proceedings of the 6-th Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 293-298. IEEE Computer Society Press.

[15] Vijay Varadharajan, Chris Crall, and Joe Pato. 1998. Authorization in Enterprise-wide Distributed System, A Practical Design and Application. *14th Annual Computer Security Application Conference*.

[16] T.Y.C. Woo and S.S. Lam. 1992. Authorizations in Distributed Systems: A Formal Approach. *Proceedings of the IEEE Symposium on Research in Security and Privacy*. 1992, pp. 33-50.

[17] Extensible Markup Language (XML) 1.0 – W3C Recommendation 10-Feb-98. HTTP://www.w3.org/TR/1998/REC-xml-19980210.