CERIAS Tech Report 2003-26

ACCESS CONTROL IN DYNAMIC XML-BASED WEB SERVICES WITH X-RBAC

by Rafae Bhatti, James B. D. Joshi,
Elisa Bertino, Arif Ghafoor

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

Access Control in Dynamic XML-based Web-Services with X-RBAC

Rafae Bhatti[1],
James B. D. Joshi[1],
Elisa Bertino[2],
Arif Ghafoor[1]

[1]School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN

[2]Dipartimento di Scienze dell'Informazione, Università di Milano, Milano, Italy

Abstract

Policy specification for securing Web services is fast emerging as a key research area due to rapid proliferation of Web services in modern day enterprise applications. Whilst the use of XML technology to support these Web services has resulted in their tremendous growth, it has also introduced a new set of security challenges specific to these Web services. Though there has been recent research in areas of XML-based document security, these challenges have not been addressed within the XML framework. In this paper, we present X-RBAC, an XML-based RBAC policy specification framework for enforcing access control in dynamic XML-based Web services. An X-RBAC system has been implemented as a Java application, and is based on a specification language that addresses specific security requirements of these Web services. We discuss the salient features of the specification language, and present the software architecture of our X-RBAC system.

Keywords: XML, RBAC, Access Control, Dynamic Web-Services.

*Portions of this work were supported by Grant IIS-0209111 from the National Science Foundation, and by sponsors of the Center for Education and Research in Information Assurance and Security.

1. Introduction

With the advent of massively distributed computing systems requiring secure interoperation, and the demand for sharing online information content across various Internet applications, Web services security is becoming an increasingly important task. These Web services introduce a new set of security challenges that are not addressed by traditional security models. Key amongst these challenges is to develop access control models that provide content-based context-aware access, and handle the heterogeneity of subjects and objects. This means that the access control model should capture security-relevant content and context information, such as time, location, or environmental state, available at the time the access requests are made, and incorporate it in its access control decisions. The object heterogeneity implies that the requested resource may be one of several media types available to the system, or even represent an abstract concept. Heterogeneity of subject implies that the user profile may change dynamically, and hence need to be constantly updated and incorporated into access control decisions. Such profiles are used by Web services to authenticate transfer of a user from one domain to another based on his/her current login information. For example, a registered user at an online bookstore could be offered a complementary purchase at a partner Web site based on his/her net dollars worth of purchases. Handling both the object and subject heterogeneity complicates access control specification. These specific challenges have not been addressed in recent research on Web-based document security models.

In this paper, we present a Java-based system for implementing Role-Based Access Control (RBAC) policy specification for addressing the access control needs of dynamic XML-based Web services. The RBAC model [Fer01a] is characterized by the notion that permissions are assigned to "roles", and not directly to users. Users are assigned appropriate "roles" according to their job functions, and hence indirectly acquire the permissions associated with those roles. Our implementation is based on an XML-based RBAC policy specification language outlined in [Bha03]. Therein, we have proposed a specification language based on an Extended RBAC model that incorporates the content and context-based dynamic access control requirements of XML-based Web services. Here we use an updated version of that specification language to develop an X-RBAC system in Java that supports the specification of RBAC policies for secure Web services. The schema specifications of the language are attached as appendices to the paper.

The remainder of this paper is organized as follows. A compendium of related work is provided to motivate the utility of X-RBAC. In section 3, we briefly discuss the salient features of the specification language. The discussion not only summarizes but also updates portions of the original specification. Section 4 discusses the system architecture of X-RBAC, illustrated throughout by sample policy instances to highlight the various functionalities of the system. We conclude the paper by discussing some of our future implementation goals.

2. Related Work and Motivation

Securing Web-based documents and applications using XML and/or RBAC has been the topic of many recent research publications. An XML-based language to secure Web-based XML documents, and its Java-based implementation, Author-X, has been discussed in [Ber01a,Ber01b]. A related effort has been reported in [Dam02]. An XML access control language, XACL, and its integration in a Web application, has been discussed in [Had00]. All these efforts are non-RBAC and have been based on XML DTDs, and not on XML schemas, and hence lack the set of enhanced capabilities that XML schema provides over DTD [URLb]. Much recent work has been reported in the literature about implementing the RBAC model in various application domains. The RBAC model is widely accepted because of its generality, which allows it to be used for defining a diverse set of access control policies [San96, Osb00, Fer93, Fer01b], and its suitability for Web applications [Jos01]. Another advantage of the RBAC model is that it simplifies authorization administration. An XML-based approach to specify enterprise RBAC policies has been reported in [Vuo01]. A reference implementation has been provided to address access control needs of corporate intranets in [Fer99]. In both these approaches, there is no credential-based or another mechanism provided to allow the assignment of authenticated users to a particular role. Any user can select any available role as long as the consistency constraints are satisfied. A closely related approach is presented in the OASIS XACML [URLk] specification. It is based on an extension of XML to define an access control specification that supports notions similar to those of user credentials and context-based privilege assignments. It, however, does not directly support the notion of roles, and hence lacks the essential features as separation of duty constraints, role hierarchy and cardinality. The absence of roles also prohibits the provision of a comprehensive mechanism to supply and evaluate sophisticated constraints on assignments of users to privileged tasks, since direct user-permission assignments reduces

scalability and complicates manageability of the resulting scheme. To the best of our knowledge, no earlier effort within XML and RBAC framework has been reported to target the aforementioned specific security requirements of Web services.

3. X-RBAC Specification Language

The specification language for X-RBAC aims at modeling the basic RBAC elements and their associated set-relations. To represent the RBAC elements in XML, schemas definitions are generated for "user", "role", and "permission". A set of instances for these definitions is shown in Figures 1 and 2. We reproduce below, for completeness, the salient features of the policy specification.

User Credentials: To evaluate the users being assigned to a particular role, X-RBAC uses the notion of credentials as discussed in [Ber99]. A "credential type" enforces a common set of attributes for all users belonging to that type. This set of attributes constitute the "cred_expr" for the given credential type. A credential type definition schema (XCredTypeDef) is supplied as part of the specification language to facilitate the creation of new credential types. In Figure 1(a), the users

```
<XUS>
 <user user_id="john">
   <user_name>John</user_name>
   <cred_type cred_type_id="C100">
     <type_name>Nurse</type_name>
     <cred_expr>
        <age>30</age>
        <field>opthalmology</field>
        <level>5</level>
        <status>single</status>
     </cred_expr>
   </cred_type>
   <max_roles>2</max_roles>
  </user>
  <user user_id="nancy">
   <user_name>Nancy</user_name>
   <cred_type cred_type_id="C200">
     <type_name>Eye_Doctor</type_name>
     <cred_expr>
        <age>34</age>
        <field>opthalmology</field>
        <level>8</level>
        <grade>A</grade>
     </cred_expr>
   </cred_type>
   <max_roles>1</max_roles>
  </user>
</XUS>                          (a)
```

```
<XRS>
 <roles>
  <role role_id="R100">
   <role_name>Nurse</role_name>
    <senior>Eye_Doctor</senior>
    <cardinality>8</cardinality>
  </role>
  <role role_id="R200">
    <role_name>Eye_Doctor</role_name>
    <DSD_Role_Set_id>DSD1</DSD_Role_Set_id>
    <junior>Nurse</junior>
    <senior>Eye_Surgeon</senior>
    <cardinality>6</cardinality>
  </role>
  <role role_id="R300">
    <role_name>Eye_Surgeon</role_name>
    <DSD_Role_Set_id>DSD1</DSD_Role_Set_id>
    <junior>Eye_Doctor</junior>
    <cardinality>2</cardinality>
  </role>
 </roles>
 <DSD_Role_Sets>
 <DSD_Role_Set DSD_Role_Set_id="DSD1"
DSD_cardinality="1">
    <DSD_Role>Eye_Doctor</DSD_Role>
    <DSD_Role>Eye_Surgeon</DSD_Role>
  </DSD_Role_Set>
 </DSD_Role_Sets>
</XRS>                                  (b)
```

Figure 1: XML instances of (a) XUS (b) XRS

"john" and "nancy" have credential types "Nurse" and "Eye_Doctor" respectively. We refer to this document as XML User Sheet (XUS). A sample XCredTypeDef instance is provided in Figure 2(b). The user credentials are dynamically updateable to support consistent authorization decisions.

Roles: A role has an associated set of credentials that must be satisfied by the users who are assigned to that role. Each role has an optional "SSD_Role_Set_id" and "DSD_Role_Set_id" tag to indicate any separation of duty constraints. The RBAC model uses the notion of Static Separation of Duty (SSD) and Dynamic Separation of Duty (DSD) constraints on roles. The semantics of static separation of duty require that no `n` roles that are part of a "Static Separation of Duty Role Set" (SSD_Role_Set) be assigned to the same user, where `n` is any positive integer. The semantics of dynamic separation of duty require that no `m` roles that are part of a "Dynamic Separation of Duty Role Set" (DSD_Role_Set) be simultaneously activated by the same user, where `m` is any positive integer. A role also has a "junior" and "senior" tag to reflect role hierarchy. Role "cardinality" is the maximum number of users that may be assigned to this role at any time. An

XML instance document describing "Nurse", "Eye_Doctor" and "Eye_Surgeon" roles, along with the SSD and/or DSD role sets, is shown in Figure 1(b). We refer to this document as XML Role Sheet (XRS).

Permissions: Permissions associate objects in the system with corresponding operations. An "object" in our framework can represent either a (i) concept cluster, (ii) schema, (iii) instance document, or (iv) document element to which permission is being assigned. A "concept cluster" is a group of XML schemas related to a specific concept. Exercising access control at conceptual level is very desirable in huge data repositories, such as digital libraries. A more treatment of concept clusters is given in [Bha03]. The available "operations" are "read", "write", "navigate", and "all" (which includes all three). An XML instance document of permission specifications is shown in Figure 2 (a). We refer to this document as XML Permission Sheet (XPS).

User-role & permission–role assignments: The information about users, roles and permissions available from the corresponding XML sheets are used to specify the access control policy for the protected documents. The documents generated in this phase include an XML User-to-Role Assignment Sheet (XURAS) and an XML Permission-to-Role

```
<XPS>
 <permission perm_id="P1">
  <object_type>Schema</object_type>
  <object_id>XS101</object_id>
  <operation>all</operation>
 </permission>
 <permission perm_id="P2">
  <object_type>Instance</object_type>
  <object_id>XI100</object_id>
  <operation>all</operation>
 </permission>
 <permission perm_id="P3">
  <object_type>Element</object_type>
  <object_id>/EyeCareMedicalHistory/
            Patient/ Name</object_id>
  <operation>navigate</operation>
 </permission>
</XPS>
```

(a)

```
<XCredTypeDef>
 <credential_type cred_type_id="C100">
  <type_name>Nurse</type_name>
  <attribute_list>
   <attribute_nametype="integer">age</attribute_name>
   <attribute_name type="string">field</attribute_name>
   <attribute_name type="integer">level</attribute_name>
   <attribute_name type="string">status</attribute_name>
  </attribute_list>
 </credential_type>
 <credential_type cred_type_id="C200">
  <type_name>Eye_Doctor</type_name>
  <attribute_list>
   <attribute_name type="integer">age</attribute_name>
   <attribute_name type="string">field</attribute_name>
   <attribute_name type="string">level</attribute_name>
   <attribute_name type="string">grade</attribute_name>
  </attribute_list>
 </credential_type>
</XCredTypeDef>
```

(b)

Figure 2: XML instances of (a) XPS (b) XCredTypeDef

Assignment Sheet (XPRAS). These assignments are captured through XML schemas, the instances of which are shown in Figure 3. Keeping the user, role, and permission specifications separate from their assignments allows independent design and administration of the policy, and hence supports a modular implementation of the X-RBAC system.

In the XURAS instance of Figure 3(a), the "logical_expr" is used to evaluate the "predicate" for the "credential_condition". This example associates a set of credentials to the "Eye_Doctor" role. It states that all users with the "Nurse" credential type can be assigned to the "Eye_Doctor" role only if "field" is "opthalmology" and, either "level" is greater than 4, or "age" is less than 80. Note that a user with user_id = "any" is recognized by the system as an unknown user, who may be required to supply additional credential conditions in order to be assigned to a particular role. If no explicit conditions are specified, then any user could be assigned the particular role, which usually is the "guest" role in most Web services.

In the XPRAS instance of Figure 3(b), the assignment identified by "PRA2" associates the "Eye_Doctor" role with the permissions "P1" and "P2", which refer to a schema object and an instance document, respectively (see Figure 2(a)). In this case, an "Eye_Doctor" role is authorized to "read/write/navigate" all instance documents conforming to the schema identified by XML schema id "XS101", and also the instance document identified by XML instance id "XI100". Similarly, the assignment identified by "PRA1" associates the "Nurse" role with the permission "P3", which refers to the "Name" element identified by an XPath expression (see Figure 2(a)). Hence, the "Nurse" role is authorized only to "navigate" the "Name" element in all documents conforming to the structure imposed by the given XPath expression.

```
<XURAS>
 <ura ura_id="URA1">
  <role_name>Eye_Doctor</role_name>
  <users>
   <user user_id="john">
    <cred_conditions>
     <cred_condition>
      <cred_type>Nurse</cred_type>
      <logical_expr op="AND">
       <predicate>
         <operator>eq</operator>
         <name_param>field</name_param>
         <value_param>ophthalmology
         </value_param>
       </predicate>
       <predicate>
         <logical_expr op="OR">
          <predicate>
            <operator>lt</operator>
            <name_param>age</name_param>
            <value_param>80</value_param>
          </predicate>
          <predicate>
            <operator>gt</operator>
            <name_param>level</name_param>
            <value_param>4</value_param>
          </predicate>
         </logical_expr>
       </predicate>
      </logical_expr>
     </cred_condition>
    </cred_conditions>
   </user>
  </users>
 </ura>
</XURAS>
```

Figure 3(a): XML instance of XURAS

```
<XPRAS>
 <pra pra_id="PRA1">
    <role_name>Nurse</role_name>
    <permissions>
     <perm_id>P3</perm_id>
    </permissions>
 </pra>
 <pra pra_id="PRA2">
    <role_name>Eye_Doctor</role_name>
    <permissions>
     <perm_id>P1</perm_id>
     <perm_id>P2</perm_id>
    </permissions>
 </pra>
</XPRAS>                          (b)
```

Figure 3(b): XML instance of XPRAS

4. X-RBAC System Architecture

In this section, we present the system architecture of X-RBAC. We first provide an overview of the system components and technologies, and then elaborate upon it with the help of the XML instances presented in section 3 to illustrate the process of specification and enforcement of the said policy. We also outline some limitations of the current implementation, and certain desirable features that should be incorporated in future.

4.1 Overview

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

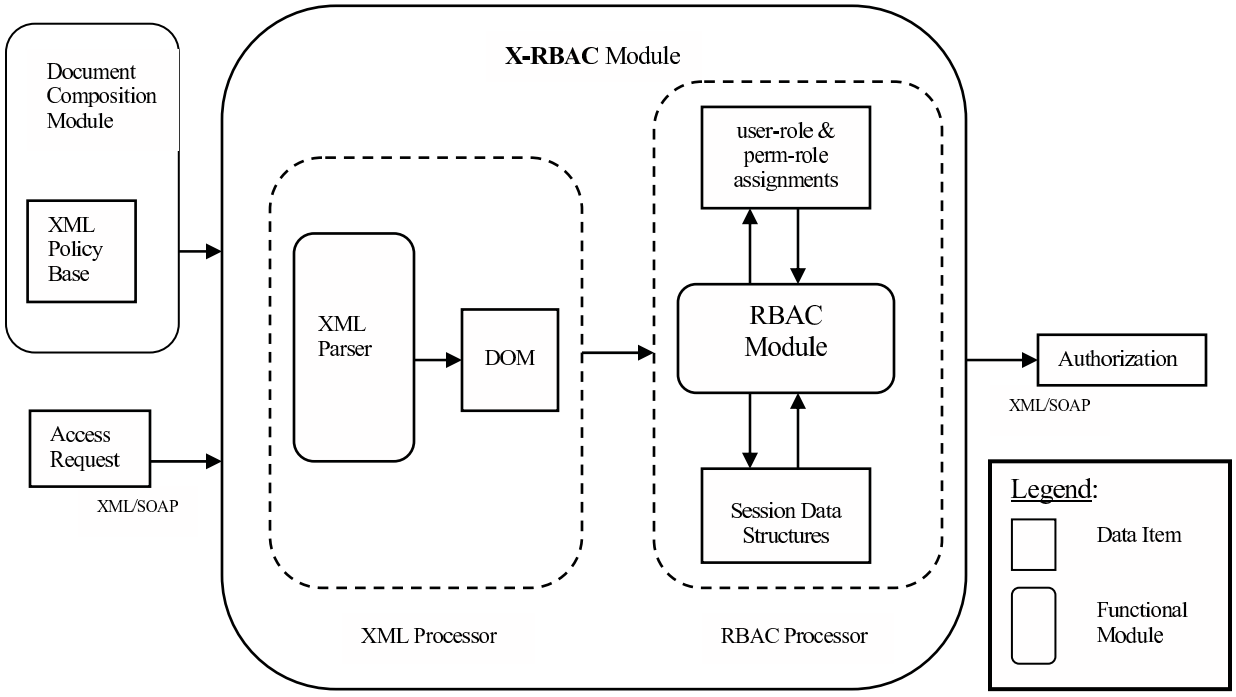

Figure 4: X-RBAC System Architecture

The overall system design is depicted in Figure 4. As indicated in the figure, the two main sub-systems of X-RBAC Module are the XML Processor and the RBAC Processor. The XML processor is implemented in Java using Java API for XML Processing (JAXP). Custom modules have been designed to get the DOM instance of parsed XML documents and forward them on to the RBAC Processor. The RBAC Module then administers and enforces the policy according to the supplied policy information.

The policy information is contained in the XML Policy Base. A document composition module external to X-RBAC is provided to compose the policy documents. In addition to composing the five primary policy sheets, this module allows the system administrator to create a credential type definition sheet, XCredTypeDef, and hence specify the credential types for a given policy. All these XML sheets together constitute the XML Policy Base. The policy sheets from the XML Policy Base are loaded into the X-RBAC Module by the system administrator. Since X-RBAC can act as both stand-alone and web-deployable application, it may be invoked from either the local system, or remotely through an XML-aware browser. Hence, the X-RBAC Module seamlessly interfaces with an external client. The client may submit an access request through any standard XML-based Web services messaging protocol, like SOAP [URLa]. Similarly, the access authorization is returned via the same protocol.

4.2 XML Processor

The XML Processor contains the XML Parser and the DOM tree representations of the supplied XML documents. The X-RBAC system provides a Policy Loader to load the policy sheets for a given policy. As a next step, functionality is provided to validate the policy sheets in terms of existence checking and type conformance. This means that all users, roles, and permissions referenced in XURAS and XPRAS must exist in the corresponding XUS, XRS and XPS respectively. Also, the credential types associated with the users must conform to the type definitions in the XCredTypeDef sheet. In the context of the XML instances displayed in Figures 1-3, it checks for the definition of credential types “Nurse” and “Eye_Doctor”, the user ids “john” and “nancy”, and the role names “Nurse”, “Eye_Doctor”, and “Eye_Surgeon”. This validation support is provided by Apache Xalan XSLT engine built into JAXP. Once the policy sheets are validated, the corresponding DOM tree representation is generated and passed on to the RBAC Processor. A facility is provided to display the instance of the DOM tree via the X-RBAC GUI. A snapshot of the DOM tree display of a loaded XUS in X-RBAC GUI is shown in Figure 5(a).

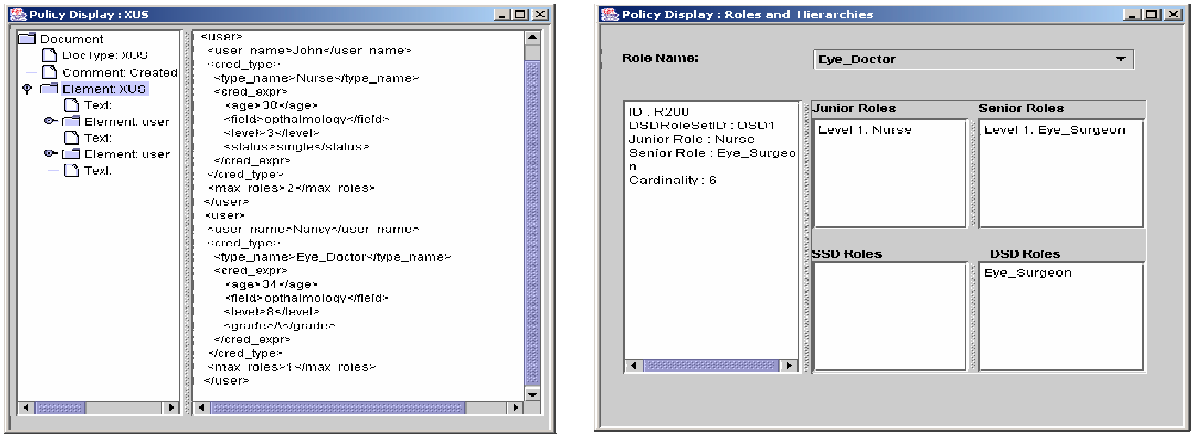

Figure 5: Snapshots of Policy Display (a) XUS (b) Roles and Hierarchies Information

4.3 RBAC Processor

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

Policy Administration: The RBAC Module provides functionality to parse the DOM tree structures supplied by the XML Processor, and retrieves the relevant information into its internal data structures. The policy assignments are checked against the RBAC consistency rules, similar to those outlined in [Gav98], against violations of any SSD, DSD, or cardinality constraints. In the context of the XML instances of Figures 1-3, this means that user "john" will be assigned by the RBAC Module to the "Eye_Doctor" role because it satisfies all the required credential and consistency conditions. The permissions in the system are also assigned to roles under similar consistency notions. It may be noted that the actual role activation for "john" would occur when he actually logs into the system and requests a role. The notion of role assignment in the context of "john" is of static type, i.e. it implies that "john" has been declared as assignable to the said role based on already supplied credential information. There can also be a dynamic role assignment for an unknown user based on his/her credentials supplied at the time of login. These static and dynamic policy assignments create the complete internal representation of the XML Policy Base within the RBAC Processor for enforcement of the policy. A facility is provided to display the policy information items via the X-RBAC GUI. A snap shot of the "roles and hierarchies" information as extracted from RBAC Module data structures is shown in Figure 5(b).

Policy Enforcement: The information from the internal data structures is then used by the RBAC Module to enforce the policy and manage user sessions. The initial login into the system will create a default session for the user with a pre-specified "minimal" set of roles activated based on the supplied user credentials. The initial login can be the "user_id" from the XUS, if it is a known user, or a "user_id" of "any", as discussed in section 3. In addition to the default set of activated roles, more roles can also be activated if the user credentials so allow. Access to resources is requested in the form of an XML Access Request (XAR) that specifies the "object type" and "object id" of the requested resource. An XAR could be submitted locally or remotely as an assertion in SOAP or similar XML-based messaging protocol. This access request is then evaluated based on the currently activated roles for this user. Only those resources may be accessed during a session for which the activated set of roles has associated permissions. Both the login information and XARs for a user are stored in an XML Access Sheet (XAS). A log of the sessions is maintained and the user credentials are dynamically updated to capture the activity profile of the user for future access decisions.

4.4 Limitations and Improvements

The current implementation of X-RBAC is in the form of a Java-based application. We have also experimented with transforming it to an applet (or servlet) that can be made to run in a local web browser. Integration of X-RBAC Module in

web browsers could pose some known technical issues in terms of file access as governed by Java security APIs. A documented approach to handle this is to add the to-be-accessed files to the ACLs of the browser serving the resources. The web browser running X-RBAC can then evaluate and fulfill the incoming access requests of each user in each session independently, according to its currently activated set of roles. Additionally, efforts are underway to standardize the composition of the XML Policy Base by adopting WS-Policy specifications. Such provisions would allow an X-RBAC module implemented as an applet to be readily deployed as a Web Service and enforce domain-specific access control policies.

We are also contemplating a few improvements in system design. More transparent form of validation mechanisms needs to be researched for validating the policy sheets. We are keenly looking into the possibility of providing validation support through the use of XML schema, instead of the currently adopted XSLT/XPath model, and this will be further investigated in future. It is also being considered to provide a facility to generate the schemas for each credential type defined in the XCredTypeDef sheet, so that the credential types can be validated through the use of XML schema references, rather than through program code.

5. Conclusions

In this paper, we have presented X-RBAC, an XML-based RBAC policy specification framework for enforcing access control in dynamic XML-based Web services. The X-RBAC implementation is a Java-based application, and is based on a policy specification language that addresses the security challenges specific to these Web services. The specification language adopted provides compact representation of access control policies, and conforms to the NIST RBAC model. Our framework can be used to specify and enforce RBAC policies for securing XML documents at conceptual, schema, instance as well as element levels, and allows dynamically capturing context information. The software architecture presented in the paper separates the XML and RBAC components of the system into distinct modules, which allows independent design and administration of the policy. Modifications and extension to our X-RBAC system in several directions are being considered. We plan to add support for a multi-domain environment where policy authorizations may not be centrally located, but are distributed across several domains. We also plan to allow specification of the elaborate set of temporal constraints introduced in our Generalized Temporal Access Control Model [Jos02].

References

[Ber99] E. Bertino, S. Castano, E. Ferrari, M. Mesiti, "Controlled Access and Dissemination of XML Documents", Workshop On Web Information And Data Management, November 1999.

[Ber01a] E. Bertino, S. Castano, E. Ferrari, "Securing XML Documents with Author X", IEEE Internet Computing May-June 2001.

[Ber01b] E. Bertino, S. Castano, E. Ferrari, "On Specifying Security Policies for Web Documents with an XML-based Language", Proceedings of the Sixth ACM Symposium on Access control models and technologies, 2001.

[Bha03] R. Bhatti, J. B. D. Joshi, E. Bertino, A. Ghafoor, "XML-based RBAC Policy Specification for Web-Services", *Submitted to IEEE Computer*.

[Dam02] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, P. Samarati, "A Fine Grained Access Control System for XML Documents", ACM Transactions on Information and System Security Vol. 5 No. 2, May 2002.

[Fer93] D. F. Ferraiolo, D. M. Gilbert, and N. Lynch, "An Examination of Federal and Commercial Access Control Policy Needs," In Proceedings of NISTNCSC National Computer Security Conference, Baltimore, MD, September 20-23 1993, pp. 107-116.

[Fer99] D. F. Ferraiolo, J. F. Barkley, D. R. Kuhn, "A Role Based Access Control Model and Reference Implementation Within a Corporate Intranet", ACM Transactions on Information and System Security Vol. 2 No. 1, Feb 1999.

[Fer01a] David F. Ferraiolo , Ravi Sandhu , Serban Gavrila , D. Richard Kuhn , Ramaswamy Chandramouli, "Proposed NIST standard for role-based access control", ACM Transactions on Information and System Security (TISSEC), Volume 4 , Issue 3 (August 2001).

[Fer01b] D. Ferraiolo, R. Sandhu, S. Gavrila, R. Kuhn, R. Chandramouli, "The NIST Model for Role-Based Access Control: Towards a Unified Standard," ACM Transactions on Information and System Security, Vol 4, Issue 3, August 2001, pp. 224-274.

[Gav98] S. I. Gavrila , J. F. Barkley, "Formal Specification for Role Based Access Control User/role and Role/role Relationship Management",Proceedings of the third ACM workshop on Role-based access control, Fairfax, Virginia, United States, October 22-23, 1998, pages81-90.

[Had00] S. Hada, M. Kudo, "XML Access Control Language: Provisional Authorization for XML Documents", October 16 2000, Tokyo Research Laboratory, IBM Research.

[Jos01] J. B. D. Joshi, W. G. Aref, A. Ghafoor and E. H. Spafford, "Security Models for Web-based Applications", *Communications of the ACM*, 44, 2 (Feb. 2001), pages 38-72.

[Jos02] J. B. D. Joshi, Elisa Bertino, Usman Latif, Arif Ghafoor, "Generalized Temporal Role Based Access Control Model (GTRBAC) (Part I) - Specification and Modeling", Submitted to IEEE Transaction on Knowledge and Data Engineering.

[Osb00] S. L. Osborn, R. Sandhu, Q. Munawer, "Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies," *ACM Transactions on Information and System Security*, Vol. 3, No. 2, February 2000, pp. 85-106.

[San96] R. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman, "Role Based Access Control Models", IEEE Computer Vol. 29, No 2, February 1996.

[Vuo01] N. N. Vuong, G. S. Smith, Y. Deng, "Managing Security Policies in a Distributed Environment Using eXtensible Markup Language (XML)", Symposium on Applied Computing, March 2001

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

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

Appendix – A
XML Schemas for RBAC elements

(i) User Credential

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

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

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

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

  <xs:/schema>
```

(ii) Credential Type Definition

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

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

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

 <xs:/schema>
```

(iii) Role Definition

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

```
        <xs:element name = “DSD_Role_Set_id” type=” xs:string” minOccurs=”0”/>
       <xs:element name = “senior” type=” xs:string” minOccurs=”0”/>
       <xs:element name = “junior” type=” xs:string” minOccurs=”0”/>
       <xs:element name = “cardinality” type=” xs:integer” minOccurs=”0”/>
     </xs:/sequence>
    <xs:/complexType>
  <xs:/element>
 </xs:complexType>
<xs: /element>

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

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

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

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

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

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

<xs:schema>
```

(iv) **Separation of Duty Definitions**

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

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

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

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

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

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

<xs:schema>
```

(v) **Permission Definition**

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

```
                    <xs:/complexType>
                  <xs:/element>
                  <xs:element name = “operation” >
                    <xs:simpleType>
                     <xs:restriction base="xs:string">
                       <xs:enumeration value="read"/>
                       <xs:enumeration value="write"/>
                       <xs:enumeration value="navigate"/>
                       <xs:enumeration value=”all"/>                (read/write/navigate)
                      <xs:/restriction>
                     <xs:/simpleType>
                    <xs:/element name>
                    <xs:element name = “prop” minOccurs=”0”/>
                      <xs:simpleType>
                       <xs:restriction base="xs:string">
                        <xs:enumeration value="no_prop"/>
                        <xs:enumeration value="first_level"/>          (includes direct children)
                        <xs:enumeration value="cascade"/>             (includes all descendants)
                       </xs:restriction>
                      </xs:simpleType>
                     <xs:/element>
                  <xs:/sequence>
                 <xs:/complexType>
               <xs:/element>
    <xs:/complexType>
  <xs:/element>

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

  <xs:/schema>
```

Appendix – B
XML Schemas for Policy Administration Documents

(i) User to Role Assignment

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

```
                <xs:/complexType>
              <xs:/element>
            <xs:/sequence>
          <xs:/complexType>
        <xs:/element>
      <xs:/complexType>
    <xs:/element>
 <xs:schema>

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

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

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

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

```
                <xs:element name = “logical_expr” type=”logical_expr_type” />
                <xs:element name = “predicate” type=”predicate_type” />
            <xs:/choice>
          <xs:/complexType>
      <xs:/element>
     <xs:/sequence>
    <xs:/complexType>

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

 <xs:/schema>
```

(ii) **Permission to Role Assignment**

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

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

              <xs:/schema>
```

(iii) **XML Access Sheet**

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

  <xs:element name = “xar”  maxOccurs=”unbounded”>
     <xs:complexType>
```

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

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

(iii) **XML Sessions Sheet**

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

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

<xs:/schema>
```

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