

A FLEXIBLE AUTHENTICATION AND AUTHORIZATION SCHEME FOR A LEARNER INFORMATION MANAGEMENT WEB SERVICE

C. STEVEN CUMMINGS, HONGCHI SHI* and YI SHANG

*Department of Computer Science, University of Missouri-Columbia
Columbia, MO 65211, USA*

**shih@missouri.edu*

SU-SHING CHEN

*Department of Computer and Information Science and Engineering
University of Florida, Gainesville, FL 32611, USA*

As many software systems migrate their component communication to the web service paradigm, security becomes an immediate concern. Many existing solutions cannot handle complex situations involving the user, session, application, and any number of XML inter-related documents. The learner information management web service we develop for an e-learning environment has to control access of inter-related XML documents representing different roles of users in the e-learning environment. In this paper, we present an authentication and authorization model for web services that provides access control of inter-related XML documents. This scheme works especially well when the documents that will be operated on are organized in hierarchical structures, like the collections in a native XML database.

Keywords: Extensible markup language; web service; access control mechanism; role-based access control; learner information management.

1. Introduction

Many software solutions are migrating to the web service paradigm to achieve inter-operability between the components that provide platform and programming language independence. One of the main concerns accompanying this trend is that of security. As software entities communicate with one another, how does a web service ensure that the requestor is authenticated for using the system or sub-system? How can any operation of the web service or the data and information that is being operated on be authorized based on the privileges of the requestor? In this paper, we propose a new technique to authenticate and authorize web service requests that offers flexibility and robustness. We present the technique as it is applied to a learner information management web service in an e-learning environment.

1.1. *Learner information management in an e-learning environment*

Learner information is essential in an advanced e-learning environment that aims to improve the learning experience of students. While the learner is working through course materials and conducting assessment (via quizzes and projects), an advanced e-learning environment builds up a model of the student and stores it back to the learner information profile. This allows the environment to better adapt learning materials to individual learners and their learning styles. The environment users comprise the learners, teachers, parents, and system administrators.

LIMS2 (Learner Information Management System, version 2) is an updated version of LIMS presented previously in a conference paper,¹¹ and it is an important component in IDEAL (an Intelligent Distributed Environment for Asynchronous Learning).¹² In this paper, we discuss our access control mechanism in the context of providing secure access for various users to the learner information managed by LIMS2.

In the learner information management system, the security must be provided with facilitation for different roles that users of e-learning take on. That is why LIMS2 is role-based. For example, a security implementation should prevent a learner from modifying his own performance results or student model even though that information is stored with the learner's profile. This seems obvious from the fact that the environment that provides the user-interface should just be coded to this logic, but LIMS2 is a web service, and so all users can potentially code against it themselves using their login names and passwords. In such a case, the content returned needs to be controlled and filtered appropriately for reads and updates at the service level, before data and results reach the interface.

1.2. *Using a native XML database for storing learner information*

The learner information managed by LIMS2 can be considered as a collection of documents with each document representing a learner's information. Considering what type of database to store the learner information, there is a trend of migrating from relational database systems to native XML databases. A significant amount of code in complex web applications is based on the serialization of objects to a persistent format. Oftentimes this format is the relational model, and each object and sub-object must be mapped into the strictly-typed tuple-format required of RDBMS's. Additionally, the application must specifically know how to utilize each field. Most applications cannot dynamically use relational data. There is a general acknowledgement that not all application entities require this format, and a new alternative is semantic, self-describing data encoded with XML. Native XML databases^{18,19} provide weaker constraints on data operations, and complex object-entities can be described more naturally. Native XML databases actually store their content in raw XML form, or some compressed form. Many RDBMS's are now providing adapters to store XML content represented in relational tables, but this

is not native XML storage and is subject to performance penalties. The general database operations are also more natural with respect to semantic data and XML. XML Documents do not have to contain a set of tuples of any specific format, nor do they even have to follow any specific content structure schema. They can have any arbitrary structure composed of arbitrarily specified elements that describe the entities. It is up to the application utilizing the XML database to enforce such schemas, which help to give meaning to the raw markup. The application can do this independently of the XML database, at a layer utilizing its resources.

XML documents are stored by key in collections, which are hierarchically arranged. An entire collection can be queried with XPath¹⁴ expressions and the results returned will include matching resources from all documents in the collection. The contents of the database are stored in a hierarchical fashion outside of the documents as well, which lends itself better to semi-structured content. Overall, native XML databases provide a better information repository for web service code.

The native XML database in LIMS2 stores XML documents for users of different roles such as learners, teachers, and parents. The XML documents are semantically inter-related. For example, when a teacher teaches a course a student takes, the XML document for the information about the teacher is semantically related to the XML document for the information about the student in LIMS2.

1.3. *Related work*

The core of our technique is effectively and efficiently authenticating and authorizing access requests for information managed by a web service. A few approaches for XML access control have been proposed in the literature.^{6-8,10,13} They only address control of access to parts of single XML documents. Our approach provides a flexible model for web service authentication and authorization that can be easily described in XML¹⁶ and can handle access control of inter-related XML documents in most web service infrastructures.

One related project is *Author-X* that provides two major features — the temporal description of constraints and the broadcasting of documents maintaining compliance with the access control model.^{6,7} Some ideas of the *Author-X* project are more efficiently implemented in a scalable XML access control system.¹³ With *Author-X*, users can subscribe to certain XML documents from the document repository, and each subscription comes with an associated policy. Different users might possibly be allowed to view different portions of a document in question. Additionally, each policy is encrypted with a separate private key for the user accessing the document portion. The policies are specified by indicating a privilege (**view**, **navigate**, or **browse_all** for some document with an additional pair of XPath statements to specify the portion of the document viewable and specify a condition on the user's credentials. The temporal interval is given with a "periodic impression" stating what days or times within the interval that the policy is granted. Finally, a cascade option is specified, which indicates whether or not sub-elements

are allowed the same as the explicitly specified elements from the document. The policy language used is called *X-Sec*. The one feature that this system does not provide which LIMS2 does is the specifying of conditions that relate semantically different documents that might be accessed by the user.

Another approach for XML access control focuses on a provisional authorization model and an XML access control language, XACL.¹⁰ Actions such as log, verify, encrypt, transform, write, create, and delete are allowed or prohibited to proceed according to the conditions specified in the provisional authorization model. The control mechanism is useful for Web-based XML data access control. However, it does not take into account any relationship between XML documents in a collection. Our learner information management web service requires control of access to inter-related XML documents.

Yet another XML access control approach is making the SOAP request itself the object of authorization.⁸ This makes authorization for web services simpler and not dependant on transport details like firewalls. Fine-grained control of the service is also maintained at the message level. Such constraints can be achieved as specifying required parameters, required parameter values, and explicitly including fragments of the request document payload in a security expression or condition. Users can be identified by a user-id, role-id, and IP-address or domain. The subject can be specified in any one of these forms, and the value can be passed via custom SOAP headers. The paper presents a specific access-control structure to be used for content in the SOAP header of a request. Access policy specifications consist of the required subject, or possible subjects specified by user, role, or location, and the object (a specific fragment of the SOAP request). This method does not allow for much control beyond the immediate SOAP request. The SOAP request documents must contain all necessary information about the operation, object, subject, and possible side-effects. Our approach avoids having to rely so much on the SOAP-request documents.

In the following sections, we discuss our authentication and authorization scheme, which is partly implemented on our LIMS2 web service that manages learner information in an e-learning environment.

2. LIMS2 and Access Control

The technique that we present in this paper is used for authenticating and authorizing access to LIMS2 (Learner Information Management System, version 2), a major component of a distributed e-learning system called IDEAL.¹² IDEAL provides adapted learning-objects to the user and constantly passes messages, calling services on LIMS2. When the environment and other components are communicating with LIMS2, it is crucial that security be maintained. Therefore, LIMS2 implements a specific kind of access control policy that is role-based and can be specialized by the context of the particular user. LIMS2 provides a SOAP-based service that maintains authenticated sessions and manages the authorization of

requested operations. The authorization process uses the role and learner profiles and semantic relationships between these entities and involved XML documents to determine what actions are allowed for the caller.

The schema for the learner profiles complies with the IMS LIP (Learner Information Packaging) specification.⁹ The IMS LIP standard enables the inclusion of mechanisms for privacy. The LIP standard is meant to describe learner information that can be exchanged with other e-learning systems. The learner profiles in LIMS2 are based on this specification explicitly. Below, we will demonstrate the specific schema we use and how the IMS-LIP-compliant content is used to specify privileges on document content.

LIMS2 provides a specific implementation of access control for the learner information and XML data provided through the web service. Authentication is based on learner profiles in a special collection of the XML database. These learner-information profiles contain three pieces of relevant information, username, password, and user-role. Special “handlers” monitor the web service requests for authentication and authorization purposes. The first handler is the authentication handler. If a particular user has not yet been authenticated, then the handler checks inside the SOAP headers for the username and password. If these are not provided, then the request is rejected. Otherwise, the handler tries to find a learner profile containing the username and password. If one is found, the handler is passed for authentication, otherwise the request is rejected. Rejections are implemented with SOAP faults. The service endpoint itself can maintain a session with clients, so the authentication process should only have to occur on the first access and not again until a later time when the session has expired.

Next is the authorization handler. If the user has not been authorized before, the handler will try to build up the user’s access profile from the role-specific policy. If the user has been authorized before, then the profile exists. In a special XML database collection called “roles”, all of the role-specific policies exist. The following XML schema fragment shows the top-level of the access control policy. In the schema code, the prefix `xsd` is for the namespace URI `http://www.w3.org/2001/XMLSchema`.

```
<xsd:element name="role" type="roleType"/>
<xsd:element name="accessrights">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="role" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Any number of `role` elements can exist under `accessrights`. Additionally, these roles can be kept in separate files in the XML database collection. This detail really does not matter, as the authorization system will find them anywhere in

the collection. The following fragments show how the `role` element breaks down into the actions that members of the role can perform and the elements that those operations are allowed on.

```

<xsd:element name="operations" type="operationsType"/>
<xsd:complexType name="roleType">
  <xsd:sequence>
    <xsd:element ref="operations"/>
  </xsd:sequence>
  <xsd:attribute name="roletype" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:element name="operation" type="operationType"/>
<xsd:complexType name="operationsType">
  <xsd:sequence>
    <xsd:element ref="operation" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:element name="allowedelement" type="allowedelementType"/>
<xsd:complexType name="operationType">
  <xsd:sequence>
    <xsd:element ref="allowedelement" minOccurs="0"
      maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="operationtype" type="xsd:string"
    use="required"/>
</xsd:complexType>

<xsd:element name="elementname" type="xsd:string"/>
<xsd:element name="conditions" type="conditionsType"/>
<xsd:complexType name="allowedelementType">
  <xsd:sequence>
    <xsd:element ref="elementname"/>
    <xsd:element ref="conditions" minOccurs="0"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

```

In the complex type `operationType`, the attribute `operationtype` typically takes on one of the four values: `read`, `write`, `create`, or `delete`. However, this attribute is left open to custom operations that might be described and implemented, for instance, the `replace` operation.

When a service request arrives, the authorization handler determines what type of access is needed based on the *service method*. For instance, a `getDocument` operation requires read access of all contained elements, and an `updateDocument` call requires write access of all elements contained in the document plus create access of all elements in the replacement document. So, all operations are specifically coded in the handler to check for the appropriate access.

The handler checks the access policy as described in the schema fragment above for the needed operation and element names. A note should be made about the difference here from LIMS to the current LIMS2. LIMS required that all elements that would be operated upon in any manner be listed in the access control policy. If the element was not mentioned, then the operation was not permitted. LIMS2 relaxes the scheme and allows unlisted elements to be operated on. The thinking is that one cannot possibly list all of the possible elements that the user might need to view within the application. This is especially true if the information repository is going to have dynamic exchange or management of information. Therefore, the access policy should focus on setting up “choke points” of elements to disallow always or in specific situations (see schema segments for specifying conditions below). The other aspect of element permissions is that once an element has been disallowed, the entire sub-tree rooted at it (including the element and all of its children) is excluded from permissible operation-space. Therefore, only the element at the root of the sub-tree in question needs to be specified for filtering, and we can assume the remaining elements above it are allowed. Typically, the allowed content is much larger and less well-defined than the content that will be restricted. This makes it harder to explicitly allow certain elements ahead of time. Not only is restricted content usually more well-defined, but it should usually be precisely defined as it is known to be restricted from particular users and roles.

For each element, there can be condition sets (established with the `conditions` element) that have one or more `condition` elements. An element listed with no condition sets at all is automatically allowed for the given operation. The conditions are allowed to fall under multiple condition sets so that optional scenarios can be supported. Within a condition set, at least one condition must be true, i.e. the conditions in a condition set are combined by a logic OR operation. Overall, all of the condition sets must evaluate to true to allow the given operation to be performed. The following schema segments show how to specify conditions.

```
<xsd:element name="condition" type="conditionType"/>
<xsd:complexType name="conditionsType">
  <xsd:sequence>
    <xsd:element ref="condition" minOccurs="0"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="source_element" type="source_elementType"/>
```

```

<xsd:element name="destination_element"
  type="destination_elementType"/>
<xsd:complexType name="conditionType">
  <xsd:sequence>
    <xsd:element ref="source_element" minOccurs="0"/>
    <xsd:element ref="destination_element"/>
  </xsd:sequence>
  <xsd:attribute name="type" default="prohibit">
    <xsd:simpleType>
      <xsd:restriction base="xsd:NMTOKEN">
        <xsd:enumeration value="prohibit"/>
        <xsd:enumeration value="exist"/>
        <xsd:enumeration value="equal"/>
        <xsd:enumeration value="notequal"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>

```

There are four types of conditions that can be specified for possible allowed-elements. The first is **prohibit**. This condition type specifies that from the current context, the expression given as **destination** cannot exist, or the current element is not allowed. The **exist** condition type allows specifications where the expression must exist for the current element-context, or that element is not allowed. Finally, the simplest conditions, **equal** and **notequal**, test to see whether the source and destination values are equal or not equal respectively. These values can be specified as document- or query-relative. The details on how to specify this are given in the schema fragment below.

```

<xsd:complexType name="destination_elementType">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="type" use="required">
        <xsd:simpleType>
          <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="QueryRelative"/>
            <xsd:enumeration value="DocumentRelative"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

```

2.1. Context variables

Access control policies can contain references to a few pre-defined context variables. For now, these simply include some variables from the user session. They are `username`, `userid`, `userrole`, and `password` by default. If further attributes are defined for the user context in the application, they are available to the access control expressions.

The conditions are where the power of LIMS2 comes in. In the conditions, expressions can be constructed that utilize values from the learner's profile relative to the role access policy. Because all of the documents in a collection can be queried at the same time, conditions can be specified that test content from any of the result set. For instance, suppose a query retrieves the current "model" for a student which includes performance and progress data. Obviously, we do not want the student altering (or possibly even viewing) this particular data. So a condition can be constructed that relates the current user's role and name to the data being retrieved. If we assume the learner profile in question contains the model data at the path `/learnerinformation/model`, then we can issue the following conditions to ensure that the viewer or updater is a teacher and the particular student's advisor.

```
<condition>$userrole$='teacher'</condition>
<condition>/learnerinformation/registration/advisor=$username$
</condition>
```

2.2. Further capabilities and techniques

It would also be beneficial to explicitly indicate a variable context that a piece of information is coming from. The newer expression prefix follows the pattern set by the `document()` function from XSLT. The prefixes specify variable contexts that are implemented and maintained by the web service. The suggested variable context prefixes are `request()`, `session()`, and `application()`. Our authentication and authorization scheme is considered in the context of an application that provides an XML document repository somehow through a web service. Therefore, it is beneficial to implement the `document()` function itself from XSLT to allow the indication of the XML database collection in which the document resides. However, it should also the specifcaiton of full URIs, as originally implemented in XSLT. The following examples demonstrate some of the more advanced conditions that are possible using these extensions.

```
<!-- User is administrator -->
<condition>session()/userrole/text()='Administrator'</condition>

<!-- Request is coming from a valid IP address -->
<condition>application()//allowed_ips[text()='request()/headers/
remotehost/text()']</condition>
```

```

<!-- User is not logged in multiple times -->
<condition>count(application()//sessions/session[username/text()
    =session()/username/text()])=1
</condition>

<!-- Get all (text) sections specified in the index for the
    current course -->
<condition>collection('/learning-objects')//section/@id=./@id
</condition>

```

2.3. Limitations

There are currently two limitations to specifying access policies in LIMS2. The first is that context variables are only provided at the level of the user session (described previously). In the future, LIMS2 will support context variables at the application level (providing global variables) and user-request level (providing request attribute, parameter, and header variables).

The second limitation is that in the allowed-elements, the name specified provides no facilities for namespace qualification. Elements are just matched based on tag names. In the future, LIMS2 will support namespace qualified elements by specifying the element name in the form `{namespace-uri}local-name`. So to specify the `table` element from XHTML 1.0, one would type `{http://www.w3.org/1999/xhtml}table`. This would allow more flexible description of outgoing content for users and roles, and it would also allow the different types of permissible data to be organized into specific schemas that contain the relevant types.

3. Web Service Interface

As mentioned previously, LIMS2 is accessed through a web-service endpoint. Any client that knows how to access a SOAP web-service can access LIMS2. Specifically, LIMS2 is implemented with Apache AXIS (currently 1.1-beta) and augmented with service handlers (described earlier). What follows is a detailed discussion of the message passing to the web-service and a discussion of methods that can be used to invoke it.

3.1. SOAP message

As specified by the SOAP 1.1 standard, messages and RPC calls are contained in an XML document called an “envelope”. The envelope can contain an optional header (used by LIMS2 to find the username and password for initial authentication and authorization profile construction) and the required body that contains the message or RPC call itself. The general form looks like the following.

```

<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/
  soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Header>
    <!-- ... named header elements ... -->
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <!-- ... message or rpc-call ... -->
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

The `xsd` and `xsi` mappings are used for RPC calls to specify the data types of arguments be passed with the method call. The returned values will also be decorated with data type specifications. Data types are specified by supplying the attribute `xsi:type` with the element containing the value. The `xsd` namespace supplies some basic types like `xsd:string`. The SOAP encoding takes the typical namespace value.

3.2. LIMS2 requests and responses

The following is a SOAP request envelope demonstrating an authentication request. No arguments are explicitly passed to the `authenticate` method because the username and password are picked up from the SOAP header (as described above for the handlers). The namespace used for the method itself shows the object that the method is being called on (the web service).

```

<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/
  soap/envelope/">
  <SOAP-ENV:Header>
    <username>joe</username>
    <password>s3cr3t</password>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <ns1:authenticate
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/
        soap/encoding/"
      xmlns:ns1="urn:XMLDataService">
    </ns1:authenticate>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

A typical response might look like the following.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/
  soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <ns1:authenticateResponse xmlns:ns1="urn:XMLDataService"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/
        soap/encoding/">
      <authenticateReturn xsi:type="xsd:boolean">true
      </authenticateReturn>
    </ns1:authenticateResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Method calls generally look the same, so only one example of a request and response will be shown. The only difference from the authentication request is that the headers are optional (the service endpoint should keep a session and recognize the user again via the POST headers or similar mechanism) and that the method calls take arguments. Here are a request and its response for requesting learner skill levels for a learning object about the Java JINI technology.

The request is as follows:

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/
  soap/envelope/">
  <SOAP-ENV:Body>
    <ns1:executeQuery SOAP-ENV:encodingStyle="http://
      schemas.xmlsoap.org/soap/encoding/"
      xmlns:xsd="http://www.w3.org/1999/XMLSchema"
      xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
      xmlns:ns1="urn:XMLDataService">
      <collectionName xsi:type="xsd:string">/users
      </collectionName>
      <xpath xsi:type="xsd:string">//unitsfield[fieldlabel/
        typename/tyvalue/text()='ctjini050101']
      </xpath>
    </ns1:executeQuery>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

and the response is the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/
  soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <ns1:executeQueryResponse xmlns:ns1="urn:XMLDataService"
      soapenv:encodingStyle="http://schemas.xmlsoap.org/
        soap/encoding/">
      <ns1:executeQueryReturn xmlns:xsi="http://www.w3.org/2001/
        XMLSchema-instance"
        xmlns:ns2="http://www.w3.org/2001/XMLSchema"
        xsi:type="ns2:string">
        <?xml version="1.0"?>
        <xapi:resourceSet xapi:collectionURI="xmldb:
          xindice:///users"
          xmlns:xapi="http://www.xmldb.org/xapi/ResourceSet">
          <xapi:resource xapi:documentID="guest">
            <unitsfield xmlns:src="http://xml.apache.org/
              xindice/Query"
              src:col="/db/users" src:key="guest">
              <fieldlabel xmlns:src="http://xml.apache.org/
                xindice/Query">
                <typename xmlns:src="http://xml.apache.org/
                  xindice/Query">
                  <tysource sourcetype="imsdefault"
                    xmlns:src="http://xml.apache.org/
                      xindice/Query" />
                  <tyvalue xmlns:src="http://xml.apache.org/
                    xindice/Query">ctjini050101</tyvalue>
                </typename>
                </fieldlabel>
                <fielddata xmlns:src="http://xml.apache.org/
                  xindice/Query">3</fielddata>
              </unitsfield>
            </xapi:resource>
            <xapi:resource xapi:documentID="liu">
              <unitsfield xmlns:src="http://xml.apache.org/
                xindice/Query"
                src:col="/db/users" src:key="liu">
```

```

        <fieldlabel xmlns:src="http://xml.apache.org/
          xindice/Query">
        <typename xmlns:src="http://xml.apache.org/
          xindice/Query">
          <tysource sourcetype="imsdefault"
            xmlns:src="http://xml.apache.org/
              xindice/Query" />
          <tyvalue xmlns:src="http://xml.apache.org/
            xindice/Query">ctjini050101</tyvalue>
        </typename>
      </fieldlabel>
      <fielddata xmlns:src="http://xml.apache.org/
        xindice/Query">3</fielddata>
    </unitsfield>
  </xapi:resource>
  <xapi:resource xapi:documentID="mike">
    <unitsfield xmlns:src="http://xml.apache.org/
      xindice/Query"
      src:col="/db/users" src:key="mike">
      <fieldlabel xmlns:src="http://xml.apache.org/
        xindice/Query">
        <typename xmlns:src="http://xml.apache.org/
          xindice/Query">
            <tysource sourcetype="imsdefault"
              xmlns:src="http://xml.apache.org/
                xindice/Query" />
            <tyvalue xmlns:src="http://xml.apache.org/
              xindice/Query">ctjini050101</tyvalue>
          </typename>
        </fieldlabel>
        <fielddata xmlns:src="http://xml.apache.org/
          xindice/Query">5</fielddata>
      </unitsfield>
    </xapi:resource>
  </xapi:resourceSet>
</ns1:executeQueryReturn>
</ns1:executeQueryResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

There are currently four methods that can be used to invoke the LIMS2 web service in Java alone. The web service can also be invoked using different programming languages. The first is to use a custom Java-class that we have implemented

called `XMLDataClient`. This class basically acts like a stub, just as with Java RMI or some other RPC protocol. The object provides the exact same method signatures as the web service itself, and it performs the XML messaging (SOAP, described previously) behind the scenes. The next two invocation methods are client libraries that can be downloaded and used to invoke methods against the web service given the URL of its end-point or WSDL specification. These libraries are the Apache Axis client library¹ which makes calls against the end-point URL, and the Web Services Invocation Framework (WSIF)³ which makes calls against the available WSDL specification. Finally, Java has another API package, `javax.xml.soap` that provides classes for opening and maintaining a SOAP connection with an end-point URL and constructing messages to send through that connection.

4. Conclusion

To summarize, our XML access control system provides a flexible model that can be efficiently implemented for access control of inter-related XML documents in many existing web service platforms. We find that it is important not only to authenticate and authorize users based on their username, roles, groups, operations, and other usual means, but also complex relationships between system entities and different context levels chosen by the authorization system.

Acknowledgements

The work presented in this paper was partially supported by the National Science Foundation under grant EIA-0086230. The authors also wish to acknowledge the insightful comments of the unknown reviewers.

References

1. Apache Software Foundation, Apache Axis, <http://ws.apache.org/axis>.
2. Apache Software Foundation, Apache Xindice, <http://xml.apache.org/xindice>.
3. Apache Software Foundation, Web Service Invocation Framework, <http://ws.apache.org/wsif>.
4. Apache Software Foundation, Apache Cocoon, <http://cocoon.apache.org>.
5. M. Aoyama, S. Weerawarana, H. Maruyama, C. Szyperski, K. Sullivan and D. Lea, Web services engineering: Promises and challenges, in *Proceedings of the 24th International Conference on Software Engineering*, Orlando, FL (2002), pp. 647–648.
6. E. Bertino, S. Castano and E. Ferrari, Securing XML documents with Author-X, *IEEE Internet Computing* **5**(3) (2001) 21–23.
7. E. Bertino, B. Carminati and E. Ferrari, A temporal key management scheme for secure broadcasting of XML documents, in *Proceedings of the 9th ACM Conference on Computer and Communication Security*, Washington, DC (2002), pp. 31–40.
8. E. Damiani, S. De Capitani di, S. Paraboschi and P. Samarati, Fine grained access control for SOAP e-services, in *Proceedings of the 10th International World Wide Web Conference*, Hong Kong (2001), pp. 504–513.
9. IMS. IMS learner information package specification, <http://www.imsproject.org/profiles>.

10. M. Kudo and S. Hada, XML document security based on provisional authorization, in *Proceedings of the 7th ACM Conference on Computer and Communication Security* (2000), pp. 87–96.
11. V. Parmar, H. Shi and S. Chen, XML access control for semantically related XML documents, in *Proceedings of the 36th Hawaii International Conference on System Science (HICSS-36)*, Big Island, HI (2003).
12. Y. Shang, H. Shi and S. Chen, An intelligent distributed environment for active learning, *Journal of Educational Resources in Computing* **1**(2) (2001) 1–17.
13. Y. Wang and K. Tan, A scalable XML access control system, in *Proceedings (Poster) of the 10th International World Wide Web Conference*, Hong Kong (2001), pp. 150–151.
14. World Wide Web Consortium, XML Path Language (XPath), version 1.0, W3C Recommendation (16 November 1999), <http://www.w3.org/TR/xpath>.
15. World Wide Web Consortium, Simple Object Access Protocol (SOAP) 1.1, W3C Note (08 May 2000), <http://www.w3.org/TR/SOAP>.
16. World Wide Web Consortium, Extensible Markup Language (XML) 1.0 (2nd edn.), W3C Recommendation (6 October 2000), <http://www.w3.org/TR/REC-xml>.
17. World Wide Web Consortium, Web Services Description Language (WSDL) 1.1, W3C Note (15 March 2001), <http://www.w3.org/TR/wsdl>.
18. XML:DB Initiative, Application programming interface for XML databases, <http://www.xmldb.org/xapi/xapi-draft.html>.
19. XML:DB Initiative, XUpdate — XML Update Language, <http://www.xmldb.org/xupdate/xupdate-wd.html>.

Copyright of International Journal of Information Technology & Decision Making is the property of World Scientific Publishing Company and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.