CHAPTER 2

Author-$\mathcal{X}$: A JAVA-BASED SYSTEM FOR XML DATA PROTECTION*

E. Bertino, M. Braun, S. Castano, E. Ferrari, M. Mesiti

Abstract Author-$\mathcal{X}$ is a Java-based system for access control to XML documents. Author-$\mathcal{X}$ implements a discretionary access control model specifically tailored to the characteristics of XML documents. In particular, our system allows (i) a set-oriented and single-oriented document protection, by supporting authorizations both at document type and document level; (ii) a differentiated protection of document/document type contents by supporting multi-granularity protection objects and positive/ negative authorizations; (iii) a controlled propagation of authorizations among protection objects, by enforcing multiple propagation options.

Keywords: XML, access control, authorization base, eXcelon, Java.

1. Introduction

Since the Web is becoming the main information dissemination means for most organizations, an increasing number of applications at Internet and Intranet level need access control mechanisms enforcing a selective access to information retrieved/exchanged over the Web. XML [9] has recently emerged as the most relevant standardization effort in the area of markup languages, and it is increasingly used as the language for information exchange over the Web. In this context, developing an access control mechanism in terms of XML is an important step for Web information security.

In this paper, we present Author-$\mathcal{X}$, a Java-based system for discretionary access control to XML documents. Author-$\mathcal{X}$ takes into account XML document characteristics, the presence of document types (called *Document Type Definitions* (DTDs)), and the types of actions that can be executed on XML documents (i.e., navigation and browsing), for implementing an access control mechanism tailored to XML. In particular,

*This work has been partially supported by a grant from Microsoft Research.

Author-$\mathcal{X}$ has the following distinguishing features: both a *set-oriented* and *instance-oriented* document protection, by supporting DTD-level as well as document-level authorizations; *differentiated protection* of XML document and DTD contents by supporting positive and negative authorizations and fine grained protection objects, identified on the basis of the graph structure of XML documents and DTDs; *controlled propagation* of authorizations among protection objects at different granularity levels, by enforcing multiple propagation options stating how an authorization defined on a document/DTD applies by default to protection objects at a finer granularity level within the document/DTD.

Author-$\mathcal{X}$ exploits authorizations stored in an XML authorization base and their propagation options to evaluate access requests issued by users and determines if they can be completely satisfied, partially satisfied, or not satisfied at all. In case of a partially satisfied request, only a view of the requested document(s) is returned by Author-$\mathcal{X}$. Author-$\mathcal{X}$ is implemented in Java on top of the eXcelon data server [4], which is used to store both the sources to be protected and the XML authorization base of the system. Architectural and implementation issues of Author-$\mathcal{X}$ are described, with particular attention to the authorization base and the access control mechanism. An application of Author-$\mathcal{X}$ to the protection of a real XML source, derived from the *Sigmod Record Articles XML Database* [7], is presented.

As far as we know, Author-$\mathcal{X}$ is the first tool, we are aware of, supporting XML document protection. In fact, research work in this field has concentrated more on the development of access control models for Web documents [6]. XML documents have a richer structure than HTML documents and can be coupled with DTDs describing their structures. Such aspects require the definition and enforcement of more sophisticated access control mechanisms for XML, than the ones devised for HTML. An access control model for XML documents has been recently proposed in [3]. Such model borrows some ideas from previous models for object-oriented databases and does not actually take into account some peculiarities of XML. For example, the case of documents not conforming/partially conforming to a DTD is not considered, and no support is provided to the Security Officer for protecting such documents.

The paper is organized as follows. Section 2 summarizes characteristics of the Author-$\mathcal{X}$ discretionary access control model and describes the overall system architecture. Section 3 describes the structure of an XML source and of the authorization base. Section 4 presents the access control module of Author-$\mathcal{X}$. Section 5 illustrates expected interactions of the Security Officer with Author-$\mathcal{X}$ for authorization management.

Finally, Section 6 concludes the paper and outlines future research directions.

2. Overview of Author-X

In the following we first briefly review the access control model of Author-X [1]. Then, we present its overall architecture.

2.1. Author-X access control model

Authorizations in the Author-X model have the following format:

`< users, protection-objs, priv, prop-opt, sign >`

Component `users` denotes a (set of) user(s) to which the authorization applies. Component `protection-objs` denotes the (portions of) documents/DTDs (called *protection objects*) to which the authorization applies. The `priv` component denotes the access modes that can be exercised on the protection objects specified in the authorization. We support two different kinds of privileges: *browsing* and *authoring* privileges. Browsing privileges allow users to read the information in an element (`read` privilege) or to navigate through its links (`navigate` privilege). Authoring privileges allow users to modify (or delete) the content of an element (`write` privilege) or to append new information in an element (`append` privilege). The `prop-opt` component allows one to specify how authorizations specified at a given level propagate to lower level elements. The following options are provided: *i)* CASCADE: the authorization propagates to all the direct and indirect subelements of the element(s) specified in the authorization; *2)* FIRST_LEV: the authorization propagates only to all the direct subelements of the element(s) specified in the authorization; *3)* NO_PROP: no authorization propagation occurs. Finally, the component **sign** $\in \{+, -\}$ specifies whether the authorization is a permission (**sign** = **+**) or a prohibition (**sign** = **-**).

In addition to such "explicit" propagation, Author-X supports a form of "implicit" propagation according to which an authorization specified on a certain protection object *o* "applies by default" to a set of protection objects that have a relationship with *o*. In Author-X, the relationships considered for propagation are the *element-to-subelements*, *element-to-attributes*, *element-to-links* relationships, deriving from the graph structure of documents and DTDs, and the *DTD-to-instances* relationship, holding between a DTD and the set of its valid instances. Note that, the possibility of specifying both positive and negative authorizations allows the Security Officer to always override these "by default" propagation principles.

The possibility of specifying both positive and negative authorizations introduces potential conflicts among authorizations, in that a user may have two authorizations for the same privilege on the same protection object but with different signs. These conflicting authorizations can be either explicit or derived through propagation. We do not consider the simultaneous presence of conflicting authorizations as an inconsistency; rather we define a conflict resolution policy which is based on the notion of *most specific authorization*. The conflict resolution policy of Author-$\mathcal{X}$ is based on the following principles: authorizations specified at the document level prevail over authorizations specified at the DTD level; authorizations specified at a given level in the DTD/document hierarchy prevail over authorizations specified at higher levels; when conflicts are not solved by the previous rules, we consider as prevailing negative authorizations.

2.2. Architecture of Author-$\mathcal{X}$

Author-$\mathcal{X}$ is built on top of eXcelon [4], an XML data server for building Web applications. EXcelon manages an XMLstore where the XML data can be indexed and manipulated using the Document Object Model (DOM) [8], and queried using the XQL language [5]. Programmers can extend eXcelon functionalities by writing Java server extensions. The purpose of server extensions is to extend the eXcelon server with custom modules to cover specific application requirements.

Figure 1 shows the general architecture of an eXcelon document server enhanced with Author-$\mathcal{X}$. Author-$\mathcal{X}$ components of the architecture are:

- *XMLStore*, which is organized in two components: *XML source*, which stores XML documents and DTDs to be protected, and $\mathcal{X}$*-base*, the authorization base storing authorizations.
- $\mathcal{X}$*-core*, which is the main component of the architecture. It is composed of two Java server extensions, $\mathcal{X}$*-access* and $\mathcal{X}$*-admin*. $\mathcal{X}$*-access* is the server extension implementing access control over the *XML source* based on authorizations contained in $\mathcal{X}$*-base*. $\mathcal{X}$*-admin* is the server extension providing support functionalities to the Security Officer for authorization management.

$\mathcal{X}$*-core* is part of the eXcelon data server and interacts with the external environment by means of an eXcelon client API. Users and the Security Officer interact with $\mathcal{X}$*-core* by means of specific user applications, or through the Web, using an eXcelon explorer or a Web server extension. Users submit access requests which are processed by the $\mathcal{X}$*-access* component of $\mathcal{X}$*-core*, and receive back the (portion of) re-

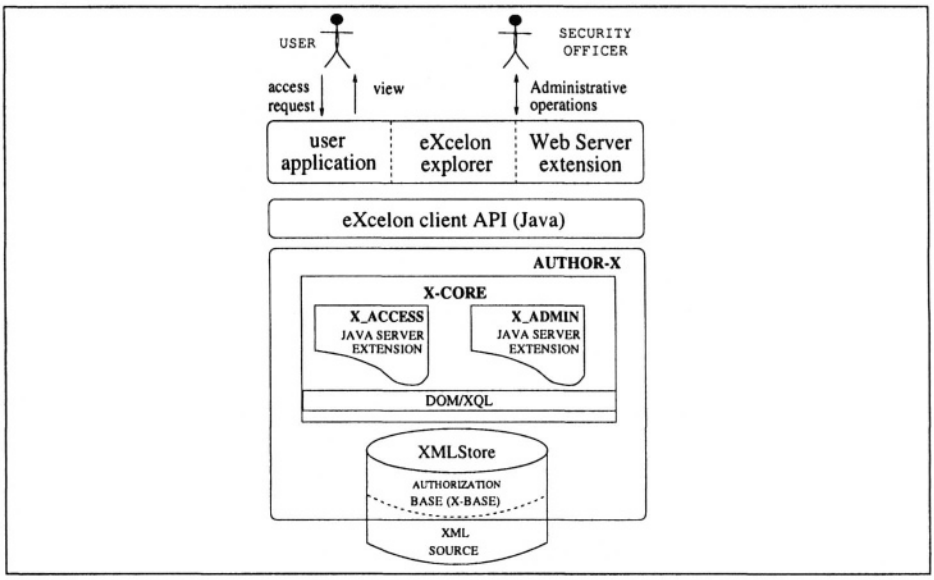

Figure 1: Architecture of an eXcelon document server enhanced with Author-X

quested data in the *XML source* they are authorized for. The Security Officer interacts with the *X-admin* component of *X-core*, for performing administrative operations on authorizations in the *X-base*.

3. Structure of Author-X XMLStore

In this section, we describe in more detail the structure of the XML-Store, i.e., the *XML source* and the *X-base*.

3.1. XML source

The *XML source* component of the XMLStore contains XML documents to be protected with their DTDs, if defined. In particular, the source can contain well-formed or valid documents. A *well-formed* document is a document that follows the grammar rules of XML [9], while, *valid documents* have an associated DTD defining their structure.

To illustrate functionalities of Author-X, we have defined an *XML source* derived from the *Sigmod Record Articles XML Database* [7], containing sigmod record documents and associated DTDs. Figure 2(a) shows a portion of an XML document in the *XML source*. For each sigmod record issue, the document provides information about the number and volume, and about articles therein contained. Each article is characterized by information about title, authors, abstract, initial page and final page in the issue. Moreover, information about related articles

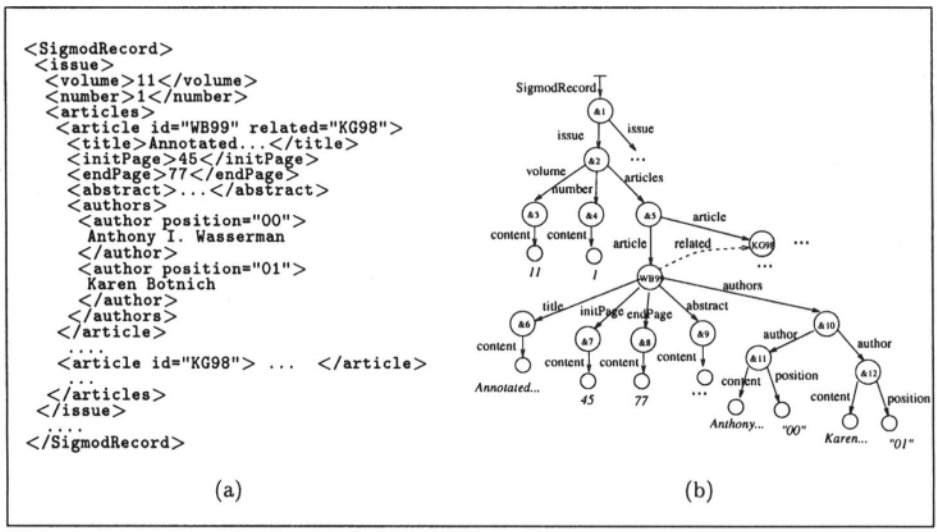

Figure 2: (a) An example of SigmodRecord XML document and (b) its corresponding graph representation

is provided in the document. Figure 2(b) shows a graph representation of the XML document in Figure 2(a). This representation is compliant with the Document Object Model (DOM) specification [8] adopted by eXcelon to internally represent XML documents.

3.2. $\mathcal{X}$-base

The $\mathcal{X}$-*base* part of the XMLStore is an XML file (`auth.xml`) storing authorizations on the XML source contents. Authorizations in the $\mathcal{X}$-*base* conform to the DTD presented in Figure 3.

According to the DTD of Figure 3, authorizations are organized into an `authorizations` XML document with a subelement `users`, denoting users, and a subelement `auths`, denoting access authorizations. The `users` subelement contains a `user` subelement for each user to be authorized to access the *XML source*. Each `user` element is identified by the login of the corresponding user (attribute `id`) and contains the password of the user (attribute `passwd`). The `auths` element contains an `authspec` subelement for each authorization given to a certain user on the *XML source*. Each `authspec` element is characterized by the following attributes:

- `userid`: it contains a reference to the user to which the authorization refers;

```
<!DOCTYPE authorizations[
<!ELEMENT authorizations (users,auths)>
<!ELEMENT users (user)+>
<!ELEMENT user EMPTY>
<!ELEMENT auths (authspec)*>
<!ELEMENT authspec EMPTY>
<!ATTLIST user id ID #REQUIRED passwd CDATA #REQUIRED>
<!ATTLIST authspec userid IDREF #REQUIRED target CDATA #REQUIRED path CDATA #REQUIRED
          priv (READ | NAVIGATE | APPEND | WRITE) #REQUIRED
          type (GRANT | DENY) #REQUIRED
          prop (NO_PROP | ONE_LEVEL | CASCADE) #REQUIRED> ]>
```

Figure 3: *X-base* DTD

- `target`: it stores the file name of the XML document/DTD to which the authorization refers;
- `path`: it stores a path within the `target` document corresponding to the specific protection object(s) to which the authorization applies, `path` is based on the following notation, compliant with Xpath [10] and XQL language [5]:

  /[{elem['[expr]']/}]TRGelem['[expr]'][/@TRGattr['[expr]']]

  where: first symbol '/' denotes the root element; `TRGelem` denotes the name of a target element; `TRGattr` denotes the name of a target attribute;[1] {elem/} denote optional intermediate element(s) (separated by '/') in the path from the root element to the target element/attribute in the considered file, and [expr] is an optional condition on an element/attribute content to select specific document portions in the considered XML source files.
- `perm`: it stores the authorization privilege (i.e., READ, NAVIGATE, APPEND, or WRITE);
- `type`: it stores the authorization type (i.e., GRANT, or DENY);
- `prop`: it stores the propagation option of the authorization (i.e., NO_PROP, ONE_LEVEL, or CASCADE).

Example 1 *An example of X-base is shown in Figure 4. According to this authorization base, users* `Mary` *and* `Rose` *are authorized to read all information about issues contained in the Sigmod Record document of the* XML source, *except articles'* `abstract`. *Additionally,* `Mary` *is authorized to read all the information about the article identified by* `WB99`. ○

[1] Symbol @ denotes attribute names as in Xpath and XQL.

```
<authorizations>
 <users>
  <user id="Mary" passwd="%$#$%"/ >
  <user id="Rose" passwd="&#$%"/ >
 </users>
 <auths>
  <auth-spec userid="Mary" target="SigmodRecord.dtd" path="/issue"
    priv="READ" type="GRANT" prop="CASCADE"/ >
  <auth-spec userid="Rose" target="SigmodRecord.dtd" path="/issue"
    priv="READ" type="GRANT" prop="CASCADE"/ >
  <auth-spec userid="Mary" target="SigmodRecord.dtd" path="/issue/articles/article/abstract"
    priv="READ" type="DENY" prop="NO_PROP"/ >
  <auth-spec userid="Rose" target="SigmodRecord.dtd" path="/issue/articles/article/abstract"
    priv="READ" type="DENY" prop="NO_PROP"/ >
  <auth-spec userid="Mary" target="SigmodRecord.xml"
    path="/issue/articles/article[@id='WB99']" priv="READ" type="GRANT" prop="CASCADE"/ >
 </auths>
</authorizations>
```

Figure 4: An example of $\mathcal{X}$-*base*

4. The $\mathcal{X}$-access component of Author-$\mathcal{X}$

The $\mathcal{X}$-access component of Author-$\mathcal{X}$ enforces access control on the *XML source*.

Users request access to documents under two different modalities: *browsing* and *authoring*. A user requests a browsing access when he/she wants to access a document (and navigating its links), without modifying it, whereas, requests an authoring access when a modification of the document is required. Access can also be requested wrt a specific portion(s) of a document. Thus, an access request `r` is represented as a tuple **r= ⟨user,target,path,acc_modality⟩**, where `user` is the user requesting the access, `target` is the XML document to which the access is requested, `path` is a path within the requested document (specified through an XQL query [5]) which eventually selects specific portions of the requested document, and `acc_modality` $\in$ {`browsing`, `authoring`} specifies whether a browsing or authoring access is requested.

Upon issuing an access request `r`, $\mathcal{X}$-access checks which authorizations (both positive and negative) `user` has on the `target` document. Such authorizations can be either explicitly stored in the $\mathcal{X}$-*base* or implicitly given by the propagation policies enforced by Author-$\mathcal{X}$ model. Based on such authorizations, `user` can receive a *view* of the requested document that contains only those portions for which he/she has a corresponding positive authorization which is not overridden by a negative conflicting authorization. In the case of totally authorized requests, the view coincides with the whole document (or with all the requested portions in the case the user does not require the access to the whole document). When, no positive authorizations are found for the requested document, or all of them are overwritten by negative authorizations, the access is denied.

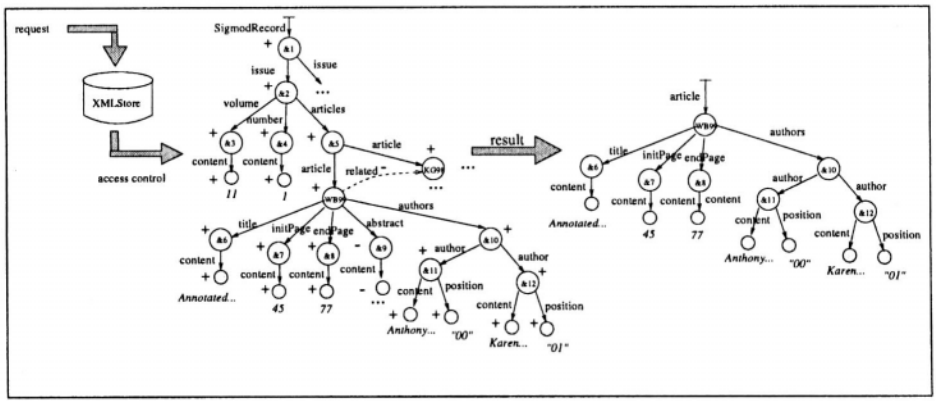

Figure 5: The access control process

To enforce access control, Author-X adopts the following strategy: all the elements and/or attributes for which `user` does not have an appropriate authorization are removed from the `target` document, before evaluating the `path` contained in the access request. The `path` is then evaluated against such pruned version and the result is returned to `user`. This strategy allows us to define an access control mechanism independent from any query language. This possibility is very important, because XQL is not yet a standard query language.

For lack of space we do not report here the access control algorithm implementing the above strategies. A detailed description of the algorithm can be found in [1].

Example 2 *Suppose that* `Rose` *submits the access request*

⟨`Rose, SigmodRecord.xml,/issue/articles/article[@id ='WB99'],browsing`⟩

Figure 5 shows the access control process. Author-X extracts the browsing authorizations for `Rose` *(from the X-base reported in Figure 4) and evaluates them against the file* `SigmodRecord.xml`. *The result of the evaluation is a graph, where each node is labeled with symbol "–", if a negative authorization applies to the corresponding attribute/element, or with symbol "+", if a positive authorization applies to the corresponding attribute/element. The view to be returned to* `Rose` *is obtained by pruning from the graph nodes with a label different from "+", and by extracting from the resulting graph the elements/attributes identified by the path:* `/issue/articles/ article[@id='WB99']`. *As a result, a view of article* `WB99` *is returned to* `Rose` *(shown on the right hand side of Figure 5) that does not contain the* `abstract` *element.* ○

Figure 6 shows the graphical interface provided by Author-X for access request submission (the access request is the one of Example 2).

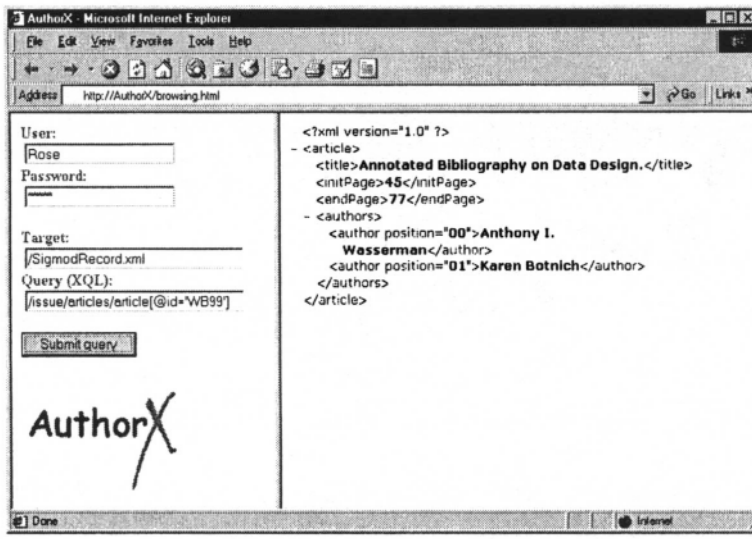

Figure 6: Access request submission in Author-$\mathcal{X}$

The left hand side of the figure shows how the user can submit his/her request, whereas the right hand side shows the corresponding result.

5. The $\mathcal{X}$-admin component of *Author-$\mathcal{X}$*

The $\mathcal{X}$-admin component of Author-$\mathcal{X}$ is designed to support administrative operations on authorizations under responsibility of the Security Officer. In particular, $\mathcal{X}$-admin supports the Security Officer in defining new authorizations to protect XML documents in the *XML source*.

Author-$\mathcal{X}$ supports two kinds of policies for authorization specification: a *DTD-based policy*, in which authorizations are specified at the DTD level, and propagated within the DTD due to the element-to-subelements and the element-to-attributes/links relationships as well as to all XML documents that are valid instances of the DTD, due to the DTD-to-instances relationship, and a *document-based policy*, in which authorizations are specified at the document level, and apply only to the considered document. In this case, authorization propagation occurs only due to the element-to-subelements and the element-to-attributes/links relationships within the considered document.

Based on these policies, authorization management for valid and well-formed documents can be enforced by the Security Officer in Author-$\mathcal{X}$ as follows:

- **Valid document protection**: the DTD-based policy is adopted, in that valid documents are instances of some DTD in the source.

As a consequence, the Security Officer specifies authorizations at the DTD level which apply by default to all its valid document instances. Exceptions to the policy defined at the DTD level are modeled by specific authorizations defined in the *X-base* on the involved document instances. For instance, if all information in a set of valid documents have the same protection requirements, then the Security Officer invokes the definition of authorizations at the DTD level, with the CASCADE option. By contrast, when different subelement(s) (respectively, attributes/links) of a DTD need different authorization policies, it is convenient to guarantee a minimal common protection on the whole DTD by defining an authorization with the NO_PROP or FIRST_LEV propagation option. A number of additional authorizations are then defined on the subelement(s) (respectively, attribute(s)/link(s)) of the DTD with the most appropriate privileges and sign to enforce the policy holding on the specific subelement(s) (respectively, attribute(s)/link(s)).

- **Well-formed documents**: different approaches are supported by Author-X. According to a *classification-based approach*, the Security Officer decides to adopt the DTD-based policy also for well-formed documents. To this end, well-formed documents to be protected are first classified against available DTDs in the source, with the goal of finding the "best matching" DTD. If such a DTD is found by the tool, protection of a well-formed document is enforced by propagating authorizations defined for the selected DTD to the document. This propagation can be total or partial, depending on the level of conformance between the well-formed document and the selected DTD (conformance vs. partial conformance). In case of partial conformance, the Security Officer can manually define additional authorizations on non-matching portions of the well-formed document, if necessary. According to an *instance-based approach*, the Security Officer defines from scratch authorizations needed to implement the access control policy for the considered well-formed document (document-based policy). The document-based policy is also adopted when no conforming DTD is found after the classification process, and also when exceptions for the policy defined in the selected DTD have to be specified, when the classification-based approach is taken.
  The classification-based approach exploits the propagation principle to limit the manual activity of the Security Officer in the definition of the authorization policy for a well-formed document. The classification-based approach relies on suitable mechanisms to

automatically derive all required authorizations for all the involved users. The Security Officer can interactively validate derived authorizations to check their suitability to the well-formed document to be protected.

6. Concluding Remarks

In this paper, we have presented Author-$\mathcal{X}$, a Java-based system for access control to XML sources. Author-$\mathcal{X}$ supports positive and negative authorizations for browsing and authoring privileges with a controlled propagation. Core functionalities of access control and authorization base management have been implemented as Java server extensions on top of the eXcelon data server. Currently, we are setting up a comprehensive administration environment, by developing interactive tool support for the Security Officer to guide the choice of the best policy to be adopted for document protection, based on the results of document classification process.

References

[1] E. Bertino, M. Brawn, S. Castano, B. Ferrari, and M. Mesiti. Author-$\mathcal{X}$: a Java-Based System for XML Data Protection. In pre-Proc. of 14th IFIP WG11.3 Working Conference on Database and Application Security. Schoorl, The Netherlands, August, 2000.

[2] E. Bertino, S. Castano, E. Ferrari, and M. Mesiti. Specifying and Enforcing Access Control Policies for XML Document Sources. *World Wide Web Journal*, 3(3), 2000.

[3] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Securing XML Documents. In *Proc. of EDBT*, 2000.

[4] Object Design Inc. An XML Data Server for Building Enterprise Web Applications, 1998. `http://www.odi.com/excelon`.

[5] J. Robie. XQL Tutorial, 2000. `http://www.ibiblio.org/xql/xql-tutorial.html`.

[6] P. Samarati, E. Bertino, and S. Jajodia. An Authorization Model for a Distributed Hypertext System. *IEEE TKDE*, 8(4):555–562, 1996.

[7] Sigmod Record XML Database, `http://www.dia.uniroma3.it/Areneus/Sigmod/`.

[8] W3C. Document Object Model 1, 1998. `http://www.w3.org/DOM/`.

[9] W3C. Extensible Markup Language 1.0, 1998. `http://www.w3.org/TR/REC-xml`.

[10] W3C. XML Path Language, 1.0, 1999. `http://www.w3.org/TR/xpath`.