ELSEVIER

Data & Knowledge Engineering 43 (2002) 237–260

DATA & KNOWLEDGE ENGINEERING

www.elsevier.com/locate/datak

Protection and administration of XML data sources ☆

Elisa Bertino [a,*], Silvana Castano [a], Elena Ferrari [b], Marco Mesiti [c]

[a] *Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Via Comelico, 39/41, 20135 Milano, Italy*

[b] *Dipartimento di Scienze Chimiche, Fisiche e Matematiche, Università degli Studi dell'Insubria, Via Valleggio, 11, 22100 Como, Italy*

[c] *Dipartimento di Informatica e Scienze dell'Informazione, Università degli Studi di Genova, Via Dodecaneso, 35, 16146 Genova, Italy*

Received 9 February 2002; received in revised form 9 February 2002; accepted 19 June 2002

Abstract

EXtensible Markup Language (XML) security has become a relevant research topic due to the widespread use of XML as the language for information interchange and document definition 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 the protection and administration facilities of Author-$\mathcal{X}$, a Java-based system for discretionary access control to XML documents. Relevant features of Author-$\mathcal{X}$ are both a set-oriented and a document-oriented credential-based document protection, a differentiated protection of document/document type contents through the support of multi-granularity protection objects and positive/negative authorizations, and the support for different access control strategies. In this paper, we focus on the strategies we have developed for enforcing access control. Additionally, we provide a description of the environment we have developed to help the Security Officer in performing administrative activities related to both security policy and subject credential management.
© 2002 Elsevier Science B.V. All rights reserved.

Keywords: XML security; Access control; Administration facilities; eXcelon DBMS; Java

☆ A preliminary version of this paper appeared in Proc. of 14th *IFIP WG11.3 Working Conference on Database and Application Security*, 2001, with the title "Author-$\mathcal{X}$: A Java-Based System for XML Document Protection".

* Corresponding author. Tel.: +39-02-55006202; fax: +39-02-55006373.

E-mail addresses: bertino@dsi.unimi.it (E. Bertino), castano@dsi.unimi.it (S. Castano), elena.ferrari@uninsubria.it (E. Ferrari), mesiti@disi.unige.it (M. Mesiti).

0169-023X/02/$ - see front matter © 2002 Elsevier Science B.V. All rights reserved.
PII: S0169-023X(02)00127-1

1. Introduction

EXtensible Markup Language (XML) [5,15] is currently the most relevant standardization effort in the area of document representation through markup languages and is rapidly becoming a standard for data representation and exchange over the Web. XML is based on some simple, yet powerful concepts. A key concept is the one of tagged element. A tagged element identifies a content portion of a document or a complex data object. Tags are defined by applications that are thus able to convey the semantics of the various content portions. Elements can be nested, that is, an element may consist of other elements, and attributes can be associated with elements. Attributes provide additional information on elements, thus increasing the semantics one can specify for elements. An additional key concept is the one of Document Type Definition (DTD), describing the structure of a set of similar documents. DTDs are instrumental in promoting standardization of application documents and data objects.

The widespread use of XML is pushing the need of models and techniques for securing XML data. Such models and techniques are crucial in order to facilitate a selective dissemination of XML data containing information of different sensitivity levels, among (possibly large) user communities. An overview of research work and commercial products related to XML security can be found at [6]. Securing XML documents entails addressing three main issues: *confidentiality*, *integrity*, and *authenticity*. Ensuring confidentiality means that the data object contents be only disclosed to subjects authorized according to the specified security policies. Ensuring integrity means ensuring that the object contents are not altered during transmission from the source to the intended recipient. Ensuring authenticity means that the subject receiving a data object is assured that the data object actually is from the source it claims to be. Confidentiality is ensured by access control mechanisms; integrity is usually enforced by access control mechanisms and by the use of encryption techniques, whereas authenticity requires the use of digital signature techniques [12].

In this paper, we focus on the confidentiality requirements of XML data by presenting the protection and administration facilities of Author-$\mathcal{X}$ [3], a Java-based system for discretionary access control to XML documents. Author-$\mathcal{X}$ takes into account XML document characteristics, the presence of DTDs describing the structure of documents at a schema level, 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. Author-$\mathcal{X}$ exploits authorizations stored in an XML policy base and specified propagation options to evaluate access requests 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-$\mathcal{X}$ to the requesting user. Author-$\mathcal{X}$ supports policy specification at varying granularity levels, ranging from a set of documents, to a specific document or to a document portion. Furthermore, Author-$\mathcal{X}$ supports the specification of *subject credentials* as a way to enforce access control based on subject qualifications and profiles. Author-$\mathcal{X}$ supports two different document distribution policies: *push distribution*, for document broadcast, and *pull distribution*, for document distribution on user demand. Additionally, Author-$\mathcal{X}$ is complemented by $\mathcal{X}$-Sec [2], an XML-based language for specifying subject credentials and security policies and for organizing them into credential and policy bases, respectively.

Managing such a complex access control system requires the design and implementation of suitable administration tools that help the Security Officer (SO) in performing administrative operations on security policies. A key requirement is to provide the SO with interactive tools for

controlling the effects of an operation on the policy base, such as for instance which are the authorizations implied by a security policy or which conflicts the specification of a new policy causes. To this purpose, Author-$\mathcal{X}$ provides a user-friendly environment for assisting the SO in performing administration activities related to policy and credential management.

In this paper, we focus on the strategies we have developed for enforcing the pull mode for access control. Details on the push mode can be found in [1]. Additionally, we provide a description of the comprehensive environment we have developed to help the SO in performing administrative operations on both the policy and the credential base. In particular, we illustrate the methodology Author-$\mathcal{X}$ provides for policy design and the tools that the SO can invoke to verify the effects of a new specified policy.

1.1. Related work

XML security is a recent research topic and work in this field has concentrated mainly on the development of access control models and encryption techniques [6]. Some recent related work is reported in [4]. However, such work mainly borrows some ideas from previous models for object-oriented databases and does not actually take into account some relevant peculiarities of XML. For example, the case of documents not conforming/partially conforming to a DTD is not addressed, and no support is provided to the SO for protecting such documents. Moreover, such work only provides the read access mode and does not support credential-based access control, nor it supports different access control strategies.

Other related work is in the area of HTML document protection. For instance, in [10] an authorization model has been proposed, where authorizations can be given either on the whole document or on selected portions within the document. XML documents have a richer structure than HTML documents and there is the possibility of attaching a DTD to an XML document, describing its structure. Such aspects require the definition and enforcement of more sophisticated access control mechanisms for XML, than the ones devised for HTML documents.

Original features of the Author-$\mathcal{X}$ system are the support for different policies for securing XML documents also in the case of partially and not conforming documents, and the support for a number of specialized access modes for browsing and authoring. Furthermore, Author-$\mathcal{X}$ provides, in addition to the traditional user-on-demand mode for document release, a broadcast mode based on a combination of digital signatures and encryption techniques [1]. As far as we know, Author-$\mathcal{X}$ is the first tool, we are aware of, supporting such a comprehensive environment for the protection of XML documents, with a set of tools for enforcing access control according to different strategies and for performing administrative operations on both security policies and subject credentials.

1.2. Organization of the paper

The paper is organized as follows. Next section provides a general overview of the Author-$\mathcal{X}$ system. Section 3 deals with access control for XML document protection, whereas Section 4 presents the Author-$\mathcal{X}$ facilities for credential management, and for policy specification, validation, and maintenance. Finally, Section 5 concludes the paper and outlines future research directions.

2. Overview of Author-$\mathcal{X}$

The Author-$\mathcal{X}$ system has been conceived for enforcing XML document protection and security administration, with the following requirements:

- *Identity-based and credential-based user qualification*. According to an identity-based access qualification, Author-$\mathcal{X}$ allows the specification of security policies in terms of user identity. Moreover, Author-$\mathcal{X}$ supports the specification of *subject credentials* as a way to enforce a more flexible access control, taking into account subject profiles with a variety of characteristics. Subject credentials assert properties of a subject, either personal characteristics, or characteristics and properties deriving from relationships the subject has with other subjects (e.g., qualifications within an organization) [13].
- *Differentiated protection of documents/DTDs* stored in an XML source to reflect the different protection requirements against the source. An XML document (whose graph representation is shown in Fig. 1) can contain information with different protection requirements in different portions of its graph structure. The same considerations hold for DTDs. Author-$\mathcal{X}$ enforces different security policies on different portions/elements of the same document/DTD, by implementing a fine-level authorization granularity.
- *Propagation of security policies* defined on documents/DTDs and/or portions of them. According to the propagation principle, a policy specified on a certain protection object o "applies-by-default" to a set of protection objects that have a relationship with o in the graph document structure. In Author-$\mathcal{X}$, 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.
- *Tool-assisted security administration* to help the SO in performing administration activities related to policy and credential management. Author-$\mathcal{X}$ has been conceived to interactively support the specification and maintenance of subject credentials and security policies for documents/DTDs with a structure, possibly complex. All functionalities are supported by graphical tools to limit as much as possible the administrator effort required to the SO.

Fig. 2 shows the general architecture of the Author-$\mathcal{X}$ system. Author-$\mathcal{X}$ is built on top of the eXcelon [7] XML server. eXcelon manages an XML source where the XML data can be indexed

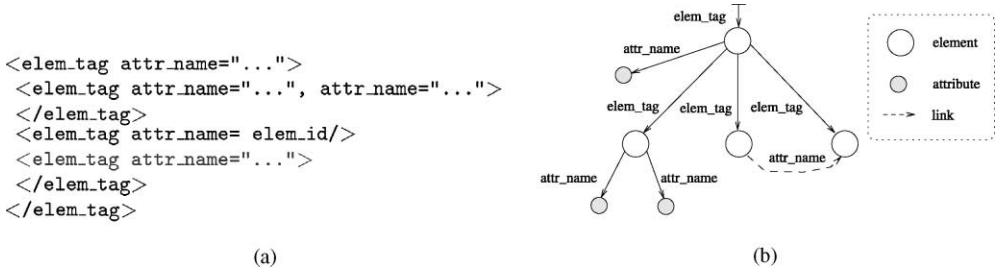


Fig. 1. (a) Example of XML document and (b) its corresponding graph representation.

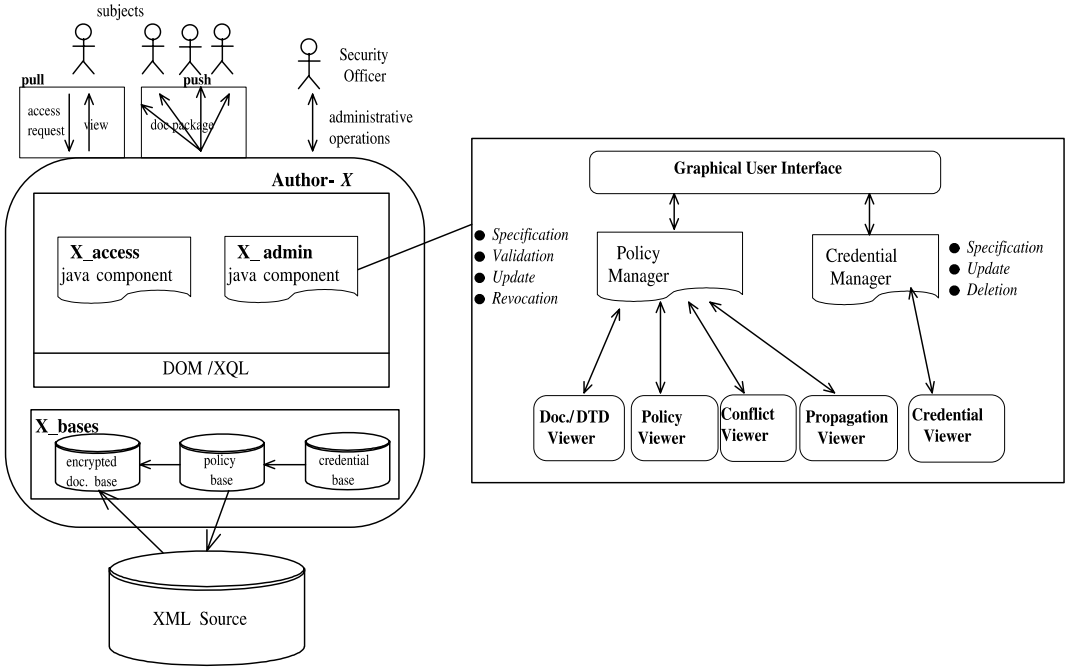


Fig. 2. Architecture of Author-$\mathcal{X}$.

and manipulated using the Document Object Model (DOM) [16] and queried using the XQL language [9].

The XML source contains XML documents to be protected with their DTDs, if defined. The main component of the architecture is composed of two Java server extensions, $\mathcal{X}$-*access* and $\mathcal{X}$-*Admin*. $\mathcal{X}$-*access* is the Java module implementing access control over the XML source. $\mathcal{X}$-*Admin* is the Java module providing support functionalities to the SO for security administration.

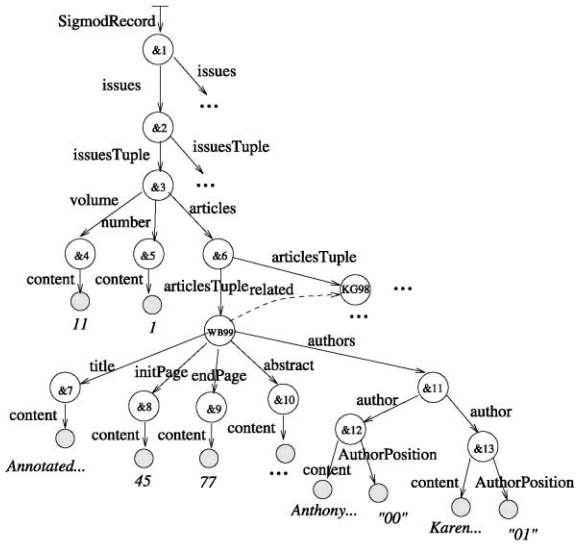
XML document protection is enforced according to both pull and push dissemination modes. Under the pull mode, subjects explicitly request XML documents to the source when needed, as in traditional DBMS environments. By contrast, under the pull mode, the source periodically P (or whenever some relevant events arise) broadcasts documents to subjects, without the need of an explicit request.

Three repositories ($\mathcal{X}$-*bases*) store security information: the *policy base*, which contains the security policies holding on the XML source; the *credential base*, which contains subject credentials and credential-types defined for the considered source, and the *encrypted document base*, which contains an encrypted copy of (a portion of) XML documents of the XML source used for information push. Security information in the policy base and in the credential base are specified in $\mathcal{X}$-Sec, the XML language for security information specification of Author-$\mathcal{X}$ [2].

Author-$\mathcal{X}$ interacts with the external environment by means of an eXcelon client API. Users and the SO interact with Author-$\mathcal{X}$ by means of specific applications, or through the Web, using eXcelon explorer or a Web browser.

```
<SigmodRecord>
 <issues>
  <issuesTuple>
   <volume>11</volume>
   <number>1</number>
   <articles>
    <articlesTuple id="WB99" related="KG98">
     <title>Annotated...</title>
     <initPage>45</initPage>
     <endPage>77</endPage>
     <abstract>...</abstract>
     <authors>
      <author AuthorPosition="00">
       Anthony I. Wasserman
      </author>
      <author AuthorPosition="01">
       Karen Botnich
      </author>
     </authors>
    </articlesTuple>
    ....
    <articlesTuple id="KG98">
     ...
    </articlesTuple>
     ...
   </articles>
  </issuesTuple>
  ....
 </issues>
 ....
</SigmodRecord>
```

(a)



(b)

Fig. 3. (a) A portion of Sigmod Record XML document and (b) its corresponding graph representation.

In the following, we focus on the protection of XML documents according to the pull mode and on the security administration facilities of Author-$\mathcal{X}$. XML document protection under the push mode is described in [1].

Fig. 3(a) shows a portion of an XML document source extracted from the *Sigmod Record Articles XML Database* [11], that we consider as a running example in the paper. For each 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 is provided in the document. Fig. 3(b) shows the graph representation of the XML document in Fig. 3(a). Fig. 4(a) shows the DTD for the Sigmod Record document of Fig. 3, whereas Fig. 4(b) shows the DTD graph representation. Examples given in the remaining of the paper will refer to the XML document and DTD illustrated in Figs. 3 and 4, respectively.

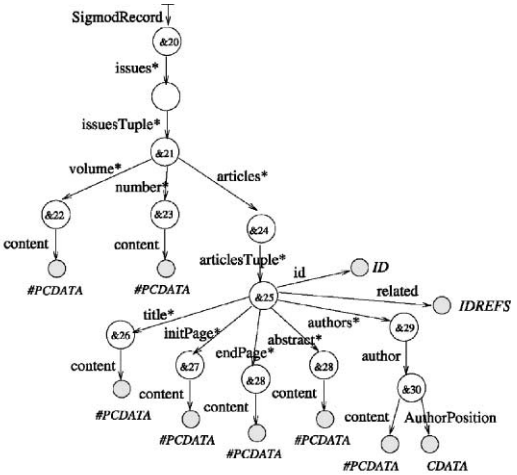
In the remainder of the section, we describe the most relevant characteristics of the credential base and of the policy base.

2.1. Credential base

The credential base contains the information to qualify the users subscribed to Author-$\mathcal{X}$, who are authorized to access documents in the XML source. Author-$\mathcal{X}$ supports both a conventional

```
<!DOCTYPE SigmodRecord[
<!ELEMENT SigmodRecord (issues*)>
<!ELEMENT issues (issuesTuple*)>
<!ELEMENT issuesTuple (volume*,number*,articles*)>
<!ELEMENT volume (#PCDATA)>
<!ELEMENT number (#PCDATA)>
<!ELEMENT articles (articlesTuple*)>
<!ELEMENT articlesTuple (title*,initPage*,endPage*,
                  abstract*,authors*)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT initPage (#PCDATA)>
<!ELEMENT endPage (#PCDATA)>
<!ELEMENT abstract (#PCDATA)>
<!ELEMENT authors (author*)>
<!ELEMENT author (#PCDATA)>
<!ATTLIST author AuthorPosition CDATA #REQUIRED>
<!ATTLIST article id ID #REQUIRED
      related IDREFS #IMPLIED>
]>
```

(a)



(b)

Fig. 4. (a) DTD specification for the document of Fig. 3 and (b) its corresponding graph representation.

identity-based scheme and a more advanced credential-based access control scheme to documents in an XML source. According to an identity-based access control scheme, security policies are expressed in terms of user identity. In this type of policies, users holding accounts in the XML source are considered as subjects of security policies, identified according to an ID-based mechanism (e.g., the login name with which the user connects to the server).

To provide a more flexible and expressive way of specifying security policies, Author-$\mathcal{X}$ supports also a credential-based access control scheme. In this way, the specification of security policies becomes more direct and intuitive, since security policies are defined in general terms, close to the high-level rules and conventions holding for the documents to be protected.

Credential specification in Author-$\mathcal{X}$ is based on the concept of *credential-type*. A credential-type is a template for the specification of subject credentials with a similar structure. Examples of credential-types for the `SigmodRecord` source are `ACMmember`, `noACMmember`, and `SpecialInterestUser`. A credential-type is a pair of the form:

$$\langle \mathit{CredentialName},\ \mathit{CredentialProperties} \rangle$$

where:

- *CredentialName* is the name of a credential-type *ct*.
- *CredentialProperties* is a set of property specifications for *ct*. A property specification provides the name and the domain of a property. Author-$\mathcal{X}$ allows the definition of either simple or composite properties for credential-types. Simple properties take values from basic domains (e.g., `integer`, `string`) whereas composite properties take value from domains defined by applying conventional constructors (e.g., `set`, `record`, `list`) on basic domains. Credentials are certified by a credential issuer (e.g., a certification authority) using standard digital signature techniques [12]. For example, the `ACMmember` credential-type has the following properties: the name of the ACM member (in particular, the first and last names, and, optionally,

```
<!DOCTYPE ACMmember[
<!ELEMENT ACMmember (name,organization,
                     email*, memberNr)>
<!ELEMENT name (fname, mname?, lname)>
<!ELEMENT organization (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT memberNr (#PCDATA)>
<!ELEMENT fname (#PCDATA)>
<!ELEMENT mname (#PCDATA)>
<!ELEMENT lname (#PCDATA)>
<!ATTLIST ACMmember credID ID #REQUIRED>
<!ATTLIST ACMmember CIssuer CDATA #REQUIRED >
]>
```

(a)

```
<ACMmember credID="154"
           CIssuer = "CA16">
  <name>
    <fname> Tom </fname>
    <lname > Watson </lname>
  </name>
  <organization> MIT </organization>
  <email> twatson@mit.com </email>
  <memberNr> 2001 </memberNr>
</ACMmember>
```

(b)

Fig. 5. (a) Example of $\mathscr{X}$-Sec credential-type and (b) a corresponding subject credential.

the middle name), the organization of the ACM member, the email addresses, and the ACM membership number.

In the credential base, credentials are represented as XML documents, while credential-types as DTDs. An XML credential has two default attributes, namely the `credID` of type `ID`, to specify the credential identifier in the credential base, and the `CIssuer` attribute of type `CDATA`, to identify the issuer of the credential. The credential base contains also the identifier and password of the users subscribed to Author-$\mathscr{X}$. An example of $\mathscr{X}$-Sec credential-type and of a corresponding subject credential are reported in Fig. 5(a) and (b), respectively.

On credentials contained in the credential base, the SO can define *credential expressions* for identifying the users to which a policy applies. Credential expressions can identify all the users with a given credential (e.g., all `ACMmember`) or all the users whose credential properties satisfy specified conditions (e.g., all `ACMmember` whose ACM membership number belongs to a specified range). Credential expressions are specified through Xpath [14].

2.2. Policy base

The policy base stores $\mathscr{X}$-Sec policies defined for protecting the XML source contents. $\mathscr{X}$-Sec policy base conforms to the DTD reported in Fig. 6. The policy base is an XML document with a subelement `policySpec` for each security policy defined for the XML Source. Each `policySpec` element is characterized by the following subelements:

- `subject`, identifying the users to whom the policy refers. Users can be identified through identifiers (`user` subelement of `subject`) or can be qualified by a credential expression (`credential` subelement of `subject`), by means of the following attributes:
  - `userid`: it is an attribute of the `user` element specifying the user ID (e.g., the login name with which the user connects to Author-$\mathscr{X}$), for identity-based access control.
  - `targetCredType`: it is an attribute of the `credential` element specifying the name of subject credential(s) to which the security policy applies, for credential-based access control.
  - `credExpr`: it is an optional attribute of the `credential` element specifying the credential

```
<!DOCTYPE policyBase[
<!ELEMENT policyBase (policySpec)*>
<!ELEMENT policySpec (subject, object, accessModes)>
<!ELEMENT subject (user* | credential)>
<!ELEMENT object EMPTY >
<!ELEMENT accessModes EMPTY>
<!ELEMENT user EMPTY>
<!ELEMENT credential EMPTY>
<!ATTLIST user userid CDATA #REQUIRED>
<!ATTLIST credential targetCredType CDATA #REQUIRED credExpr CDATA #IMPLIED>
<!ATTLIST object target CDATA #REQUIRED path CDATA #REQUIRED>
<!ATTLIST accessModes priv (READ | NAVIGATE | APPEND | WRITE) #REQUIRED
          type (GRANT | DENY) #REQUIRED
          prop (NO_PROP | ONE_LEVEL | CASCADE) #REQUIRED> ]>
```

Fig. 6. Policy base DTD.

expression to be evaluated on the subject credentials specified in the `targetCredType`. When specified, such an attribute restricts the set of users of `targetCredType` to whom the policy applies, based on the conditions specified in the credential expression. When such an attribute is not specified, the security policy applies to all users qualified by subject credentials specified in `targetCredType`.

- `object`, identifying the protection objects to which the policy refers, by means of the following attributes:
  - `target`: it stores the file name of the XML document/DTD to which the policy refers.
  - `path`: it is an Xpath expression on the `target` selecting the specific protection object(s) to which the policy applies. [1]
- `accessModes` contains information about the type of privilege, the policy type, and the propagation options, by means of the following attributes:
  - `priv`: it stores the policy privilege and it is a value in the set {READ, NAVIGATE, APPEND, WRITE}; where: READ and NAVIGATE are browsing privileges and allow users to read the information in an element and to navigate through its links, respectively; WRITE and APPEND are authoring privileges and allow users to modify (or delete) the content of an element and to append new information in an element, respectively.
  - `type`: it stores the policy type and it is a value in the set {GRANT, DENY}. `type` = GRANT denotes a positive policy, that is, a policy specifying a permission, whereas `type` = DENY denotes a negative policy, that is, a policy specifying a denial.
  - `prop`: it stores the propagation option of the policy and it is a value in the set {NO_PROP, ONE_LEVEL, CASCADE}. The CASCADE option allows the propagation of the policy to all the direct and indirect subelements of the element(s) specified in the policy; whereas, the ONE_LEVEL option allows the propagation to only the direct subelements of the element(s) specified in the policy. Finally, specifying the NO_PROP option no policy propagation occurs.

[1] When the `target` is a DTD, the Xpath expression is specified on one of its valid documents.

```
<policyBase>
 <policySpec>
  <subject><credential targetCredType="ACMmember"/></subject>
  <object target="SigmodRecord.xml" path="/issues"/>
  <accessModes priv="READ" type="GRANT" prop="CASCADE"/>
 </policySpec>
 <policySpec>
  <subject><credential targetCredType="noACMmember"/></subject>
  <object target="SigmodRecord.xml" path="/issues"/>
  <accessModes priv="READ" type="GRANT" prop="CASCADE"/>
 <policySpec>
  <subject><credential targetCredType="noACMmember"/></subject>
  <object target="SigmodRecord.xml"
       path="/issues/issuesTuple/articles/articlesTuple/abstract"/>
  <accessModes priv="READ" type="DENY" prop="NO_PROP"/>
 </policySpec>
 <policySpec>
  <subject><user userid="bob@someuniversity.edu"/></subject>
  <object target="SigmodRecord.xml"
       path="/issues/issuesTuple/articles/articlesTuple[@id='WB99']"/>
  <accessModes priv="READ" type="GRANT" prop="CASCADE"/>
 </policySpec>
</policyBase>
```

Fig. 7. An example of policy base.

Example 1. Suppose that users with `ACMmember` credentials are authorized to see everything in the Sigmod Record XML source, and that users with `noACMmember` credentials must be authorized to read all information about issues contained in the Sigmod Record XML source, except articles' `abstract`. The first requirement is enforced by specifying one `READ` policy for `ACMmember` on the whole source (`SigmodRecord.xml`) with `CASCADE` propagation. The second requirement is fulfilled by specifying two policies for `noACMmember`: the first one is a positive policy for the `READ` privilege with `CASCADE` propagation on the `issues` elements, and the second one is a negative `READ` policy on the `abstract` subelement of `issues`.

Suppose now that user `Bob Smith (bob@someuniversity.edu)` must be authorized to read all the information about the article identified by `WB99`. The SO can enforce this requirement by specifying an identity-based positive policy for the `READ` privilege with `CASCADE` propagation on this article. The policy base containing all these security policies is shown in Fig. 7.

3. Protection of XML sources with Author-$\mathcal{X}$

The $\mathcal{X}$-*access* component of Author-$\mathcal{X}$ enforces access control on the XML source. Different subjects can have different views of the same XML document, depending on their credentials and the policies they possess on the involved document or associated DTD. Upon an access request,

X-access returns to the requesting user only the portion(s) of the required XML document(s) for which he/she holds a proper (possibly propagated) authorization, according to the security policies specified for the XML source. In the following, we first present the strategy we have developed for enforcing access control, we then present the access control algorithm.

3.1. Access control strategy

Subjects request access to documents under two different modes: *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, he/she requests an authoring access when a modification of the document is required. Access can also be requested against a specific portion (or portions) of a document. Thus, an access request `r` is represented as a tuple $\mathtt{r} = \langle \mathtt{user}, \mathtt{target}, \mathtt{path}, \mathtt{acc\_mode} \rangle$, where `user` is the identifier of 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 [9]) which eventually selects specific portions of the requested document, and $\mathtt{acc\_mode} \in \{\mathtt{browsing}, \mathtt{authoring}\}$ specifies whether a browsing or authoring access is requested.

Upon issuing an access request `r`, *X-access* checks the identity and the credentials associated with `user` in the credential base. Moreover, based on subject credentials, *X-access* checks which policies (both positive and negative) `user` has on the target document. Based on such policies, `user` can receive a *view* of the requested document that contains only those portions he/she is entitled to access according to the security policies stored into the policy base. In generating such view, it must be taken into account that the possibility of specifying both positive and negative policies introduces potential conflicts among policies, in that two policies can be specified for the same subject and protection object but with different signs. These conflicting policies can be either explicit or derived through propagation options. In Author-*X* the simultaneous presence of conflicting policies is not considered an inconsistency; rather a conflict resolution policy has been designed, which is based on the *Strongest Policy Principle*. The Strongest Policy Principle of Author-*X* states that:

- policies specified at the document level prevail over policies specified at the DTD level;
- policies specified at a given level in the DTD/document hierarchy prevail over policies specified at higher levels;
- when conflicts are not solved by the above rules, we consider as prevailing negative policies.

Fig. 8 summarizes the conflict resolution policy, where `prop(±)` denotes the sign of a propagated policies.

Example 2. Consider the `READ` policies for users with `noACMmember` credentials in the policy base presented in Fig. 7. The first policy is a positive policy with `CASCADE` propagation specified at the source level on the `issues` elements. The second one is a negative policy with `NO_PROP` propagation specified on all `abstract` elements. These two policies originate a conflict on `abstract` elements. Based on the Strongest Policy Principle, the second policy prevails over the first one, because it is the most specific policy for the `abstract` element.

	+	-	prop(+)	prop(-)
+	+	-	+	+
-	-	-	-	-
prop(+)	+	-	+	+/-*
prop(-)	+	-	+/-*	-

* the most specific propagated policy is taken

Fig. 8. Conflict resolution policy.

Thus, the view returned to `user` contains all the portions of the target document on which a positive policy has been defined for `user`, and which is not overwritten by a negative conflicting policy. 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 policies are found for the requested document, or all of them are overwritten by negative policies, the access is denied.

To enforce access control, two alternative strategies can be adopted:

(1) The query contained in the access control request is analyzed to verify whether `user` has the right(s) to partially or totally execute it against the target document. Based on this analysis, a view of the document (possibly empty) is returned to the user.
(2) All the elements and/or attributes for which the `user` does not have an appropriate authorization according to the security policies in the policy base are removed from the target document, before executing the query contained in the access request. The query is then applied to such pruned version and the result is returned to `user`.

Author-$\mathcal{X}$ adopts the second strategy (graphically depicted in Fig. 9). The main reason is that enforcing the first strategy makes the access control mechanism dependent from the query language (since it requires an analysis of the query to verify whether the user is or is not authorized to

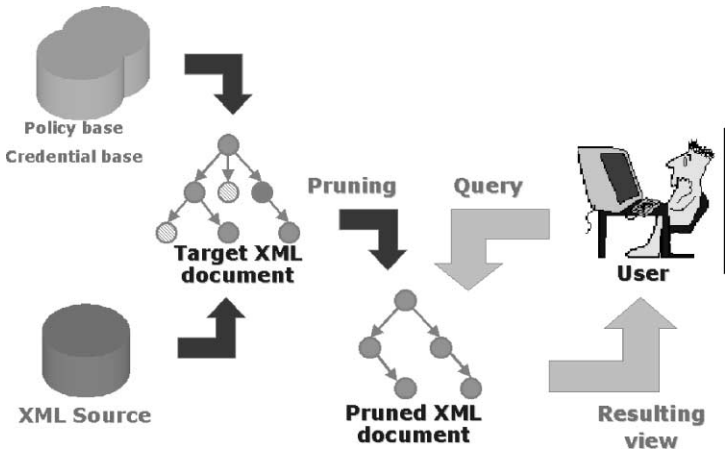


Fig. 9. Access control strategy.

Algorithm 1 Access Control Algorithm

INPUT: 1) An access request r = ⟨user, target, path, acc_modes⟩
2) The policy base (PB)

OUTPUT: 1) V_{user}(target), if the view is not empty,
2) ACCESS DENIED, otherwise

METHOD:

(1) **If** target has a DTD dtd: PolicyOnDTD = *GetBrowsingPolicies*(user,dtd,PB)
(2) PolicyOnDoc = *GetBrowsingPolicies*(user,target,PB)
(3) Let Policy be the concatenation of PolicyOnDTD and PolicyOnDoc
(4) **If** Policy is empty: **return** ACCESS DENIED
(5) structTarget = *ExtractStructure*(target)
(6) **For** each policy P in Policy: *Apply*(P,structTarget)
(7) prunedTarget = *Prune*(target,structTarget)
(8) **If** path is not null: V_{user}(target) = Evaluate(path, prunedTarget)
else V_{user}(target) = prunedTarget
(9) **If** V_{user}(target) is empty: **return** ACCESS DENIED
else: **return** V_{user}(target)

Fig. 10. An algorithm for browsing access request.

its execution). XQL, the query language provided by eXcelon, is not a standard. This means that any modification in the XQL syntax would require a modification of the access control module. Thus, we prefer to adopt the second strategy that, although less efficient than the first one, allows us to develop an access control module independent from the adopted query language.

To speed up access control, the policy base is maintained ordered, according to the priority relation among policies imposed by the conflict resolution policy. Less specific policies are stored at the top of the file, whereas more specific policies are stored towards the bottom. When two policies with the same subject, protection object(s), and privilege but different sign are entered, the positive policy is stored before the negative one. Sorting is executed each time a new policy is entered into the policy base. Thus, policies are analyzed according to increasing order of importance by performing a sequential scan of the file. As we will see in the next section, this sorting makes the access control mechanism more efficient.

3.2. Access control implementation

The algorithm for enforcing access control is reported in Fig. 10 for browsing access requests. A similar algorithm has been developed for authoring requests.

The algorithm takes as input an access request and returns a view of the target document containing only those portions of the document for which the user requesting the access has an appropriate authorization ($V_{\texttt{user}}(\texttt{target})$). If the view is empty, the access in denied. The strategy to enforce access control is based on the following steps. First a query is executed on the policy base to extract all the browsing policies specified for the `target` document (including those possibly specified on the associated DTD) which apply to `user` (either directly, because they contain his/her ID, or implicitly, because they contain a credential expression verified by one of the `user` credentials). Such policies are put into file `Policy` (step 3). This file is ordered according to the priority relation among policies implied by our strongest policy principle. [2] If

[2] We recall that policies in the policy base are kept ordered in increasing priority order.

```
Function Apply(Policy P, XMLdocument D)
{  Obj = findObjs(P.path,D)
   For each o ∈ Obj: ApplyRic(P.type,P.prop,o) }

Function ApplyRic(type tp, propagation opt, node o)
{  If o is an element {
       o.policyType = tp
       For each attribute a of o: a = tp
       For each subelement q of o:
       {  If (opt = ONE_LEVEL): ApplyRic(tp,NO_PROP,q)
          If (opt = CASCADE): ApplyRic(tp,CASCADE,q) }
   } else: o = tp
}
```

Fig. 11. Function *Apply*.

Policy is empty the access is denied (step 4). Otherwise, to speed up policy checking, a new XML document (i.e., structTarget) having the same structure of the target document is built (step 5). Such document is obtained by replacing each attribute value in the target document with a null value. Attribute values are then used to record the type of policy (that is, negative vs. positive) that applies to such attributes. structTarget also contains an additional attribute, called policyType, for each element in the target document, which is used to keep track of the type of policy which applies to the corresponding element in the target document. Then, the algorithm sequentially scans Policy and for each policy in the file, it identifies the elements/attributes/links to which it applies, based on the protection object specification in the policy and on its propagation option. If the policy is positive, then the value of each attribute/link to which it applies is put equal to GRANT, otherwise the value is set equal to DENY (eventually overwriting its current value). If the policy applies to a whole element then the corresponding policyType attribute is set equal to GRANT or DENY, depending on the policy sign. The sign specified on an element is then assigned recursively to all (or only the direct) subelements and relative attributes, according to the propagation option. The above operations are performed by function *Apply*, reported in Fig. 11.

The view the user has on the target document is then built (step 7), by pruning from the target all the attributes having a null or DENY value in structTarget, and all the elements having an analogous value for the policyType attributes. This view is built by function *Prune*.

If the request is for the whole document, the pruned document (prunedTarget) is returned to the user submitting the request, if it is not empty (otherwise the access is denied). By contrast, if the request is for selected portion(s) of the target document, the path contained in the access request is evaluated against prunedTarget (step 8). If this evaluation results in a non empty document, such document is returned to the user, otherwise the access is denied (step 9).

Example 3. Suppose John, a noACMmember user, wants to get the article identified by WB99 by submitting the following access request:

⟨john@someuniversity.edu, SigmodRecord.xml, /issues/issuesTuple /articles/articlesTuple[@id = 'WB99'], browsing⟩

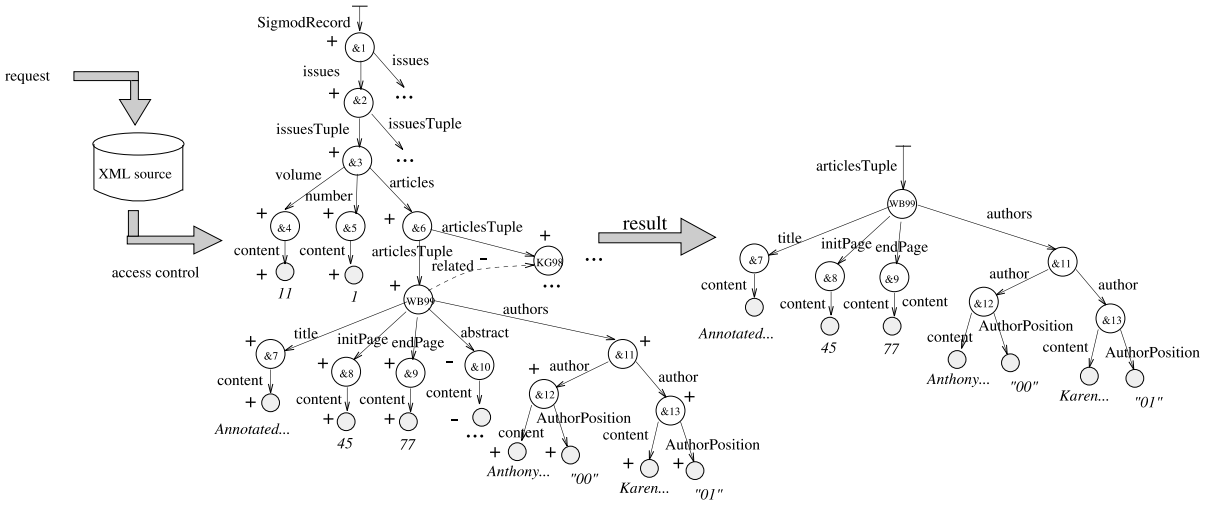


Fig. 12. The access control process.

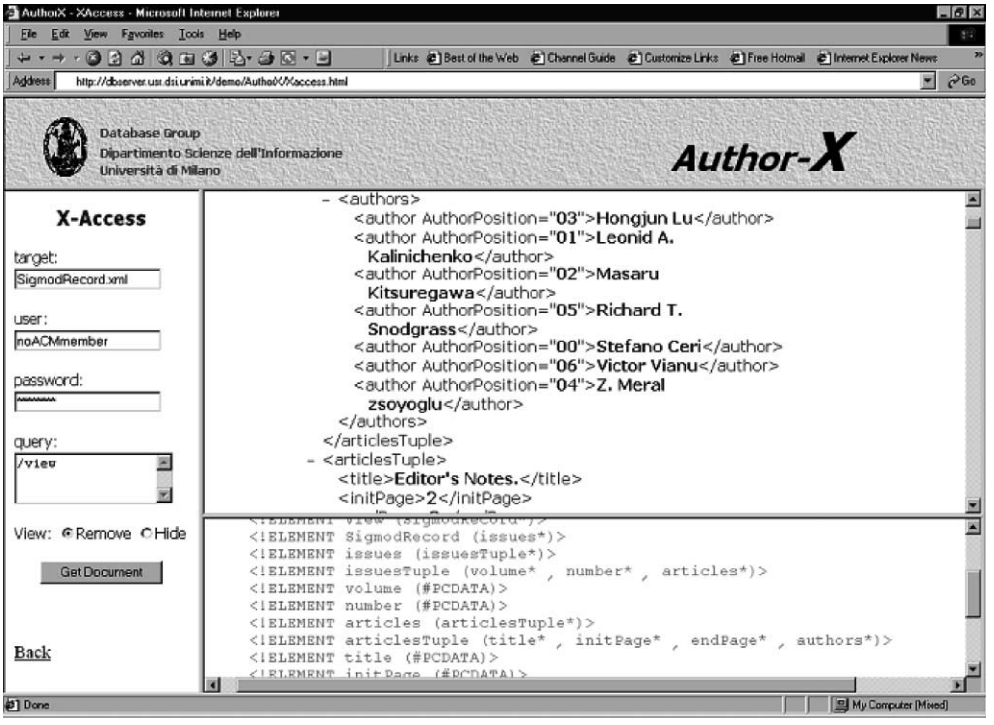


Fig. 13. Access request submission in Author-$\mathcal{X}$.

Fig. 12 shows the access control process. In the figure, we use symbol "–" instead of `DENY`, and "+" instead of `GRANT`. Author-$\mathcal{X}$ extracts from the policy base the browsing policies which apply

to noACMmember 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 policy applies to the corresponding attribute/element, or with symbol "+", if a positive policy applies to the corresponding attribute/element. The view to be returned is obtained by dropping from the graph nodes with a label different from "+", and by extracting from the resulting graph the elements/attributes identified by the path: /issues/issuesTuple/articles/articlesTuple [@id = 'WB99'] contained in the access request. As a result, a view of article WB99 is returned to the requesting user (shown on the right-hand side of Fig. 12) that does not contain the abstract element, since a negative policy is specified on this element for user with a noACMmember credential.

Fig. 13 shows the graphical interface provided by Author-$\mathcal{X}$ for access request submission. The left-hand area of the figure shows how the user can submit his/her request, whereas the right-hand area shows the corresponding resulting view and its associated DTD.

4. Security administration of XML sources with Author-$\mathcal{X}$

Author-$\mathcal{X}$ provides a user-friendly security administration environment for managing and monitoring the *$\mathcal{X}$-bases*. The overall architecture of the *$\mathcal{X}$-Admin* tool of Author-$\mathcal{X}$ is depicted in Fig. 2. In particular, *$\mathcal{X}$-Admin* provides a set of basic viewer facilities to work, in a graphical way, on documents, DTDs, policies, credential-types and credentials as well as on policy propagation and conflicts. Moreover, on top of such basic facilities, it provides two security administration components: the *Policy Manager*, providing functionalities for policy specification, validation, and maintenance, and the *Credential Manager*, providing functionalities for the management of subject credentials and credential-types. In what follows, we first outline the basic facilities of *$\mathcal{X}$-Admin* and then the security administration functionalities for credential and policy management.

4.1. Basic administration facilities

Basic viewer facilities provided by *$\mathcal{X}$-Admin* are summarized in Table 1. The *Document/DTD viewer* provides a graphical notation similar to the one adopted by conventional XML editing and parsing tools [8,17] for displaying a target document/DTD *d*. The *Policy viewer* displays the subjects and the XML documents related to a given policy. The *propagation viewer* and *conflict viewer* work on top of the graphical view of *d*, by displaying propagated policies and conflicts, respectively. The *propagation viewer* displays all propagated policies derived from the explicit policies defined for *d*, due to both explicit and implicit propagation principles. In particular, the propagation viewer adopts different graphical styles for protection object representation to highlight the type of policy (i.e., explicit vs. propagated, positive vs. negative) holding on each protection object of *d*. The *conflict viewer* displays all policy conflicts arising among security policies defined for *d*. Moreover, for each conflict, the viewer shows the conflict resolution choice adopted by default, automatically determined according to the Strongest Policy Principle (see Section 4). Besides a graphical display of the structure of credential-types and subject credentials (see Fig. 14), the *credential viewer* provides a graphical editor environment to support a form-based

Table 1
Basic facilities of *X-Admin*

Facility	Target	Functionality
Doc./DTD viewer	Doc./DTD (*d*)	• To provide a graphical view of the hierarchical structure of *d*
Policy viewer	Policy (*p*)	• To display the list of all the subjects to whom *p* applies • To display the list of all protection objects on which *p* holds
Propagation viewer	Doc./DTD (*d*)	• To provide a graphical view of propagation of explicit policies defined for *d*
Conflict viewer	Doc./DTD (*d*)	• To provide a graphical view of policy conflicts over *d*
Credential viewer	Cred.-type/ sub. cred. (*c*)	• To provide a graphical view of the structure of credential-types • To provide a graphical view of the structure of subject credentials • To provide a graphical editor environment for specifying credential expressions

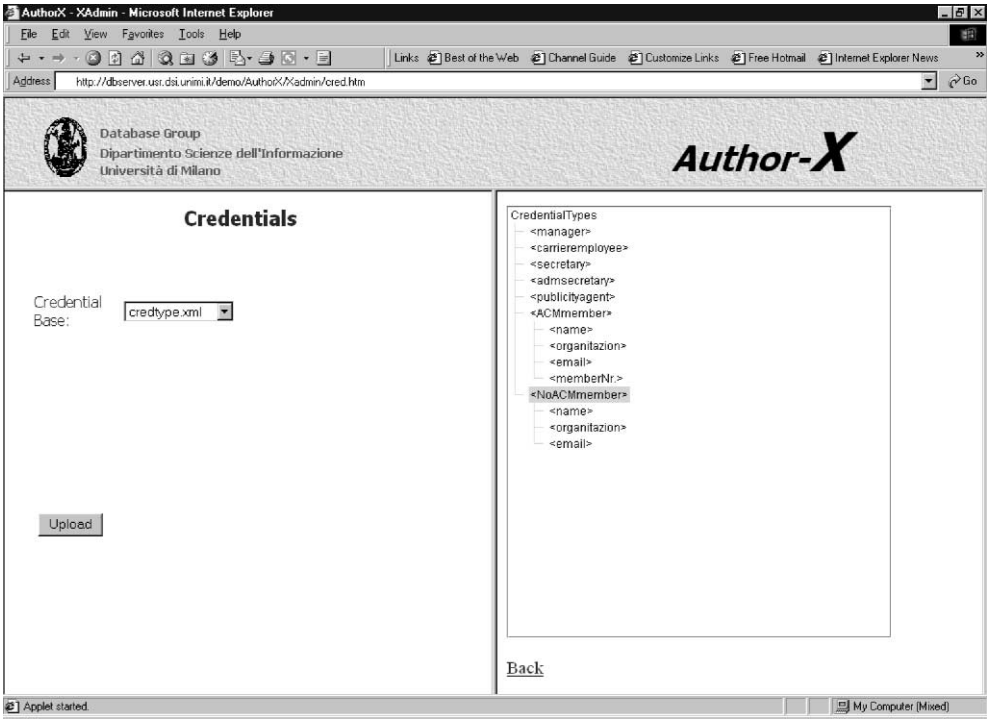


Fig. 14. Author-*X* graphical interface for credential base management.

specification of credential expressions. Such expressions denote the subjects to whom a security policy applies through combinations of conditions on defined credential-types. Using this editor, elements of the expression are selected from interactive menus listing credential-types, credential properties, constant values, as well as comparison and logic operators. Once the desired expression has been specified, the viewer editor generates the corresponding Xpath-based syntax of the credential expression. Basic facilities listed in Table 1 can be interactively invoked by the SO within an *X-Admin* working session, to perform security administration activities, as described

below. Propagation and conflict viewer facilities can also be invoked on the whole policy base, to visualize propagation and conflicts for all security policies actually stored in the policy base on a per document/DTD basis. Since in Author-$\mathscr{X}$ both documents and security-related information are specified in XML syntax, viewer facilities internally work on DOM [16] representation and exploit the XQL language [9].

4.2. Policy specification

Author-$\mathscr{X}$ allows the specification of a variety of security policies for XML documents, that differ for the kind of subject, protection object, access mode, and the granularity level at which the policy is specified. For example, the SO can specify policies regulating the access to XML documents based on subject credentials or on conventional user IDs. It is also possible to specify policies at varying granularity levels for both DTDs and XML documents, to access a set of documents, a single document, a document portion, reaching the granularity of the single element, attribute, or link. A *policy type* captures the kind of policy and is a tuple of the form:

$$\langle Who, What, How, KindOfAccess, KindOfGrant, KindOfProp\rangle$$

where:

- $Who \in \{UserID,\ Credential\}$, denotes how subjects are qualified in the policy. Author-$\mathscr{X}$ supports both ID-based and credential-based qualifications.
- $What \in \{XMLdoc,\ XMLDTD,\ setofXMLdocs,\ setofDTDs\}$, denotes the kind of protection objects to which the policy applies. Author-$\mathscr{X}$ supports policies either at the document level or DTD level as well as policies which apply to set of documents/DTDs.
- $How \in \{Whole,\ Portion,\ Content\}$, denotes the kind of granularity at which the policy applies for the protection object specified in the *What* field. Author-$\mathscr{X}$ supports a wide range of policies, ranging from policies that apply to a whole document/DTD to fine-grained policies that apply to selected portions of a document/DTD, as well as to policies which apply to (portions of) documents/DTDs based on their contents.
- $KindOfAccess \in \{Browsing,\ Authoring\}$, denotes the kind of access to be allowed/denied on the protection object(s) of the policy. Author-$\mathscr{X}$ supports both browsing and authoring access modes, to cover all access needs related to XML documents.
- $KindOfGrant \in \{Permission,\ Denial\}$, denotes the kind of authorization to be granted, either positive authorization stating a permission or a negative one, stating a denial.
- $KindOfProp \in \{Recursive,\ Limited,\ none\}$, denotes the kind of propagation to be enforced for the policy at hand. Author-$\mathscr{X}$ supports a cascading propagation, by which the policy recursively applies to all subobjects of the target protection object(s), a propagation limited to the direct subobjects of the target, as well as no propagation at all.

Example 4. Suppose that the SO is interested in specifying a positive policy for credential-based browsing of an XML document, which propagates to all its subobjects, based on the content of the document. In this case the SO can specify the following policy type:

$$\langle Credential, XMLdoc, Content, Browsing, Permission, Recursive\rangle$$

Suppose now that the SO is interested in specifying a negative policy for identity-based authoring of a whole DTD, which propagates only to the direct subobjects. In this case the SO can specify the following policy type:

$\langle UserID, XMLDTD, Whole, Authoring, Denial, Limited \rangle$

Policy specification process (depicted in Fig. 15) is performed in two stages, by choosing first an appropriate policy type and then by specifying one or more security policies for that type, according to predefined "policy specification forms", designed to ease the configuration of policy types and associated $\mathcal{X}$-Sec policies.

The SO works on the graphical representation of a target document/DTD d and selects the protection object(s) on which he/she wants to specify an explicit policy (see Fig. 16, showing a policy specification session for a purchase order document). Upon selection of a protection object, a policy specification form is supplied to the SO (see Fig. 16) containing a field for each element of the $\mathcal{X}$-Sec security policy (i.e., `subject`, `protection object`, `access mode`, `sign`, and `propagation option`). For example, to specify a credential-based security policy, the SO first selects the value *Credential* for the Subject field, and then specifies the appropriate credential expression using the credential viewer editor. The protection object field of the form is automatically filled in by *$\mathcal{X}$-Admin*, with the Xpath expression denoting the path of the selected protection object within the graphical representation of d. The access mode, sign, and propagation option fields of the form have an associated multiple-choice menu listing the set of admissible values for the field. To simplify the specification activity, default values are provided by the tool

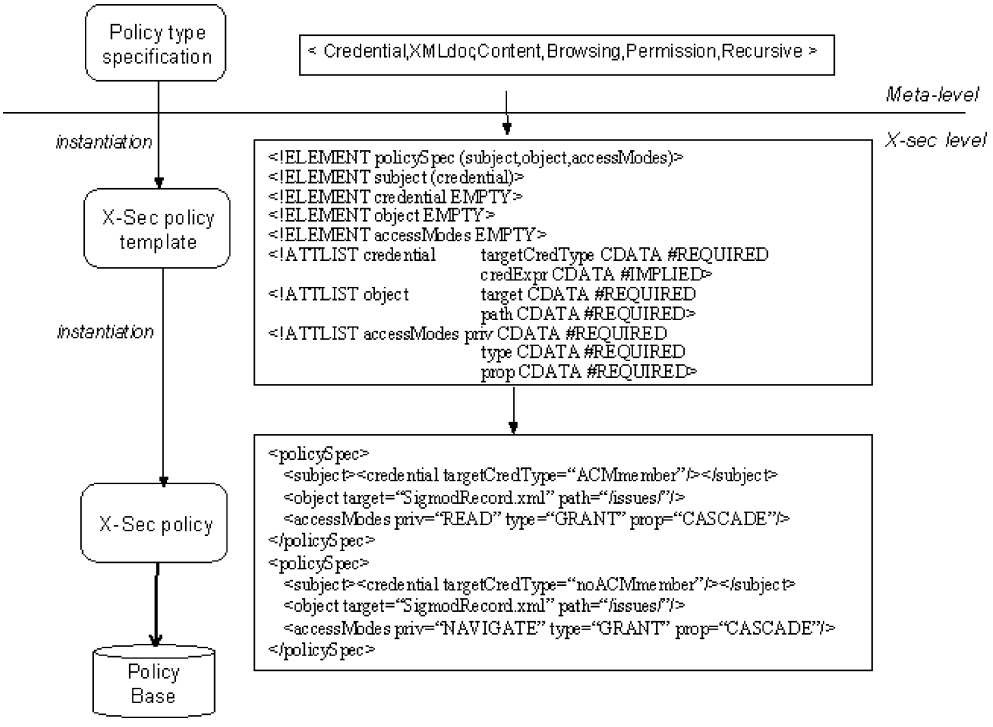


Fig. 15. Policy specification process.

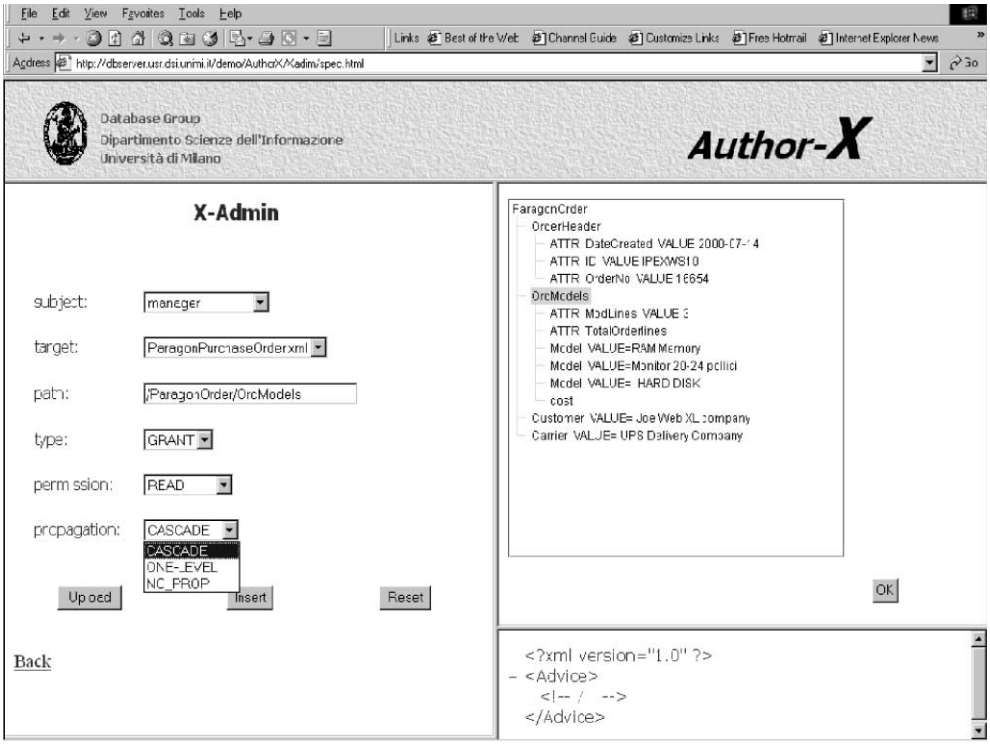


Fig. 16. Example of policy specification in Author-$\mathcal{X}$.

for these three fields (i.e., `read`, `grant`, and `cascade`, respectively), that are changed by the SO when necessary. As the policy specification process proceeds, the SO can invoke the propagation viewer to display the effects of policy propagation. In this way, the SO can interactively analyze the scope of a security policy under definition, and evaluate when the desired level of protection is reached and security requirements for d have been satisfied.

Once the policy specification form has been filled in and the policy validated, *$\mathcal{X}$-Admin* generates the $\mathcal{X}$-Sec specification of the security policy and stores it in the policy base.

Note that the policy specification form is used also for customizing propagated security policies and for updating security policies already defined and stored in the policy base.

$\mathcal{X}$-Admin offers a flexible policy specification environment, by providing tool support for protecting both documents in the XML source (in-site documents) and XML documents coming from outside (external documents) [1]. Once defined, security policies can be retrieved and visualized using the policy maintenance facilities. For example, in Fig. 17 we show the security policies specified for the `noACMmember` subject credential of the running example.

4.3. Policy validation

This facility supports interactive validation of security policies for a target document/DTD d. Two kinds of validation are supported in *$\mathcal{X}$-Admin*: validation against integrity constraints and validation with respect to policy conflicts.

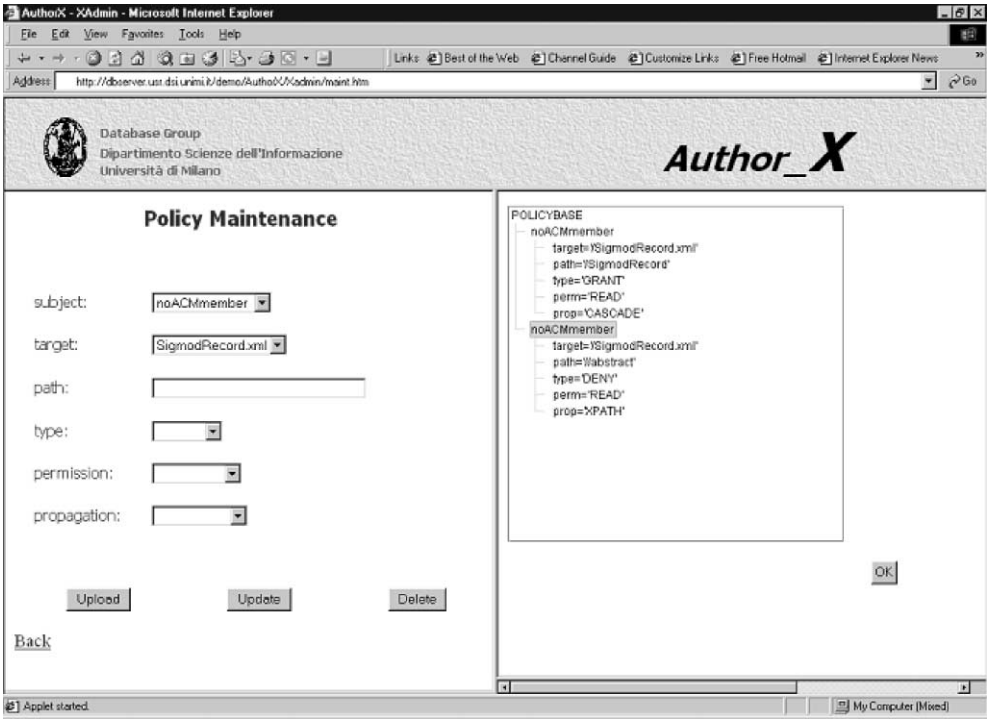


Fig. 17. Example of policy visualization.

With the *validation against integrity constraints*, security policies on a target d are validated against a set of integrity constraints enforced by $\mathcal{X}$-*Admin* to avoid combination of security policies leading to authorization for meaningless portions of d or to undesired situations (e.g., inference of protected information). These situations can occur due to the fine-level protection granularity enforced by Author-$\mathcal{X}$ combined with the possibility of defining positive and negative policies on protection objects. $\mathcal{X}$-*Admin* performs policy validation against all defined integrity constraints and notifies the SO that a violation has occurred, by submitting to the SO the corresponding policy combination for evaluation. For example, we have a constraint to check situations where a positive policy has been specified for a part of (or all) attributes of an element and a negative policy has been defined for the element. The tool notifies this situation to the SO, since this policy combination leads to subjects authorized for accessing attributes without a container element, and this could result in a meaningless document. As another example, the tool checks for combinations of a positive policy for an element and negative policies for all its subelements and attributes. A warning is sent to the SO for this policy combination, since the authorized subject could infer the existence of other information kept hidden to him. It has to be noted that the SO can decide to keep the policy combination however, although violating an integrity constraint, if he/she judges that this policy combination is adequate for the target document/DTD under consideration.

Validation with respect to policy conflicts is performed to detect and analyze possible conflicts arising between policies defined for d. Conflict validation exploits the conflict viewer facility for displaying detected policy conflicts, together with the default conflict resolution policy. Default

resolution policies proposed by *X-Admin* can be overwritten by the SO, if a policy different from the default one is more suitable for resolving a certain conflict.

4.4. Policy maintenance

This facility of *X-Admin* supports maintenance operations on the policy base, namely policy update and policy revocation.

Policy update operations allow the SO to modify the security policies of a secured document/DTD, in order to extend and/or restrict them. Policy update operations that can be performed in *X-Admin* regard: (i) the definition of a new explicit policy for documents/DTDs already secured; (ii) the modification of an existing policy, to vary its sign (from positive to negative or vice versa), or its propagation option (to reduce or augment the deep of propagation), or its subjects (to extend and/or restrict the set of subjects authorized by the policy), or its protection objects; (iii) the deletion of a security policy. All update operations are performed on the graphical view of the target document/DTD using policy specification forms and the credential viewer editor facility, for updates to policy subjects. *X-Admin* propagates all the updates to the policy base.

Policy revocation can be required for maintenance of the policy base over time. Policy revocation can be performed either at the credential-type level or at the credential level. In the former case, a policy is revoked from all the subjects of a given credential-type. In the latter case, a security policy is revoked only from specific subjects of a credential-type. Policy revocation is performed with the help of the credential viewer and associated editor, to specify necessary modifications to the subject field of the security policy to be revoked.

5. Concluding remarks

The Web community regards XML as the most important standardization tool for information exchange and interoperability. In this paper, we have presented the protection and administration facilities of Author-*X*, a Java-based system for access control to XML sources. Author-*X* supports positive and negative policies for browsing and authoring privileges with a controlled propagation. Moreover, we have addressed the administration of security policies for XML document sources. We argue that providing XML administration facilities is a key requirement to make XML access control mechanisms effective. The current prototype of Author-*X* is built on top of the eXcelon [7] XML server and implements both the core functionalities of access control and policy base management and the core specification and maintenance functionalities described in the paper.

Future work will be devoted to the extension of the administration tool with pattern-based specification facilities for both policies and credentials. Additionally, we will look at incorporating Author-*X* within existing Web-based enterprise information systems, by focusing on performance issues both at the theoretical and at the experimental level.

Acknowledgements

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

References

[1] E. Bertino, S. Castano, E. Ferrari, Author-$\mathscr{X}$: a comprehensive system for securing XML documents, IEEE Internet Computing 5 (3) (2001) 21–31.

[2] E. Bertino, S. Castano, E. Ferrari, On specifying security policies for Web documents with an XML-based language, in: Proc. of SACMAT'2001, ACM Symposium on Access Control Models and Technologies, Fairfax, VA, 2001.

[3] E. Bertino, S. Castano, E. Ferrari, Securing XML documents: the Author-X project demonstration, in: Proc. of the SIGMOD 2001 Conference, Santa Barbara, CA, 2001.

[4] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati, Securing XML documents, in: Proc. of Int'l Conference on Extending Database Technology, Konstanz, Germany, 2000.

[5] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Suciu, A query language for XML, in: Proc. Int'l Conference on World Wide Web, Toronto, Canada, 1999.

[6] C. Geuer Pollmann, The XML security page. Available from <http://www.nue.et-inf.uni-siegen.de/~geuer-pollmann/>.

[7] Object Design Inc., An XML data server for building enterprise Web applications, 1998, White paper. Available from <http://www.odi.com/excelon>.

[8] PROJECTX Parser, Sun. Available from <http://java.sun.com/products/xml>.

[9] J. Robbie, XQL'99 Proposal, 1999. Available from <http://metalab.unc.edu/xql/>.

[10] P. Samarati, E. Bertino, S. Jajodia, An authorization model for a distributed hypertext system, IEEE TKDE 8 (4) (1996) 555–562.

[11] Sigmod Record XML Database. Available from <http://www.dia.uniroma3.it/Araneus/Sigmod/>.

[12] W. Stallings, Network Security Essentials: Applications and Standards, Prentice Hall, 2000.

[13] M. Winslett, N. Ching, V. Jones, I. Slepchin, Using digital credentials on the World Wide Web, Journal of Computer Security 7 (1997).

[14] W3C, XML Path Language (Xpath), 1999. Available from <http://www.w3.org/TR/xpath>.

[15] W3C, Extensible Markup Language, 1998. Available from <http://www.w3.org/XML>.

[16] W3C, Document Object Model, 1998. Available from <http://www.w3.org/DOM/>.

[17] XERCES Parser, Apache. Available from <http://www.xml.apache.org>.

Elisa Bertino is professor of database systems in the Department of Computer Science of the University of Milan where she is currently the chair of the Department. She has also been on the faculty in the Department of Computer and Information Science of the University of Genova, Italy. Until 1990, she was a researcher for the Italian National Research Council in Pisa, Italy, where she headed the Object-Oriented Systems Group. She has been a visiting researcher at the IBM Research Laboratory (now Almaden) in San Jose, at the Microelectronics and Computer Technology Corporation in Austin, Texas, at George Mason University, at Rutgers University, at Purdue University, at Telcordia Technologies. Her main research interests include database security, object-oriented databases, distributed databases, deductive databases, multimedia databases, interoperability of heterogeneous systems, integration of artificial intelligence and database techniques. In those areas, Prof. Bertino has published more than 200 papers in all major refereed journals, and in proceedings of international conferences and symposia. She is a co-author of the books "Object-Oriented Database Systems—Concepts and Architectures" 1993 (Addison-Wesley International Publ.), "Indexing Techniques for Advanced Database Systems" 1997 (Kluwer Academic Publishers), and "Intelligent Database Systems" 2001 (Addison-Wesley International Publ.). She is member of the advisory board of the IEEE Transactions on Knowledge and Data Engineering and a member of the editorial boards of several scientific journals, including ACM Transactions on Information and System Security, IEEE Internet Computing, the Very Large Database Systems (VLDB) Journal, the Parallel and Distributed Database Journal, the Journal of Computer Security, Data & Knowledge Engineering, the International Journal of Information Technology, the International Journal of Cooperative Information Systems, Science of Computer Programming. She has been consultant to several Italian companies on data management systems and applications and has given several courses to industries. She is involved in several projects sponsored by the EEC. Elisa Bertino is a Fellow member of IEEE and a member of ACM and has been been named a Golden Core Member for her service to the IEEE Computer Society. She has served as Program Committee members of several international conferences, such as ACM SIGMOD, VLDB, ACM OOPSLA, as Program Co-Chair of the 1998 IEEE International Conference on Data Engineering (ICDE), as program chair of 2000 European Conference on Object-Oriented Programming (ECOOP 2000), and as program chair of the 7th ACM Symposium of Access Control Models and Technologies (SACMAT 2002).

Silvana Castano is full professor of Computer Science at the University of Milano, where previously she has been associate professor (1998–2001) and assistant professor (1993–1998), respectively. She received the Ph.D. degree in Computer Science from Politecnico di Milano, in 1993. Her main research interests include database and semistructured data integration, database and XML security, Web-based information systems, process analysis and reengineering, workflow design, reuse of conceptual components. On these topics, she has published several papers in refereed international journals and conferences and she has been working within several national and international research projects. She is a co-author of the book *Database Security* (Addison Wesley, 1995). She is currently chair of the AICA Working Group on Databases. She is a member of the ACM, IEEE Computer Society, and AICA.

Elena Ferrari is professor of database systems at the University of Insubria, Como Italy. She has also been on the faculty in the Department of Computer Science of the University of Milano, Italy, from 1998 to March 2001. She received a Ph.D. in computer science from the University of Milano, in 1997. She has been a visiting researcher at George Mason University, Fairfax (VA) and at Rutgers University, Newark (NJ). Her main research interests include database security, object-oriented and multimedia databases. In those areas, Prof. Ferrari has published several papers. She has also given several courses to industries on topics related to security and data management. Elena Ferrari has served as as Program Co-Chair of the first ECOOP Workshop on XML and Object Tecnology (XOT'2000) and of the first ECOOP Workshop on Object-oriented Databases.

Marco Mesiti received with honors his master's degree in computer science from the University of Genova, Italy. He is currently a Ph.D. student in Computer Science at the University of Genova and he expects to graduate by the end of this year. His main research interests include management of semi-structured data, querying and classification of XML documents, access control mechanisms for XML, database technology for telecommunication applications. He has carried out extensive teaching activity as teaching assistant at the University of Genova and the University of Milano. He has been a visiting researcher at the applied research center of Telcordia Technologies, Morristown, NJ.