Securing XML Documents*

E. Damiani[1], S. De Capitani di Vimercati[1], S. Paraboschi[2], and P. Samarati[1]

[1] Università di Milano, Dip. Scienze Informazione, 20135 Milano - Italy
edamiani@crema.unimi.it,{decapita,samarati}@dsi.unimi.it
[2] Politecnico di Milano, Dip. Elettronica e Informazione, 20133 Milano - Italy
parabosc@elet.polimi.it

Abstract. Web-based applications greatly increase information availability and ease of access, which is optimal for public information. The distribution and sharing by the Web of information that must be accessed in a selective way requires the definition and enforcement of security controls, ensuring that information will be accessible only to authorized entities. Approaches proposed to this end level, independently from the semantics of the data to be protected and for this reason result limited. The eXtensible Markup Language (XML), a markup language promoted by the World Wide Web Consortium (W3C), represents an important opportunity to solve this problem. We present an access control model to protect information distributed on the Web that, by exploiting XML's own capabilities, allows the definition and enforcement of access restrictions directly on the structure and content of XML documents. We also present a language for the specification of access restrictions that uses standard notations and concepts and briefly describe a system architecture for access control enforcement based on existing technology.

1 Introduction

An ever-increasing amount of information, both on corporate Intranets and the global Internet, is being made available in unstructured and semi-structured form. Semi-structured data sources include collections of textual documents (e.g., e-mail messages) and HTML pages managed by Web sites. While these sites are currently implemented using ad-hoc techniques, it is widely recognized that, in due time, they will have to be accessible in an integrated and uniform way to both end users and software application layers. Nevertheless, current techniques for Web information processing turn out to be rather awkward, due to HTML's inherent limitations. HTML provides no clean separation between the structure and the layout of a document. Moreover, site designers often prepare HTML pages according to the needs of a particular browser. Therefore, HTML markup has generally little to do with data semantics.

To overcome this problem, a great effort was put in place to provide *semantics-aware* markup techniques without losing the formatting and rendering capabilities of HTML. The main result of this standardization effort is the *eXtensible*

* This work was supported in part by the INTERDATA and DATA-X - MURST 40% projects and by the Fifth (EC) Framework Programme under the FASTER project

Markup Language (XML) [3], a markup meta-language recently standardized by the World Wide Web Consortium (W3C). While HTML was defined using only a small and basic part of SGML (Standard Generalized Markup Language: ISO 8879), XML is a sophisticated subset of SGML, designed to describe data using arbitrary tags. One of the main goals of XML is to be suitable for the use on the Web, thus providing a general mechanism for enriching HTML. As its name implies, extensibility is a key feature of XML; users or applications are free to declare and use their own tags and attributes. XML focuses on the description of information structure and content as opposed to its presentation. Presentation issues are addressed by separate languages: XSL (XML Style Language) [19], which is also a W3C standard for expressing how XML-based data should be rendered; and XLink (XML Linking Language) [7], which is a specification language to define anchors and links within XML documents. For its advantages, XML is now accepted in the Web community, and available applications exploiting this standard include OFX (Open Financial Exchange) [6] to describe financial transactions, CDF (Channel Data Format) [8] for push technologies, and OSD (Open Software Distribution) [17] for software distribution on the Net.

Security is among the main concerns arising in this context. Internet is a public network, and traditionally there has been little protection against unauthorized access to sensitive information and attacks such as intrusion, eavesdropping, and forgery. Fortunately, the advancement of public-key cryptography has remedied most of the security problems in communication; in the XML area commercial products are becoming available (such as AlphaWorks' XML Security Suite [1]) providing security features such as digital signatures and element-wise encryption to transactions involving XML data. However, the design of a sophisticated access control mechanism to XML information still remains an open issue, and the need for addressing it is well recognized [15].

The objective of our work is to define and implement an authorization model for regulating access to XML documents. The rationale for our approach is to exploit XML's own capabilities, defining an XML markup for a set of *security elements* describing the protection requirements of XML documents. This security markup can be used to provide both *instance level* and *schema level* authorizations with the granularity of XML elements. Taken together with a user's identification and its associated group memberships, as well as with the support for both permissions and denials of access, our security markup allows to easily express different protection requirements with support of exceptions. The enforcement of the requirements stated by the authorizations produces a view on the documents for each requester; the view includes only the information that the requester is entitled to see. A recursive propagation algorithm is also presented, which ensures fast on-line computation of such a view on XML documents requested via an HTTP connection or a query. The proposed approach, while powerful enough to define sophisticated access to XML data, makes the design of a server-side *security processor* for XML rather straightforward; guidelines for design are also provided.

1.1 Related work

Although several projects for supporting authorization-based access control in the Web have recently been carried out, authorizations and access control mechanisms available today are at a preliminary stage [15]. For instance, the Apache server (`www.apache.org`) allows the specification of access control lists via a configuration file (access.conf) containing the list of users, hosts (IP addresses), or host/user pairs, which must be allowed/forbidden connection to the server. Users are identified by user- and group-names and passwords, to be specified via Unix-style password files. By specifying a different configuration file for each directory, it is possible to define authorizations on a directory basis. The specification of authorizations at the level of single file (i.e., web pages) results awkward, while it is not possible to specify authorizations on portions of files. The proposal in [16] specifies authorizations at a fine granularity by considering a Dexter-like model for referencing portions of a file. However, again, no semantic context similar to that provided by XML can be supported and the model remains limited. Other approaches, such as the EIT SHTTP scheme, explicitly represent authorizations within the documents by using security-related HTML tagging. While this seems to be the right direction towards the construction of a more powerful access control mechanism, due to HTML fundamental limitations these proposals cannot take into full consideration the information structure and semantics. The development of XML represents an important opportunity to solve this problem. Proposals are under development by both industry and academia, and commercial products are becoming available which provide security features around XML. However, these approaches focus on lower level features, such as encryption and digital signatures [1], or on privacy restrictions on the dissemination of information collected by the server [14]. At the same time, the security community is proceeding towards the development of sophisticated access control models and mechanisms able to support different security requirements and multiple policies [10]. These proposals have not been conceived for semi-structured data with their flexible and volatile organization. They are often based on the use of logic languages, which are not immediately suited to the Internet context, where simplicity and easy integration with existing technology must be ensured. Our approach expresses security requirements in syntax, rather than in logic, leading to a simpler and more efficient evaluation engine that can be smoothly integrated in an environment for XML information processing. The use of authorization priorities with propagation and overriding, which is an important aspect of our proposal, may recall approaches made in the context of object-oriented databases, like [9, 13]. However, the XML data model is not object-oriented [3] and the hierarchies it considers represent part-of relationships and textual containment, which require specific techniques different from those applicable to ISA hierarchies in the object-oriented context.

1.2 Outline of the paper

The paper is organized as follows. Section 2 illustrates the basic characteristics of the XML proposal. Section 3 and 4 discuss the subjects and the objects,

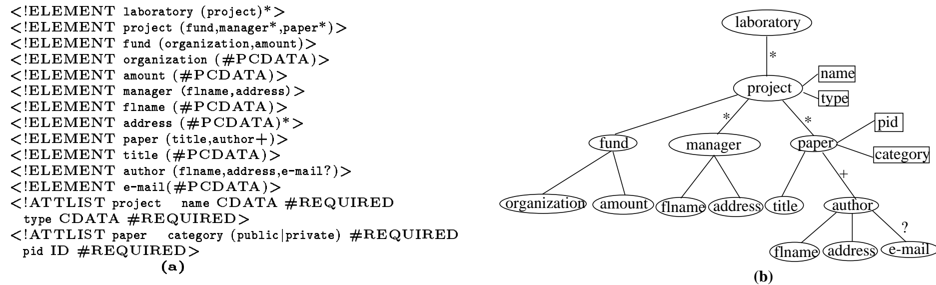

Fig. 1. An example of DTD (a) and the corresponding tree representation (b)

respectively. Section 5 presents the authorization model. Section 6 introduces the document view produced by the access control system for each requester and presents an algorithm for efficiently computing such a view. Section 7 addresses design and implementation issues and sketches the architecture of the security system. Section 8 gives concluding remarks.

2 Preliminary concepts

XML [3] is a markup language for describing semi-structured information. The XML document is composed of a sequence of nested elements, each delimited by a pair of start and end tags (e.g., `<project>` and `</project>`) or by an empty tag. XML documents can be classified into two categories: *well-formed* and *valid*. An XML document is well-formed if it obeys the syntax of XML (e.g., non-empty tags must be properly nested, each non-empty start tag must correspond to an end tag). A well-formed document is valid if it conforms to a proper *Document Type Definition* (DTD). A DTD is a file (external or included directly in the XML document) which contains a formal definition of a particular type of XML document. A DTD may include declarations for elements, attributes, entities, and notations. Elements are the most important components of an XML document. Element declarations in the DTD specify the names of elements and their content. They also describe sub-elements and their cardinality; with a notation inspired by extended BNF grammars, "$*$" indicates zero or more occurrences, "+" indicates one or more occurrences, "?" indicates zero or one occurrence, and no label indicates exactly one occurrence. Attributes represent properties of elements. Attribute declarations specify the attributes of each element, indicating their name, type, and, possibly, default value. Attributes can be marked as `required`, `implied`, or `fixed`. Attributes marked as `required` must have an explicit value for each occurrence of the elements to which they are associated. Attributes marked as `implied` are optional. Attributes marked as `fixed` have a fixed value indicated at the time of their definition. Entities are used to include text and/or binary data into a document. Notation declarations specify how to manage entities including binary data. Entities and notations are important in the description of the physical structure of an XML document, but are not

considered in this paper, where we concentrate the analysis on the XML logical description. Our authorization model can be easily extended to cover these components. Figure 1(a) illustrates an example of DTD for XML documents describing projects of a laboratory.

XML documents valid according to a DTD obey the structure defined by the DTD. Intuitively, each DTD is a *schema* and XML documents valid according to that DTD are *instances* of that schema. However, the structure specified by the DTD is not rigid; two distinct documents of the same schema may widely differ in the number and structure of elements.

DTDs and XML documents can be modeled graphically as follows. A DTD is represented as a labeled tree containing a node for each attribute and element in the DTD. There is an arc between elements and each element/attribute belonging to them, labeled with the cardinality of the relationship. Elements are represented as circles and attributes as squares. Each XML document is described by a tree with a node for each element, attribute, and value in the document, and with an arc between each element and each of its sub-elements/attributes/values and between each attribute and each of its value(s). Figure 1(b) illustrates the tree for the DTD in Figure 1(a).

In the remainder of this paper we will use the terms *tree* and *object* to denote either a DTD or an XML document. We will explicitly distinguish them when necessary.

3 Authorization subjects

The development of an access control system requires the definition of the *subjects* and *objects* against which authorizations must be specified and access control must be enforced. In this section we present the subjects; in Section 4 we describe the objects.

Usually, subjects can be referred to on the basis of their *identities* or on the *location* from which requests originate. Locations can be expressed with reference to either their numeric IP address (e.g., `150.100.30.8`) or their symbolic name (e.g., `tweety.lab.com`). Our model combines these features. Subjects requesting access are thus characterized by a triple ⟨`user-id`,`IP-address`,`sym-address`⟩, where `user-id` is the identity [1] with which the user connected to the server, and `IP-address` (`sym-address`, resp.) is the numeric (symbolic, resp.) identifier of the machine from which the user connected.

To allow the specification of authorizations applicable to sets of users and/or to sets of machines, the model also supports user *groups* and *location patterns*. A group is a set of users defined at the server. Groups do not need to be disjoint and can be nested. A location pattern is an expression identifying a set of physical

[1] We assume user identities to be local, that is, established and authenticated by the server, because this is a solution relatively easy to implement securely. Obviously, in a context where remote identities cannot be forged and can therefore be trusted by the server (using a Certification Authority, a trusted third party, or any other secure infrastructure), remote identities could be considered as well.

locations, with reference to either their symbolic or numerical identifiers. Patterns are specified by using the wild card character `*` instead of a specific name or number (or sequence of them). For instance, `151.100.*.*`, or equivalently `151.100.*`, denotes all the machines belonging to network `151.100`. Similarly, `*.mil`, `*.com`, and `*.it` denote all the machines in the Military, Company, and Italy domains, respectively. If multiple wild card characters appear in a pattern, their occurrence must be continuous (not interleaved by numbers or names). Also, consistently with the fact that specificity is left to right in IP addresses and right to left in symbolic names, wild card characters must appear always as right-most elements in IP patterns and as left-most elements in symbolic patterns. Intuitively, location patterns are to location addresses what groups are to users. Given a pair p_1 and p_2 of IP patterns (symbolic patterns, resp.), $p_1 \leq_{\mathsf{ip}} p_2$ ($p_1 \leq_{\mathsf{sn}} p_2$, resp.) only if each component of p_1 is either the wild card character or is equal to the corresponding, position wise from left to right (right to left, resp.), component of p_2.

Instead of specifying authorizations with respect to only one of either the user/group identifier or location identifier, and having the problem of how different authorizations can be combined at access request time, we allow the specification of authorizations with reference to both user/group and location. This choice provides more expressiveness (it allows to express the same requirements as the alternative and more) and provides a natural treatment for different authorizations applicable to the same request. We will elaborate on this in Section 5. Let UG be a set of user and group identifiers, IP a set of IP patterns, and SN a set of symbolic name patterns. We define the *authorization subject hierarchy* as follows.

Definition 1 (Authorization subject hierarchy). *The* authorization subject hierarchy *is a hierarchy* $\mathsf{ASH} = (\mathsf{AS}, \leq)$*, where* $\mathsf{AS} = \langle \mathsf{UG} \times \mathsf{IP} \times \mathsf{SN} \rangle$ *and* $\leq$ *is a partial order over* AS *such that* $\forall \langle ug_i, ip_i, sn_i \rangle, \langle ug_j, ip_j, sn_j \rangle \in \mathsf{AS}$, $\langle ug_i, ip_i, sn_i \rangle \leq \langle ug_j, ip_j, sn_j \rangle$*, if and only if* ug_i *is a member of* ug_j*,* $ip_i \leq_{\mathsf{ip}} ip_j$*, and* $sn_i \leq_{\mathsf{sn}} sn_j$*.*

According to the fact that requests are always submitted by a specific user (`anonymous` can also be interpreted as such) from a specific location, subjects requesting access are always minimal elements of the ASH hierarchy. Authorizations can instead be specified with reference to any of the elements of ASH. In particular, authorizations can be specified for users/groups regardless of the physical location (e.g., $\langle \texttt{Alice}, *, * \rangle$), for physical locations regardless of the user identity (e.g., $\langle \texttt{Public}, 150.100.30.8, * \rangle$), or for both (e.g., $\langle \texttt{Sam}, *, *.\texttt{lab.com} \rangle$). Intuitively, authorizations specified for subject $s_j \in \mathsf{AS}$ are applicable to all subjects s_i such that $s_i \leq s_j$.

4 Authorization objects

A set Obj of Uniform Resource Identifiers (URI) [2] denotes the resources to be protected. For XML documents, URI's can be extended with *path expressions*,

which are used to identify the elements and attributes within a document. In particular, we adopt the XPath language [20] proposed by the W3C. There are considerable advantages deriving from the adoption of a standard language. First, the syntax and semantics of the language are known by potential users and well-studied. Second, several tools are already available which can be easily reused to produce a functioning system. We keep at a simplified level the description of the constructs to express patterns in XPath, and refer to the W3C proposal [20] for the complete specification of the language.

Definition 2 (Path expression). *A* path expression *on a document tree is a sequence of element names or predefined functions separated by the character / (slash): $l_1/l_2/\ldots/l_n$. Path expressions may terminate with an attribute name as the last term of the sequence. Attribute names are syntactically distinguished preceding them with the special character* `@`.

A path expression $l_1/l_2/\ldots/l_n$ on a document tree represents all the attributes or elements named l_n that can be reached by descending the document tree along the sequence of nodes named $l_1, l_2, \ldots, l_{n-1}$. For instance, path expression `/laboratory/project` denotes the `project` elements which are children of `laboratory` element. Path expressions may start from the root of the document (if the path expression starts with a slash, it is called *absolute*) or from a predefined starting point in the document (if the path expression starts with an element name, it is called *relative*). The path expression may also contain the operators *dot*, which represents the current node; *double dot*, which represents the parent node; and *double slash*, which represents an arbitrary descending path. For instance, path expression `/laboratory//flname` retrieves all the elements `flname` descendants of the document's root `laboratory`.

Path expressions may also include functions. These functions serve various needs, like the extraction of the text contained in an element and the navigation in the document structure. The language provides a number of predefined functions, among which: `child`, that permits to extract the children of a node; `descendant`, that returns the descendants of a node; and `ancestor`, that returns the ancestors of a node. The name of a function and its arguments are separated by the character '`::`'.

For instance, expression `fund/ancestor::project` returns the `project` node which appears as an ancestor of the `fund` element. The syntax for XPath patterns also permits to associate conditions with the nodes of a path. The path expression identifies the nodes that satisfy all the conditions. Conditions greatly enrich the power of the language, and are a fundamental component in the construction of a sophisticated authorization mechanism. The *conditional expressions* used to represent conditions may operate on the "text" of elements (i.e., the character data in the elements) or on names and values of attributes. Conditions are distinguished from navigation specification by enclosing them within square brackets. Given a path expression $l_1/\ldots/l_n$ on the tree of an XML document, a condition may be defined on any label l_i, enclosing in square brackets a separate evaluation context. The evaluation context contains a predicate that compares the result of

the evaluation of the relative path expression with a constant or another expression. Conditional expressions may be combined with predefined operators `and` and `or` to build boolean expressions. Multiple conditional expressions appearing in a given path expression are considered to be `and`ed (i.e., all the conditions must be satisfied). For instance, expression `/laboratory/project[1]` selects the first `project` child of the `laboratory`. Expression `/laboratory/project[./@name = "Access Models"]/paper[./@type = "internal"]` identifies internal papers related to the project "Access Models".

5 Access authorizations

At each server, a set Auth of access authorizations specifies the actions that subjects are allowed (or forbidden) to exercise on the objects stored at the server site. The object granularity for which authorizations can be specified is the whole object for unstructured files, and the single element/attribute for XML documents. Authorizations can be either positive (permissions) or negative (denials). The reason for having both positive and negative authorizations is to provide a simple and effective way to specify authorizations applicable to sets of subjects/objects with support for exceptions [11, 12].

Authorizations specified on an element can be defined as applicable to the element's attributes only (*local* authorizations) or, in a recursive approach, to its sub-elements and their attributes (*recursive* authorizations). Local authorizations on an element apply to the direct attributes of the element but not to those of its sub-elements. As a complement, recursive authorizations, by propagating permissions/denials from nodes to their descendants in the tree, represent an easy way to specify authorizations holding for the whole structured content of an element (on the whole document if the element is the root). To support exceptions (e.g., the whole content *but* a specific element can be read), recursive propagation from a node applies until stopped by an explicit conflicting (i.e., of different sign) authorization on the descendants. Intuitively, authorizations propagate until overridden by an authorization on a *more specific object* [10].

Authorizations can be specified on single XML documents (*document* or *instance* level authorizations) or on DTDs (*DTD* or *schema* level authorizations). Authorizations specified on a DTD are applicable (through propagated) to all XML documents that are instances of the DTD. Authorizations at the DTD level, together with path expressions with conditions, provide an effective way for specifying authorizations on elements of different documents, possibly in a content-dependent way. Again, according to the "most specific object takes precedence" principle, a schema level authorization being propagated to an instance is overridden by possible authorizations specified for the instance. To address situations where this precedence criteria should not be applied (e.g., cases where an authorization on a document should be applicable unless otherwise stated at the DTD level), we allow users to specify authorizations, either local or recursive, as *weak*. Nonweak authorizations have the behavior sketched above and have priority over authorizations specified for the DTD. Weak autho-

rizations obey the most specific principle within the XML document, but can be overridden by authorizations at the schema level. Access authorizations can be defined as follows.

Definition 3 (Access authorization). *An* access authorization $a \in \mathsf{Auth}$ *is a 5-tuple of the form:* ⟨subject, object, action, sign, type⟩, *where:*

- subject $\in \mathsf{AS}$ *is the subject to whom the authorization is granted;*
- object *is either a URI in* Obj *or is of the form URI:PE, where URI* $\in$ Obj *and PE is a path expression on the tree of URI;*
- action = `read` *is the action on which the authorization is defined;*[2]
- sign $\in \{+, -\}$ *is the sign of the authorization;*
- type $\in \{\mathtt{L}, \mathtt{R}, \mathtt{LW}, \mathtt{RW}\}$ *is the type of the authorization (*L*ocal,* R*ecursive,* L*ocal* W*eak, and* R*ecursive* W*eak, respectively).*

Example 1. Consider the XML document `http://www.lab.com/CSlab.xml` instance of the DTD in Figure 1(a) with URI `http://www.lab.com/laboratory.xml`. The following are examples of protection requirements that can be expressed in our model. For simplicity, in the authorizations we report only the relative URI (`http://www.lab.com/` is the base URI).

Access to private papers is explicitly forbidden to members of the group `Foreign`.
`⟨⟨Foreign,*,*⟩,laboratory.xml:/laboratory//paper[./@category="private"],read,−,R⟩`

Information about public papers of `CSlab` is publicly accessible, unless otherwise specified by authorizations at the DTD-level.
`⟨⟨Public,*,*⟩,CSlab.xml:/laboratory//paper[./@category="public"],read,+,RW⟩`

Information about internal projects of `CSlab` can be accessed by users connected from host `130.89.56.8` who are members of the group `Admin`.
`⟨⟨Admin,130.89.56.8,*⟩,CSlab.xml:project[./@type="internal"],read,+,R⟩`

Users connected from hosts in the `it` domain can access information about managers of `CSlab` public projects.
`⟨⟨Public,*,*.it⟩,CSlab.xml:project[./@type="public"]/manager,read,+,W⟩`

The type associated with each authorization on a given object, at the instance or schema level, determines the "behavior" of the authorization with respect to the object structure, that is, whether it propagates down the tree, it is overridden, or it overrides. The enforcement of the authorizations on the document according to the principles discussed above essentially requires the indication of whether, for an element/attribute in a document, a positive authorization (+), a negative authorization (−), or no authorization applies. Since only part of the authorizations defined on a document may be applicable to all requesters, the set of authorizations on the elements of a document and the authorization behavior along the tree can vary for different requesters. Thus, a first step in access

[2] We limit our consideration to read authorizations. The support of other actions, like write, update, etc., does not complicate the authorization model. However, full support for such actions in the framework of XML has yet to be defined.

control is the evaluation of the authorizations applicable to the requester. This may entail the evaluation of the conditions associated with authorizations, but it does not introduce any complication, since each element/attribute will either satisfy or not such condition. As a complicating factor, however, several (possibly conflicting) authorizations on a given element/attribute may be applicable to a given requester. Different approaches can be used to solve these conflicts [10, 12]. One solution is to consider the authorization with the most specific subject (*"most specific subject takes precedence"* principle), where specificity is dictated by the partial order defined over ASH; other solutions can consider the negative authorization (*"denials take precedence"*), or the positive authorization (*"permissions take precedence"*), or no authorizations (*"nothing takes precedence"*). Other approaches could also be envisioned, such as, for example, considering the sign of the authorizations that are in larger number. For simplicity, in our model we refer to a specific policy and solve conflicts with respect to the "most specific subject takes precedence" principle and, in cases where conflicts remain unsolved (the conflicting authorizations have uncomparable subjects), we stay on the safe side and apply the "denials take precedence" principle. The reason for this specific choice is that the two principles so combined naturally cover the intuitive interpretation that one would expect by the specifications [12]. It is important to note, however, that this specific choice does not restrict in any way our model, which can support any of the policies discussed. Also, different policies could be applied to the same server. The only restriction we impose is that a single policy applies to each specific document. This capability goes towards the definition of multiple policy systems [11].

6 Requester's view on documents

The view of a subject on a document depends on the access permissions and denials specified by the authorizations and their priorities. Such a view can be computed through a tree labeling process, described next. We will use the term node (of a document tree) to refer to either an element or an attribute in the document indiscriminately.

6.1 Document tree labeling

The access authorizations

state whether the subject can, or cannot, access an element/attribute (or set of them). Intuitively, the analysis of all the authorizations for a subject produces a sign (plus or minus) on each element and attribute of a document to which some authorization applies. This unique sign is sufficient to represent the final outcome of the tree labeling. However, in the process itself it is convenient to associate to each node more than one sign corresponding to authorizations of different types. For instance, with respect to a given element, a negative Local authorization and a positive Recursive authorization can exist; the semantics for this would be that the whole element's structured content (with exception of its direct attributes)

```
Algorithm 61  Compute-view algorithm
Input:   A requester rq and an XML document URI
Output: The view of the requester rq on the document URI
Method: /* L is local, R is recursive, LW is local weak, RW is
        recursive weak, LD is local DTD-level, RD is recursive
        DTD-level */
1. Axml := {a ∈ Auth | rq ≤ subject(a),uri(object(a))=URI}
2. Adtd := {a ∈ Auth | rq ≤ subject(a),uri(object(a))=dtd(URI)}
3. Let r be the root of the tree T corresponding to the
   document URI
4. initial_label(r)
5. L_r := first_def([L_r,R_r,LD_r,RD_r,LW_r,RW_r])
6. For each c ∈ children(r) do label(c,r)
7. For each c ∈ children(r) do prune(T,c)

Procedure initial_label(n)
  /* It initializes ⟨L_n, R_n, LD_n, RD_n, LW_n, RW_n⟩. */
  /* Variable t_n spaces over those elements according
     the value of variable t */
1. For t in {L, R, LW, RW} do
  1a. A := {a ∈ Axml | type(a)=t, n ∈ object(a)}
  1b. A := A − {a ∈ A | ∃a' ∈ A, subject(a') ≤ subject(a)}
  1c. If A = ∅ then t_n := 'ε'
        elsif ∃a ∈ A s.t. sign(a)='−' then t_n := '−'
            else t_n := '+'
2. For t in {LD, RD} do
  2a. A := {a ∈ Adtd | type(a)=t, n ∈ object(a) }
  2b. A := A − {a ∈ A | ∃a' ∈ A, subject(a') ≤ subject(a)}
  2c. If A = ∅ then t_n := 'ε'
        elsif ∃a ∈ A s.t. sign(a)='−' then t_n := '−'
            else t_n := '+'

Procedure label(n,p)
  /* Determines the final label of node n basing on
     the label of n and that of its parent p */
  /* It uses first_def, which returns the first non null
     (different from 'ε') element in a sequence */
1. initial_label(n)
2. Case of
  2a. n is an attribute do
        LD_n := first_def([LD_n,LD_p])
        if LW_n = 'ε' then L_n := first_def([L_n,L_p,LD_n])
          else L_n := first_def([L_n,LD_n,RD_p,LW_n])
  2b. n is an element do
        if RW_n = 'ε' then R_n := first_def([R_n,R_p])
        RW_n := first_def([RW_n,RW_p])
        RD_n := first_def([RD_n,RD_p])
        L_n := first_def([L_n,R_n,LD_n,RD_n,LW_n,RW_n])
        For each c ∈ children(n) do label(c,n)

Procedure prune(T,n)
  /* Determines if n has to be removed from T */
1. For each c ∈ children(n) do prune(T,c)
2. ifchildren(n) = ∅ and L_n ≠ '+'
     then remove n from T
```

Fig. 2. Compute-view algorithm

can be accessed. Such semantics must be taken care of in the context of authorization propagation. In principle, therefore, each element can have associated a different sign with respect to the local and recursive permissions/denials, at the instance as well as at the schema level. For this reason, our tree labeling process associates to each node n a 6-tuple $\langle \mathtt{L}_n, \mathtt{R}_n, \mathtt{LD}_n, \mathtt{RD}_n, \mathtt{LW}_n, \mathtt{RW}_n \rangle$, whose content initially reflects the authorizations specified on the node. Each of the elements in the tuple can assume one of three values: '+' for permission, '−' for denial, and 'ε' for no authorization. The different elements reflect the sign of the authorization of type Local, Recursive, Local for the DTD and Recursive for the DTD, Local Weak, and Recursive Weak, holding for node n. (Note that both Local Weak and Recursive Weak for the DTD is missing, since the strength of the authorization is only used to invert the priority between instance and schema authorizations.) The interpretation of authorizations with respect to propagation and overriding (see Section 5) determines the final sign (+ or −) that should be considered to hold for each element and authorization type. Authorizations of each node are propagated to its attributes and, if recursive, to its sub-elements, possibly overridden according to the "most specific object takes precedence" principle, by which: (1) authorizations on a node take precedence over those on its ancestors, and (2) authorizations at the instance level, unless declared as weak, take precedence over authorizations at the schema level. Hence, the labeling of the complete document can be obtained by starting from the root and, proceeding downwards with a preorder visit, updating the 6-tuple of a node n depending on its values and the values of the 6-tuple of node p parent of n in the tree. In particular, the value of $\mathtt{R}_n$ ($\mathtt{RW}_n$ resp.) is set to its current value, if either $\mathtt{R}_n$ or $\mathtt{RW}_n$ is not null (most specific overrides), and to the value of $\mathtt{R}_p$ ($\mathtt{RW}_p$ resp.) propagated down by the parent, otherwise. Schema level authorizations propa-

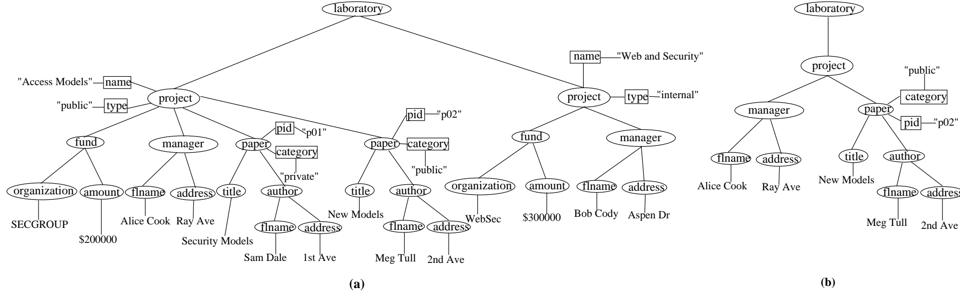

Fig. 3. Tree representation of a valid XML document (a) conforming to the DTD in Figure 1(a) and the view of user `Tom` (b)

gate in a similar way and the sign $\mathtt{RD}_n$ reflecting schema authorizations is set to the current value of $\mathtt{RD}_n$, if not null, and to the value $\mathtt{RD}_p$ propagated down by the parent, otherwise. Given this first propagation step, according to the defined priorities, the sign $(+/-)$ that must hold for the specific element n is the sign expressed by the first not null value (if any) among: $\mathtt{L}_n$, $\mathtt{R}_n$, $\mathtt{LD}_n$, $\mathtt{RD}_n$, $\mathtt{LW}_n$, and $\mathtt{RW}_n$. $\mathtt{L}_n$ is updated to such value so that, at the end of the tree visit, $\mathtt{L}_n$ contains the "winning" sign for n. The 6-tuples assigned to attributes are updated in a similar way with some minor changes due to the fact that: (1) $\mathtt{R}_n$, $\mathtt{RW}_n$, and $\mathtt{RD}_n$ are always null for an attribute (being a terminal element of the tree, no propagation is possible), and (2) authorizations specified as `Local` on the node p parent of the attribute must propagate to the attribute.

Figure 2 illustrates an algorithm, **compute-view**, enforcing the labeling procedure described. The algorithm uses function `first_def` which, given a sequence of values on the domain $\{-, +, \varepsilon\}$, returns the first value in the sequence different from 'ε'. Given a requester *rq* and an XML document *URI*, the algorithm starts by determining the set of authorizations defined for the document at the instance level (set *Axml* in step 1) and at the schema level (set *Adtd* in step 2). The initial label of the root r is determined and (since the root has no parent) its final $\mathtt{L}_r$ can easily be determined by taking the sign with the highest priority that is not null, that is, the one coming first (`first_def`) in the sequence $\mathtt{L}_r$,$\mathtt{R}_r$,$\mathtt{LD}_r$,$\mathtt{RD}_r$,$\mathtt{LW}_r$,$\mathtt{RW}_r$. Procedure **label**(c,r) is then called for each of the children (element or attribute) c of the root to determine the label of c. Procedure **label**(n,p) first computes the initial labeling of n by calling procedure **initial_label**. Step 1 of **initial_label** computes the initial value of $\mathtt{L}_n$, $\mathtt{R}_n$, $\mathtt{LW}_n$, and $\mathtt{RW}_n$. For each authorization type $t \in \{\mathtt{L}, \mathtt{R}, \mathtt{LW}, \mathtt{RW}\}$, the set A of authorizations of type t is determined (Step 1a). Set A is then updated by discarding authorizations overridden by other authorizations with more specific subjects (Step 1b). If the resulting set is empty, t_n (i.e., $\mathtt{L}_n$, $\mathtt{R}_n$, $\mathtt{LW}_n$, or $\mathtt{RW}_n$, depending on the value of t) is set to 'ε'. Otherwise, it is set to '$-$' or '$+$', according to whether a negative authorization exists ("denials take precedence") or does not exist in A (Step 1c). In a similar way, Step 2 determines the sign of $\mathtt{LD}_n$ and $\mathtt{RD}_n$, respectively. Procedure **label**(n,p) updates the initial label so produced on

the basis of the label of n's parent p as previously discussed. If n is an element, the procedure is recursively called for each sub-elements of n.

6.2 Transformation process

As a result of the labeling process, the value of $\mathtt{L}_n$ for each node n will contain the sign, if any, reflecting whether the node can be accessed ('+') or not ('−'). The value of $\mathtt{L}_n$ is equal to 'ε' in the case where no authorizations have been specified nor can be derived for n. Value 'ε' can be interpreted either as a negation or as a permission, corresponding to the enforcement of the *closed* and the *open* policy, respectively [11]. In the following, we assume the closed policy. Accordingly, the requester is allowed to access all the elements and attributes whose label is positive. To preserve the structure of the document, the portion of the document visible to the requester will also include start and end tags of elements with a negative or undefined label, which have a descendant with a positive label. The view on the document can be obtained by pruning from the original document tree all the subtrees containing only nodes labeled negative or undefined. Figure 2 illustrates a procedure, **prune**, enforcing the pruning process described.

The pruned document may not be valid with respect to the DTD referenced by the original XML document. This may happen, for instance, when required attributes are deleted. To avoid this problem, a *loosening* transformation is applied to the DTD. Loosening a DTD simply means to define as *optional* all the elements and attributes marked as *required* in the original DTD. The DTD loosening prevents users from detecting whether information was hidden by the security enforcement or simply missing in the original document.

Example 2. Consider the XML document `http://www.lab.com/CSlab.xml` in Figure 3(a) and the set Auth of authorizations in Example 1. Consider now a request to read this document submitted by user `Tom`, member of group `Foreign`, when connected from `infosys.bld1.it` (`130.100.50.8`). Figure 3(b) shows the view of `Tom` resulting after the labeling and transformation process.

7 Implementation of the security processor

We are currently implementing a *security processor* for XML documents based on the security model described in this paper. The main usage scenario for our system involves a user requesting a set of XML documents from a remote site, either through an HTTP request or as the result of a query [4]. Our processor takes as input a valid XML document requested by the user, together with its *XML Access Control List* (XACL) listing the associated access authorizations at instance level. The processor operation also involves the document's DTD and the associated XACL specifying schema level authorizations. The processor output is a valid XML document including only the information the user is allowed to access. In our system, documents and DTDs are internally represented as object trees, according to the Document Object Model (DOM) Level One (Core)

specification [18]. Our security processor computes an *on line transformation* on XML documents. Its execution cycle consists of four basic steps.

1. The *parsing* step consists in the syntax check of the requested document with respect to the associated DTD and its compilation to obtain an *object-oriented document graph* according to the DOM format.
2. The *tree labeling* step involves the recursive labeling of the DOM tree according to the authorizations listed in the XACLs associated to the document and its DTD (see Section 6.1).
3. The *transformation* step is a pruning of the DOM tree according to its labeling (see Section 6.2). Such a pruning is computed by means of a standard postorder visit to the labeled DOM tree. This pruning preserves the validity of the document with respect to the *loosened version* of its original DTD.
4. The *unparsing* step consists in generating a valid XML document in text format, simply by unparsing the pruned DOM tree computed by the previous step.

The resulting XML document, together with the loosened DTD, can then be transmitted to the user who requested access to the document.

Two main architectural patterns are currently used for XML documents browsing and querying: *server side* and *client side* processing, the former being more common in association with translation to HTML.

Our access control enforcement is performed on the server side, regardless of whether other operations (e.g., translation to HTML) are performed by the server site or by the client module. The XML document computed by the security processor execution is transferred to the client as the result of its original request. In our current design, the security processor is a *service component* in the framework of a complete architecture [5]. The reason for this architectural choice are twofold: first, server-side execution prevents the accidental transfer to the client of information it is not allowed to see or process; second, it ensures the operation and even the presence of security checking to be completely transparent to remote clients.

8 Conclusions

We have defined an access control model for restricting access to Web documents that takes into consideration the semi-structured organization of data and their semantics. The result is an access control system that, while powerful and able to easily represent different protection requirements, proves simple and of easy integration with existing applications. Our proposal leaves space for further work. Issues to be investigated include: the consideration of requests in form of generic queries, the support for write and update operations on the documents, and the enforcement of credentials and history- and time-based restrictions on access. Finally, we intend to prepare in a short time a Web site to demonstrate the characteristics of our proposal.

References

1. AlphaWorks. *XML Security Suite*, April 1999. http://www.alphaWorks.com/tech/xmlsecuritysuite.
2. T. Berners-Lee, R. Fielding, and L. Masinter. *Uniform Resource Identifiers (URI): Generic Syntax*, 1998. http://www.isi.edu/in-notes/rfc2396.txt.
3. T. Bray et.al. (ed.). *Extensible Markup Language (XML) 1.0.* World Wide Web Consortium (W3C), February 1998. http://www.w3.org/TR/REC-xml.
4. S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tanca. XML-GL: A Graphical Language for Querying and Restructuring XML Documents. In *Proc. of the Eighth Int. Conference on the World Wide Web*, Toronto, May 1999.
5. S. Ceri, P. Fraternali, and S. Paraboschi. Data-Driven, One-To-One Web Site Generation for Data-Intensive Applications. In *Proc. of the 25th Int. Conference on VLDB*, Edinburgh, September 1999.
6. CheckFree Corp. *Open Financial Exchange Specification 1.0.2*, 1998. http://www.ofx.net/.
7. S. DeRose, D. Orchard, and B. Trafford. *XML Linking Language (XLINK)*, July 1999. http://www.w3.org/TR/xlink.
8. C. Ellerman. *Channel Definition Format (CDF)*, March 1997. http://www.w3.org/TR/NOTE-CDFsubmit.html.
9. E.B. Fernandez, E. Gudes, and H. Song. A Model of Evaluation and Administration of Security in Object-Oriented Databases. *IEEE TKDE*, 6(2):275–292, April 1994.
10. S. Jajodia, P. Samarati, and V.S. Subrahmanian. A Logical Language for Expressing Authorizations. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 31–42, Oakland, CA, May 1997.
11. S. Jajodia, P. Samarati, V.S. Subramanian, and E. Bertino. A Unified Framework for Enforcing Multiple Access Control Policies. In *Proc. of the 1997 ACM International SIGMOD Conference on Management of Data*, Tucson, AZ, May 1997.
12. T.F. Lunt. Access Control Policies for Database Systems. In C.E. Landwehr, editor, *Database Security, II: Status and Prospects*, pages 41–52. North-Holland, Amsterdam, 1989.
13. F. Rabitti, E. Bertino, W. Kim, and D. Woelk. A Model of Authorization for Next-Generation Database Systems. *ACM TODS*, 16(1):89–131, March 1991.
14. J. Reagle and L.F. Cranor. The Platform for Privacy Preferences. *Communications of the ACM*, 42(2):48–55, February 1999.
15. Rutgers Security Team. *WWW Security. A Survey*, 1999. http://www-ns.rutgers.edu/www-security/.
16. P. Samarati, E. Bertino, and S. Jajodia. An Authorization Model for a Distributed Hypertext System. *IEEE TKDE*, 8(4):555–562, August 1996.
17. A. van Hoff, H. Partovi, and T. Thai. *The Open Software Description Format (OSD)*, August 1997. http://www.w3.org/TR/NOTE-OSD.html.
18. L. Wood. *Document Object Model Level 1 Specification*, October 1998. http://www.w3.org/pub/WWW/REC-DOM-Level-1/.
19. World Wide Web Consortium (W3C). *Extensible Stylesheet Language (XSL) Specification*, April 1999. http://www.w3.org/TR/WD-xsl.
20. World Wide Web Consortium (W3C). *XML Path Language (XPath) Version 1.0*, October 1999. http://www.w3.org/TR/PR-xpath19991008.