Design and Implementation of an Access Control Processor for XML Documents*

Ernesto Damiani[1] Sabrina De Capitani di Vimercati[2] Stefano Paraboschi[3] Pierangela Samarati[1]

(1) Università di Milano, Polo Didattico di Crema, Via Bramante 65, Crema (CR), Italy

(2) Università di Brescia, Dip. Elettronica per l'Automazione, Via Branze 38, 25123 Brescia, Italy

(3) Politecnico di Milano, Dip. Elettronica e Informazione, Piazza L. da Vinci 32, 20133 Milano, Italy

edamiani@crema.unimi.it, {decapita,samarati}@dsi.unimi.it, parabosc@elet.polimi.it

Abstract

More and more information is distributed in XML format, both on corporate Intranets and on the global Net. In this paper an *Access Control System* for XML is described allowing for definition and enforcement of access restrictions directly on the structure and content of XML documents, thus providing a simple and effective way for users to protect information at the same granularity level provided by the language itself.

Keywords

Security, Access control model, XML

1 Introduction

As more and more information is made available in *eXtensible Markup Language* (XML) format, both on corporate Intranets and on the global Net, concerns are being raised by developers and end-users about XML security problems. Early research work about XML was not directly related to access control and security, because XML was initially introduced as a data format for documents; therefore, many researchers assumed well-known techniques for securing documents to be straightforwardly applicable to XML data. But the way XML is being positioned has caused some to question if additional measures will be necessary.

For example, in the scenario of the oncoming FASTER (*Flexible Access to Statistics, Tables, and Electronic Resources*) project, end-users will be able to control their interaction with Web sites by pulling the information they are interested in out of dynamically generated XML documents. However, different users may well have different interests or access authorizations, and XML enabled servers will need to know which data each user should get, at a finer level of granularity than whole documents. In other words, some FASTER applications will need to block or allow access to entire XML instances, while others will control access at the tag level. The control residing at the tag level is particularly important in the view of wider use of the *XLink* and *XPointer* standards, which enable applications to retrieve portions of documents. Indeed, a clean model for dynamic access control with granularity control is needed to allow XML documents to link against arbitrary XML chunks. It is interesting to remark that the same observation applies to authentication and encryption-based techniques, that naturally complement access control in our usage scenario. With authentication, the server will know what information can be sent to the user based on that user's identity or certified property (e.g., group membership), whereas encryption

*The work presented in the paper has been supported by Esprit Project "W3I3", Esprit Project "FASTER", MURST project "Data-X", and by the HP Internet Philanthropic Initiative

will only let users with adequate decryption keys see the message. Therefore, XML security should support the entire range of coarse to fine grain granularity. In the remainder of this section, we propose five basic requirements for standardizing XML access control at the tag level. Our requirements take into account the experience of other FASTER consortium partners, and are directed at large-scale knowledge management within organizations using XML, as well as at XML-based Internet applications.

1. *Support of authorizations at different organizational levels.* Organizations may need to enforce security policies on huge document-bases, often dynamically created from heterogeneous datasources; on the other hand, site administrators require full control on authorization specifications on single documents.

2. *Extension to existing Web server technology.* XML documents are usually made available by means of Web sites, using a variety of HTTP-based protocols. XML access control must exploit current solutions in much the same way as cryptography-based services, without interfering with existing APIs and development tools.

3. *Fine-grained access control.* Access control policies should be supported at all levels of granularity, including documents and individual XML elements.

4. *Transparency.* The access control system operation should be as transparent as possible to the requesters. The requester should not be aware of the information within a document which is being hidden to them by the access control system. The transparency of the access control must be preserved by the presentation and rendering phases and may therefore impose constraints on the behavior of technologies such as CSS and XSL [18]. In particular, access control should preserve the *validity* of the documents with respect to their DTDs.

5. *Smoothless integration* with existing technologies for user authentication (e.g. digital signatures). Access control should complement tag-level authentication based on digital signatures.

Figure 1 depicts the conceptual architecture of our approach. A *central authority* uses a pool of XML DTDs to specify the format of information to be exchanged within the organization. XML documents instances of such DTDs are defined and maintained at each site, describing the site-specific information. The *schema-instance relationship* between XML documents and DTDs naturally supports the distinction between two levels of authorizations, both of them allowing for fine grained specifications. Namely, we distinguish: *1)* Low-level authorizations, associated to XML documents, providing full control on authorizations on a document-by-document basis; *2)* High-level authorizations, associated to XML DTDs, providing organization-wide and department-wide declarations of access permissions. Centrally specified DTD-level authorizations can be mandatory, stating *impositions* of the central authority to lower organizational levels where XML documents are created and managed, usually by means of a network of federated Web sites. This technique allows for easy, centralized modification of access permissions on large document sets, and provides a general, abstract way of specifying access authorizations. In other words, specifying authorizations at the DTD level cleanly separates access control specified via XML markup from access control policies defined for the individual datasources (e.g., relational databases vs. file systems) which are different from one another both in granularity and abstraction level. Each departmental authority managing a Web site retains the right to define its own authorizations (again, at the granularity of XML tags) on individual documents, or to document sets by means of wild cards. In our model local authorities can also define authorizations at the DTD level; however such authorizations only apply to the documents of the local domain.

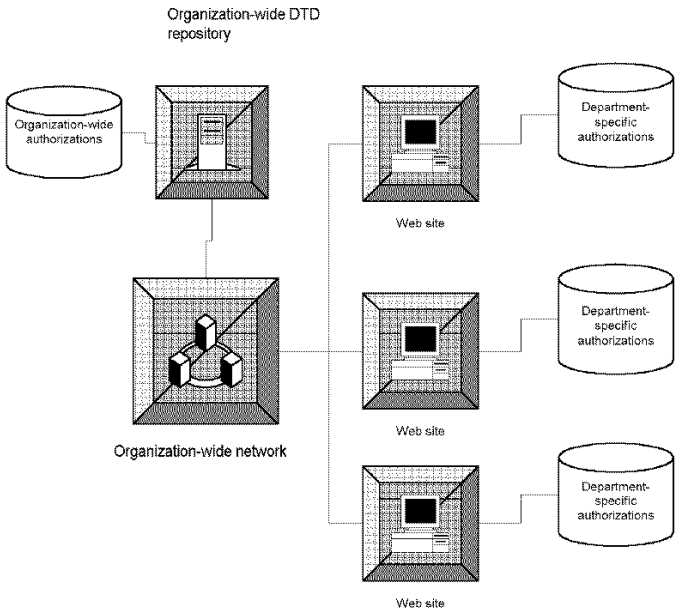

Figure 1: Conceptual architecture

2 Authorization Specification

The architectural framework depicted in Figure 1 describes the basic components taking part in the specification of access and protection requirements. We now discuss their specification. Before introducing the form and semantics of the authorizations supported by our model, we describe the basic features that they need to provide to satisfy requirements 1 and 3 discussed in the introduction.

Collection based vs instance based authorizations The different protection requirements that different documents may have call for the support of access restrictions at the level of each specific document. On the other hand, requiring the specification of authorizations for each single document would make the authorization specification task much too heavy. The system should support, beside authorizations on single documents, authorizations on collections of documents. The concept of DTD can be naturally exploited to this end, by allowing protection requirements to refer to DTD or XML documents, where requirements specified at the level of DTD apply to all those documents instance of the DTDs. The use of DTDs as a primary way to refer to sets of documents as opposed to the use of file system structures (directory) used in previous approaches, is consistent with the fact that our approach takes advantage of the data semantics, departing from the limitations of storage-based structures. The fact that instances of DTDs share a common (semi)structure, allows the association with DTD-level authorizations of conditions that limit the documents/elements to which the authorization apply. This way authorizations can be specified which apply only to certain instances of a DTD. While using DTDs as a primary way to reference classes of documents, we do not discard other methods. In particular, our model also supports the use of wild cards in the specification of document URIs and the possibility of referencing and evaluating meta properties, such as RDF markup [19]. The use of wildcards allows the specification of authorizations that apply to all documents matching a given path expression, depending on the file system organization. The reference to meta properties allows the specification of authorizations that apply to all documents satisfying specific properties, expressed by means of meta information associated with the documents (e.g., creator, creation date,

and so on). Meta properties can also be used to provide organization of documents in domains [13].

Organization's wide vs. site specific authorizations Access and protection requirements can be specified both at the level of the enterprise, stating general regulations that should hold, and at the level of specific domains (part of the enterprise) where, according to a local policy, additional constraints may need to be specified or some constraints may need to be relaxed. Organizations specify authorizations with respect to DTDs; sites can specify authorizations with respect to specific documents as well as to DTDs. The two types of DTD-level authorizations have complementary roles in increasing access control flexibility. *Global* DTD-level authorizations stated by a central authority can be effectively used to implement corporate-wide access control policies on document classes. *Local* DTD-level authorizations specified by departmental authorities allow for department-wide access control policies complementing the corporate ones. Moreover they alleviate administration chores by allowing concise specification of site-wide authorizations.

Document vs. element/attribute authorizations The identification of elements and attributes within a document provided by XML tags can be exploited to specify authorizations at a fine grained level. Authorizations specified for an element are intended to be applicable to all its attributes. Again, to avoid the need of specifying authorizations for each single element in a document, the document structure can be exploited by supporting a recursive interpretation of authorizations by which an authorization specified on an element applies to its whole content (attributes and subelements). Our model allows to specify whether an authorizations specified for an element is local to its own data (PC data and attributes) or applies recursively to all its subelements. The authorization on a document in its entirety is specified as a recursive authorization on its root.

Exception support (permissions and denials) The support of authorizations at different granularity levels allows for easy expressiveness of both fine and coarse grained authorizations. Such an advantage would remain however very limited without the ability of the authorization model to support exceptions, since the presence of a granule (document or an element/attribute) with protection requirements different from those of its siblings would require the explicit specification of authorizations at that specific granularity level. For instance, the situation where a user should be granted access to all the documents of a DTD but one specific instance, would imply the need of stating the authorizations explicitly for all the other documents as well; thereby ruling out the advantage of supporting authorizations at the DTD level. A simple way to support exceptions is by using both positive (permissions) and negative (denials) authorizations; where permissions and denials can override each other. According to intuition, overriding typically occurs when going to a finer granularity level, according to the "most specific takes precedence principle" [11, 8]. Finer grained authorizations override coarser ones – each document being at a finer grain than its DTD and each element/attribute being at a finer grain than the elements in which it is contained.

Hard and soft statements (ruling out exceptions and filling the blanks) The support of exceptions while clearly adding to the expressiveness of the model, allows stated protection requirements to be possibly overridden. When authorization specification spans different administrative competences and authorities, as it is the case of organization-wide authorizations vs. site-specific authorizations, there might be cases where such a capability needs to be restricted. The "most specific takes precedence" principle dictates that authorizations specified on a document override (where conflicting) authorizations specified on its DTD. In organizational terms, the authorization specified at a site would always override the authorizations specified at the organization level. We can imagine two scenarios where such a behavior is not wanted. First, at the organization level certain specifications may need to be declared as mandatory, meaning they should be obeyed at all the sites – no site dis-

cretionary statement allowed. Second, at the site level, certain specifications may need to be declared as *soft*, meaning they should be applied only if nothing has been stated at the organization level. In both scenarios the need is to subvert the "most specific takes precedence principle". The fact that the need may come either from the organization or from the site, requiring the ability to support its expression in association with the both DTD and document authorizations. In particular, the enterprise can specify DTD authorizations as hard, sites can specify document authorizations as soft. (For the sake of simplicity of the model, we do not allow sites to specify strong DTD authorizations as it would introduce complications while not adding in expressiveness.)

3 Authorizations

The list of features illustrated in the previous section outlines the form and semantics of the authorizations supported by our model. We can then summarize the discussion above and introduce our authorizations as follows.

- Authorizations can be specified at the level of a DTD (schema) or specific documents (instance). DTD authorizations can be specified either at the global organization level or at the local site. Document authorizations can be specified at the local site.
- Both DTD and XML authorizations can be specified with reference to each single element/attribute in a document. Authorizations on an element can be declared as *recursive* (apply to its subelements) or *local* (apply only to its direct attributes and PCdata).
- DTD-level authorizations specified at the global level can be declared as *hard*.
- Document-level authorizations can be declared as *soft*.

Authorizations specified for each XML document/DTD (elements within) are stored in a (*XML Access Sheets* - XAS) associated with the document/DTD, bringing to the organization illustrated in Figure 2. The representation and storage of authorizations in a component XAS separate from the document they protect follows the well known design principle requiring clean separation between data model and access control model [4]. Also, it has the great advantage of allowing the specification of authorizations on dynamically generated XML documents. Besides, enclosing authorizations in the documents themselves would compromise readability of both the documents and its access restrictions.

We anticipate that, in the access control processing, DTD-level authorizations specified at the global level and those specified at the local level are, with respect to each DTD, merged by performing a *flat union*. In other words, organization-wide and site specific authorizations are treated in the same way (although, remember, that organization-wide authorizations apply to all the documents in the network while site-specific authorizations apply only to documents stored at the site). Given this, in the future we will simply refer to DTD authorizations without making any distinction of where they have been specified. The reason for merging the two sets of authorizations with a simple flat union is simplicity. We do observe that, in principle, even at this level some notion of "specificity" could be applied. This reasoning could also be possibly extended by considering any number of intermediate organizational levels which could be reflected in priorities associated with the authorizations. We note however, that the most specific principle of DTD vs XML, together with the possibility of specifying hard and soft options subverting it, does already provide, on the two organizational levels considered which were of interest in our project, such expressiveness. As it may be clear from the previous discussion, we allow the specification of hard authorizations only at the global level. In this way no unresolvable conflict can arise. This does not limit expressiveness: site administrators that want their authorizations to override global authorizations

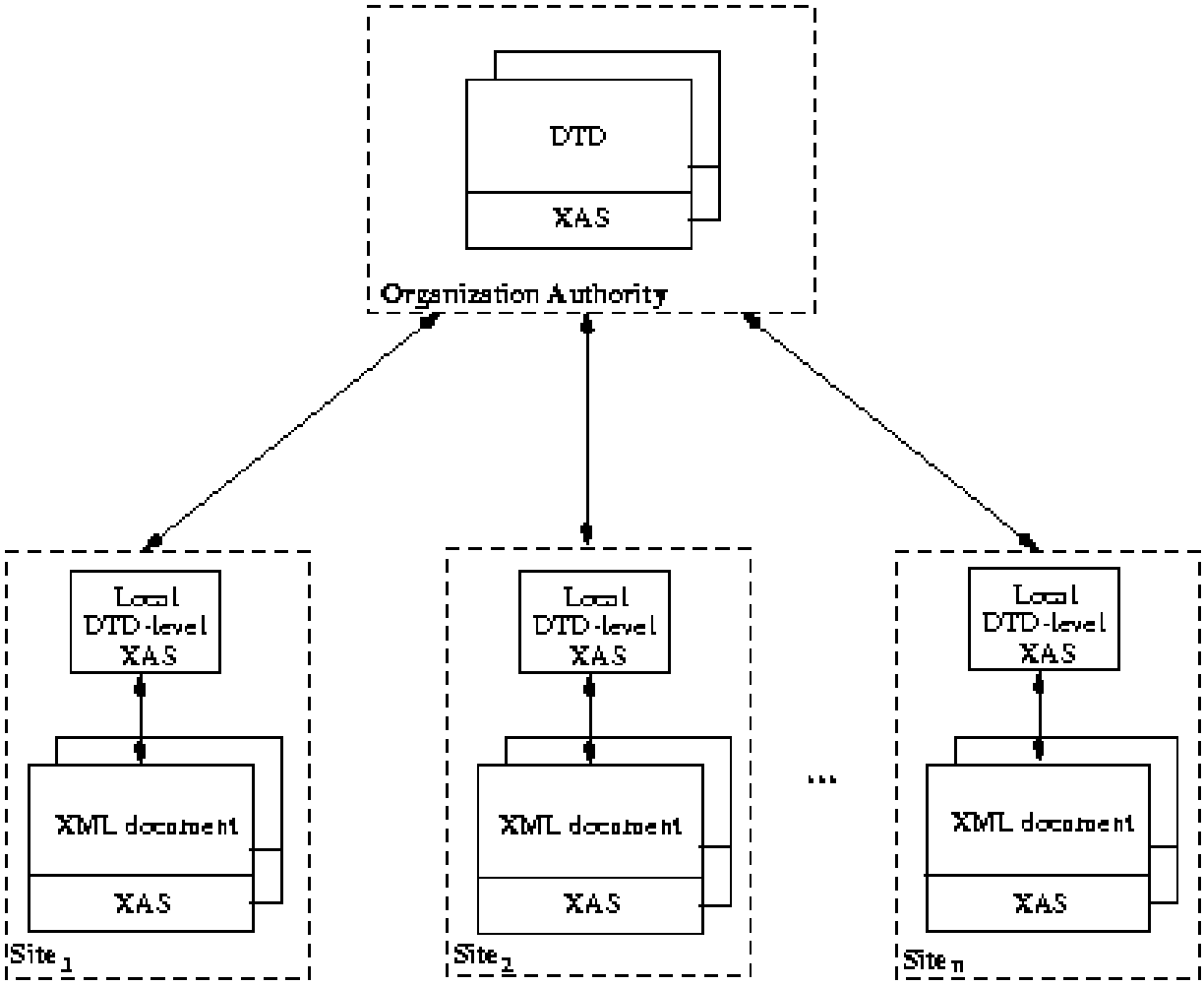

Figure 2: Authorization information stored at the different levels

```
<!ELEMENT set_of_authorizations (authorization)+>
<!ELEMENT authorization (subject,object,action,sign,type,priority)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT object (#PCDATA)>
<!ELEMENT action empty>
<!ELEMENT sign empty>
<!ELEMENT type empty>
<!ELEMENT priority empty>
<!ATTLIST set_of_authorizations about CDATA #REQUIRED >
<!ATTLIST action    value (read)            #REQUIRED>
<!ATTLIST sign      value (+|-)             #REQUIRED>
<!ATTLIST type      value (local|recursive) #REQUIRED>
<!ATTLIST priority  value (hard|soft)       #IMPLIED>
```

Figure 3: XAS syntax

can simply do so by going to the instance level (wildcard characters and meta properties allow doing so without the need of specifying an authorization for each instance).

The XAS associated with a document/DTD contains the set of authorizations specified for the document/DTD or elements within. The authorizations are expressed in XML and comply to the DTD illustrated in Figure 3. Each authorization states the permission or denial (depending on the value of `sign`) for a `subject` to execute a certain `action` on an `object`, together with the priority (`soft` vs `hard`) and type (`recursive` vs `local`) of such a statement. Here `object` identifies an element or set of elements in a document or set of documents. We now describe in more details how documents and elements/attributes within them are references to the purpose of specifying authorizations. We then discuss authorization subjects.

3.1 Identifying Authorization Objects via Path Expressions

In the traditional Web security setting, Uniform Resource Identifiers (URI) [2] are used to denote the resources to be protected. Each document and DTD is characterized by a single URI. As we go to a finer level of granularity we need to reference specific elements and attributes in documents. Elements/attributes in a document can be referenced by means of *path expressions*. A straightforward way of writing path expressions is by using the XPath language [20]. The reason for this choice is that several tools are currently available which can be easily reused to produce a functioning system. XPath expressions make reference to the tree organization of documents/DTDs which is obtained in a simple way by interpreting elements and attributes as children of the element in which they are directly contained. Each element and attribute can be then referenced by means of the tree path that must be followed to reach it. An XPath on an XML document tree is a sequence of element names or predefined functions separated by the character `/` (slash): $l_1/l_2/\ldots/l_n$. For instance, path expression `/division/about_div/member` denotes the nodes of the `member` element which are children of `about_div` elements, which are children of `division` elements. Path expressions can be *absolute* or *relative*. Absolute path expressions, prefixed by a slash character, start from the root of the document. Relative path expressions, which start with an element name, describe a path whose initial point is any element in the document.

A very interesting characteristic of path expressions which very conveniently increases the expressiveness of authorizations is the support of conditions. Conditions associated with a path expression refine the set of nodes matching the path expression. Conditions may impose constraints on element contents (i.e., the "text" of elements) or on names and values of attributes. A condition can follow any label in a path

expression and is identified as such by enclosing it between square brackets. Given a path expression $l_1/l_2/\ldots/l_n$, a condition on label l_i restricts the application of the path expressions only to those node(s) l_i for which the condition evaluates true.

3.2 Identifying Authorization Subjects

A straightforward and largely used approach to refer to authorization subjects and access requesters is via *user identity* and/or the *location* from which their requests originate, where locations can be expressed via numeric IP addresses (e.g., `159.149.51.40`) or via symbolic names (e.g., `tweety.acme.com`). Our system combines all these features. Subjects requesting access are characterized by a triple ⟨`user-id`,`IP-address`, `sym-address`⟩, where `user-id` is the login name with which the user connected to the server, `IP-address` is the address of the client machine and `sym-address`is the machine's DNS name. (Remote identities trusted by the server (using a *Certification Authority*, or any other secure infrastructure) can be considered as well.) Authorizations can also be specified with reference to user *groups* and/or *location patterns*. Groups are set of users defined at the server; they do not need to be disjoint and can be nested. A location pattern is an expression identifying a set of physical locations, with reference to either their symbolic names or IP addresses. Patterns are specified by using the wild card character `*`. For instance, `159.149.*` denotes all the machines belonging to subnetwork `159.149`. Similarly, `*.edu`, and `*.it` respectively denote all the machines in the Educational and Italy domains. A user can be seen a singleton group, a location as a simple pattern. Groups and location patterns provide an effective way to specify authorizations holding for large set of subjects: authorizations granted to a group with respect to some location pattern apply to all the members of the group when connected from a machine satisfying the pattern. For instance, authorizations granted to ⟨`Employee`,`159.149.100.*`,`*`⟩ apply to all the members of group `Employee` when connected from machines in subnetwork `159.149.100.*`. Authorizations granted to ⟨`Employee`,`*`,`*.acme.com`⟩ apply to all employees connected from the local acme network. We observe that while authorization subjects are conceptually identified by triples of the general hierarchy, relationships between address (and symbolic names) patterns can be detected straightforwardly; therefore, only the usual user-group hierarchy needs to be explicitly defined and stored at the sites (or communicated to them [7]). It is also important to note that the consideration of user's identity and location identifiers does not rule out the possibility of partial or completely anonymous connection, to which general authorizations, specified for a group `Public` to which everybody belongs and pattern `*` can be applied.

4 Authorization Enforcement

For each possible requester (user connected from a certain location) and document, the authorizations on the document applicable to the requester describe what information can or cannot be returned to the requester. Hence, given the request from a subject to access a document, the joint application of the DTD-level and document-level authorizations applicable to the subject will produce a custom *view* on the document, including only the information that particular requester is entitled to see. The access control process must therefore evaluate the authorizations applicable to an access request to compute such a view. We now briefly outline this computation process which exploits the hierarchical organization of documents, by operating on their DOM tree. Intuitively, the analysis of all the authorizations holding for the requester on a document produces an access decision (access or not access) on each node of the document. The process to obtain this final outcome starts with a *labeling procedure* whose output reflects the authorizations on the different nodes applicable to the subject. Since authorizations can be of different level (DTD vs. instance), type (local vs. recursive), and priority (hard vs. soft), more than one sign is associated with each node. More precisely, the process assigns to each node a label reflecting the sign (permission or denial) of authorizations, if any, existing for that node at the considered type, priority,

and level. A simple representation of these labels is to associate with a node an 8-tuple (2^3, each of the three fields has two possible values). The sign of each label can be: '+' (permission), '−' (denial), or 'ε' (no authorization). We note that more authorizations can exist with respect to each label. In this case a resolution policy is applied to get a unique final sign [6, 8] for the label. Simple and natural conflict resolution policies include the "most specific subject takes precedence principle" (users/subgroups are more specific than the groups to which they belong, sub-patterns are more specific than their more general form) and the "denial takes precedence principle" [8], and are those currently supported by our prototype.

After this initial labeling, propagation is applied so that local authorizations holding for each node are propagated to its attributes, while recursive authorizations are also propagated to its sub-elements. Authorizations may be overridden as follows:

1. authorizations on a node take precedence over those on its ancestors,
2. authorizations at the document level take precedence over authorizations at the local and global DTD levels, unless they are explicitly declared as *soft*.
3. *hard* authorizations at the global-DTD level override authorizations at other levels.

This labeling process can be obtained by means of a preorder visit on the document's DOM tree. At the end of the tree visit a single label is associated with each node defining its final sign, if any. If no sign has been determined for a node (no authorizations have been specified nor can be derived for it), its value is set to the null value 'ε'. Value 'ε' can be interpreted either as a negation (transformed in a '-') or as a permission (transformed in a '+'), corresponding to the enforcement of a *closed* and the *open* policy, respectively [8]. In the sequel, we shall act conservatively, choosing the closed policy.

For how the labeling process has been performed, the requester is allowed to access all the elements and attributes whose label is positive. Note that, in order 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, if the elements have a descendant with a positive label. The final view on the document can be obtained simply by pruning from the original document tree all the subtrees containing only nodes labeled negative. This pruning is performed by a procedure that executes a postorder visit on the document and removes any leaf labeled '−'. The pruned document may not be valid with respect to the DTD referenced by the original XML document. This will happen, for instance, when required attributes are deleted because the requester is not entitled to receive them. To avoid this problem, a *loosening* transformation can be applied to the DTD. In the simplest case, loosening a DTD simply means to define as optional all the elements and attributes marked as required in the original DTD. This "naive" loosening technique is currently justified by implementation-related considerations, as there is no efficient technology for processing DTDs even remotely comparable to the one available for documents. However, as DTD processing standards such as DOM level-2 [17] come of age, more sophisticated loosening techniques can be devised by taking into account the elements that are pruned by the transformation and selectively redefining them as optional. "Looser" DTDs also prevent users from detecting whether information was hidden by access control enforcement or was simply missing in the original document. The loosening process is aimed at the satisfaction of requirement 4 stated in Section 1.

5 Design and Implementation Guidelines

First of all, architectural design will be briefly discussed. Two main architectural patterns are currently used for the design of XML/XSL systems: *server side* and *client side* XSL processing (see Section 6). The former technique is common in association with translation to HTML and provides limited interaction: XML documents are translated to HTML before sending them to the client, avoiding the need for the

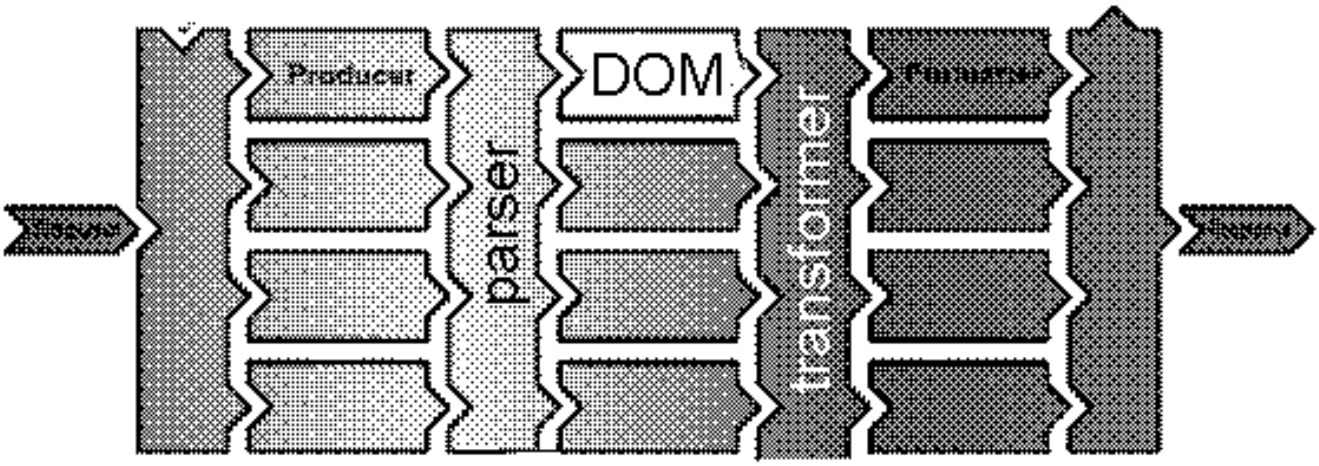

Figure 4: Design pattern for the processor transformer

client browser to provide XML support. The latter technique requires an XSL processor to be part of the client, in order to provide it with rendering capabilities. In our approach, access control enforcement is always performed on the server side, regardless of whether other operations, such as XSL-based rendering or translation to HTML, are performed by the server site or by the client module.

The reason for this architectural choice are twofold: first, server-side execution prevents transferring 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. The main usage scenario for our system involves a user requesting a set of XML elements from a remote site, either through an HTTP request or as the result of a query [5]. Our processor takes as input the valid XML document requested by the user or computed by the query, together with an *XML Access Sheet* (XAS) listing the associated access authorizations at document level. The processor operation also involves the document's DTD and the associated XAS specifying DTD level authorizations. In our design, the processor module is a *transformer* in the framework of a complete architecture complying to the well- known *Pipes and Filters* design pattern (Figure 4) [3]. The service's interface is locally available to Web servers components storing XML documents. This solution is aimed to satisfy requirement 2 stated in Section 1. The processor output is a valid XML document including only the information the user is allowed to access. The XML document computed by processor is then transferred to the client as the result of its original request.

5.1 Internal Data Model

In our system, documents and DTDs are internally represented as object trees, according to the Document Object Model (DOM) Level 1 specification [16]. DOM provides an object-oriented *Application Program Interface* (API) for HTML and XML documents. Namely, DOM defines a set of object definitions such as `Element`, `Attr`, and `Text`, to build an object-oriented document which closely models the document

structure. While DOM trees are topologically equivalent to the XML trees defined in Section 3.1, they represent element containment by means of the object-oriented *part-of* relationship. For example, an XML element is represented in DOM by an `Element` object; an element contained within another element is represented as a child `Element` object, and text contained in an element is represented as a child `Text` object. The main classes of the DOM hierarchy are `Node`, `Document`, `Element`, `Attr` and `Text`. `Node` is the generic element in an XML document and provides basic methods for insertion, deletion and editing; via inheritance, such methods are also defined for more specialized classes in the hierarchy. `Node` also provides a powerful set of navigation methods, such as `parentNode`, `firstChild` and `nextSibling`. Navigation methods allow transformer modules of the security processor to visit the DOM representation of XML documents via a sequence of calls to the interface. Specifically, the `NodeList` method, which returns in a container all the children of the current node, has been used to implement the fast labeling procedure which is the core of the access control processor. Our implementation is based on a *Secure* extension of the classes of the DOM hierarchy, like `SecureDocument` and `SecureElement`. Our extension is fully compatible with other extensions supporting element-wise digital signatures, such as DOMhash [1]. Such compatibility is a step towards satisfaction of requirement 5 stated in Section 1.

5.2 Execution Phases

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

1. *Parsing.* 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. Since parsing is performed externally when the access control processor is used as a transformer in the framework of a *Pipes and Filters* system, here we do not deal with parsing issues in detail.
2. *Tree labeling.* The labeling step involves the propagation of the labeling of the DOM tree according to the authorizations listed in the XAS associated to the document and its DTD, both at the organization and at site level. Its implementation takes advantage of the extended DOM interface for object nodes, which provides a labeling interface. Standard DOM methods allow the transformer to follow *part-of* links from each node to its children by means of a standard method call. The authorizations relevant for the user are analyzed and applied to the nodes.
3. *Transformation.* The transformation phase is a pruning of the DOM tree according to its labeling, based on the transformation presented in Section 4. Such a pruning is computed by means of a standard preorder 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. *Unparsing.* Finally, the fourth step is the generation of a valid XML document in text format, simply by unparsing (again, by means of a standard component) the pruned DOM tree computed by the previous step. Once again, this step is performed externally when the access control process is executed as a transformer module in the framework of a *Pipes and Filters* system

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

5.3 Performance and caching

In a complex server environment, performance and memory usage are critical issues. Moreover, the processing requirement for XML parsing, transformation, document processing and formatting are particularly

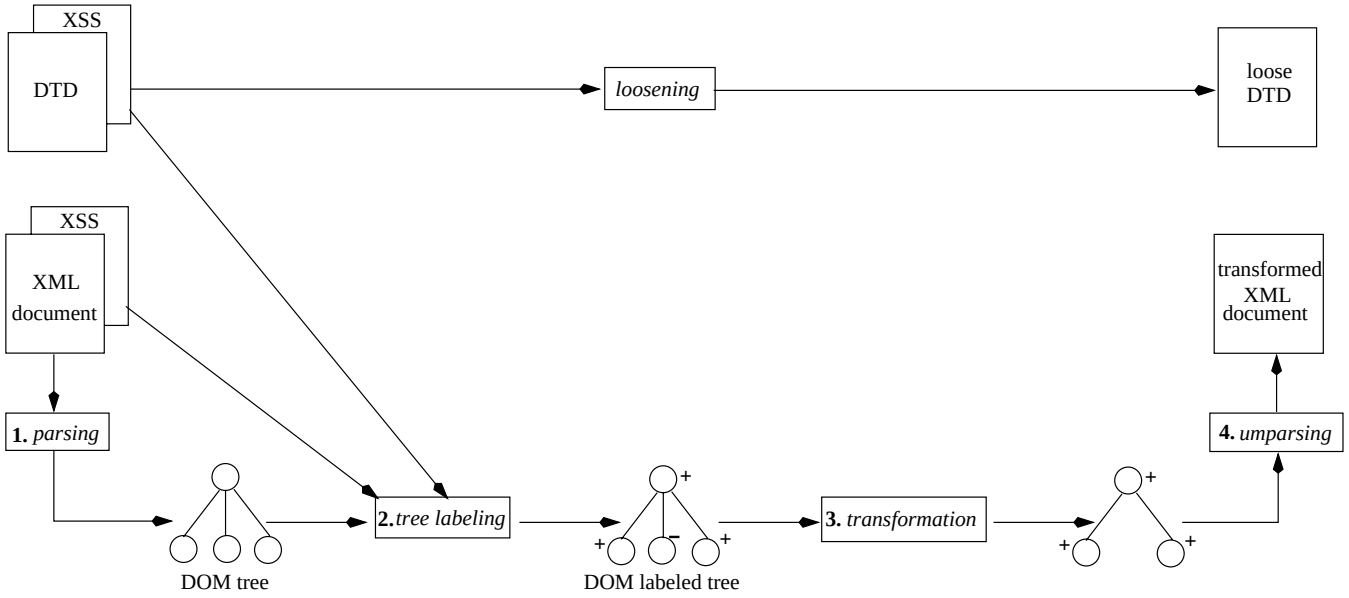

Figure 5: Document transformations by the security processor

heavy. For this reason, a special cache system is needed, in order to cache dynamically created pages. Caches of this kind are already available for XSLT processors which store their stylesheets in a pre-parsed form [12]. A cache for labelled documents is an important part of our system. When the request comes, the cache is searched. If an instance of the requested document for the same subject is found in the cache, then the cache copy is served. Otherwise, the document is parsed, labelled, transformed, unparsed and sent to the client; also, the transformed document is stored into the cache. Whenever authorizations are changed the whole cache is emptied. This technique allows dynamically generated pages (for example, XML documents created by querying a database) to be transformed and cached. Assuming that the frequency of requests is higher than that of resource changes, the cache may greatly reduce the total server load. The efficiency gain is particularly relevant when authorizations are specified with respect to a limited number of groups, as it may be the case for Internet-based servers. Moreover, the cache system can be based on a persistent object storage system which is able to save stored objects in a persistent state that outlives the module execution. This technique can be effectively used for pages that are very expensive to generate and last very long without changes, such as compiled server pages.

6 Related Work

Conventional HTML tagging is aimed at defining page rendering and is seldom if ever related to information granulation. For this reason, access control mechanisms currently available for Web sites tend to be coarse-grained. For instance, the Apache Web server (`http://www.apache.org`) allows the specification of access control lists via a configuration file (`access.conf`) containing a 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 on the Web server's disk, it is possible to define authorizations on a directory basis; files belonging to the same directory are subject to the same authorizations. The specification of authorizations at the level of single file (i.e., Web pages) is quite awkward, while it is not possible to specify authorizations on portions of files. This limitation forces protection requirements to affect data organization at the file system level. Recent proposals addressing authorization enforcement in

the Web, addressing topics such as certificate management [9] and support of groups and roles [10] are not thought for XML, and, therefore consider whole documents as granule of protection. The proposal in [14] specifies authorizations at a fine granularity providing a 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 (`http://www.ietf.org/rfc/rfc2660.txt`), explicitly represent authorizations within the documents by using security-related HTML tagging. Every document may have associated security (meta)tags describing its access authorizations. However, due to HTML fundamental limitations, even this proposal cannot take into full consideration the information structure and semantics.

The Role of Encryption Since the advancement of public-key cryptography has solved most of the security problems in communication, it is interesting to explore authentication and encryption role in providing fine-grained security to XML documents. Indeed, some commercial products are becoming available (e.g., AlphaWorks' XML Security Suite [1]) providing fine-grained security features, such as element-wise encryption and digital signatures. A rather coarser solution has been proposed by DataChannel, whose DataChannel Server product (*http://www.datachannel.com*) links XML authentication to existing directory systems, supporting both Windows NT and Lightweight Directory Access Protocol 3 directories. DataChannel servers map each XML document to the requesting user's ID and then to the file system access control. Thanks to authentication, an encryption-based XML server knows what information can be sent to a user based on that user's access level, and employs element-wise encryption to prevent users without appropriate decryption keys to access the parts of the documents containing private information. However, encryption-based approaches unequally split security responsibilities between the connection protocol, the XML content, and the application processing the document, while the need for a standardization of access control is becoming well recognized for XML data. Moreover, some encryption-based techniques leave encrypted private information in the hands of unauthorized users, a design choice which may well prove unwise in the long run.

Server-side XML/XSL processing Much work has been done recently on server side XML/XSL processing, and several design and implementation techniques have been proposed to obtain efficient, scalable systems based on DOM representation. *Cocoon* [12] is a Web publishing system for the Apache Web server whose engine is loosely based on the *Reactor* design pattern [3]. It deals with server-side *requests*, obtained processing client's requests and augmenting them with all the information needed by the processing engine. The request indicates what client generated the request, what URI is being requested and what producer should handle the request. Producer modules handle the requested URI and produce XML documents. Since producers are pluggable, they work like subservlets for this framework, allowing site designers to define and implement their own producers. It's up to the producer implementation to define the function that produces the document from the request object. Our access control processor is designed to be smoothlessly integrated in server-side architectures like Cocoon's.

7 An example

We now illustrate an example of authorization specification and document transformation.

Data organization: DTD and documents
We consider the case of an organization maintaining information regarding its departments, members, and projects. Each department is composed of one or more divisions and is responsible to create an XML document for each of them. To provide a uniform representation of this information, these

```
<!ELEMENT division (about_div,res_activity*,seminar*)>
<!ELEMENT about_div (member+,contact)>
<!ELEMENT member (name,position,e-mail?)>
<!ELEMENT e-mail (#PCDATA)>
<!ELEMENT contact (#PCDATA)>
<!ELEMENT res_activity (topic,description,project*)>
<!ELEMENT topic (#PCDATA)*>
<!ELEMENT description (#PCDATA)>
<!ELEMENT project (name,report*,fund*)>
<!ELEMENT fund (sponsor,amount)>
<!ELEMENT sponsor (#PCDATA)*>
<!ELEMENT amount (#PCDATA)*>
<!ELEMENT report (title,author+,text)>
<!ELEMENT title (#PCDATA)*>
<!ELEMENT author (#PCDATA)*>
<!ELEMENT seminar (date,title,speaker+)>
<!ELEMENT text (#PCDATA)*>
<!ATTLIST division   name CDATA #REQUIRED>
<!ATTLIST seminar   category (public|internal) #REQUIRED>
<!ATTLIST project   domain (public|private) #REQUIRED>
<!ATTLIST report   code ID #REQUIRED>
```

Figure 6: An example of DTD

XML documents must be valid with respect to a DTD defined by the organization. We consider DTD `http://www.acme.com/dtd.xml` reported in Figure 6. According to the DTD, each `division` is characterized by general information about it (`about_div` element), its current research activities (`res_activity` element), and `seminar`s. The `about_div` element includes information about the division `member`s and how to contact the division (`contact` element). The `res_activity` element contains the `topic` of the research, a `description`, and a set, possibly empty, of related `project`s. Seminars, which can be open to everybody or restricted to the division members, are characterized by a `date`, `title`, and one or more `speaker` elements. Each member of the division has a `name`, `position`, and `e-mail` address. Projects are described by a `name`, the `fund` to which the project expenses must be charged, and by zero or more `report` elements with `title` and `author` elements belonging to them. Funds are characterized by `sponsor` and `amount` elements. Attributes of elements are defined in the attribute list declarations. Element `division` has a `name` identifying the division. Element `seminar` has a `category` attribute used to make a distinction between seminars open to all (i.e., `category` = 'public') and seminars restricted to the division members (i.e., `category` = 'internal'). Element `project` has a required attribute `domain` representing the project visibility (public vs private). Finally, element `report` has an attribute `code` used as an identifier for the report.

Among the departments of the organizations is the CS Department which includes division `Security`. The information about this division is represented in the XML document `http://www.acme.com/sec.xml` illustrated in Figure 7.

Authorization specification

We now discuss some protection requirements that the acme organization and the CS department may need to express and illustrate how they are translated into authorizations of the form considered by our

```
<division name = "Security">
   <about_div>
    <member>
      <name>  Bob </name>
      <position> Computer Scientist </position>
      <e-mail> bob@acme.com </e-mail>
    </member>
    <member>
      <name>  Tom </name>
      <position> Software Engineering </position>
      <e-mail> tom@acme.com </e-mail>
    </member>
    <contact>
       Security Div. - 180 Lane St. - 81231 New Park
    </contact>
   </about_div>
  <res_activity>
    <topic>  Web security </topic>
    <description> The purpose of ... </description>
    <project domain = "private">
      <name>  Access Control </name>
      <fund>
        <sponsor> IT  </sponsor>
        <amount>  10000 </amount>
      </fund>
      <report code ="R1-99">
        <title> A new access control model  </title>
        <author> Sam  </author>
        <author> Ron </author>
        <text> ..... </text>
      </report>
    </project>
    <project domain = "public">
      <name>  Cryptography </name>
      <report code ="R2-99">
        <title> The study of encryption   </title>
        <author> Steve </author>
        <text> ..... </text>
      </report>
    </project>
  </res_activity>
  <seminar category="internal">
    <date> Tues., June 8  </date>
    <title> Safe statistics  </title>
    <speaker> Jan </speaker>
  </seminar>
  <seminar category="public">
    <date> Thurs., July 15 </date>
    <title> UML  </title>
    <speaker> Karen </speaker>
  </seminar>
</division>
```

Figure 7: An example of XML document valid with respect to the DTD in Figure 6

system. In the following , for the sake of simplicity, relative URIs (`http://www.acme.com` is the base URI) in the authorizations.

Organization's policy Specified at the DTD level – applicable to all the divisions of all departments of the organization.

1. Information about the name of members of any division in any department is publicly accessible; unless otherwise stated by the specific departments.
   `<<Public,*,*>,dtd.xml:/division/about_div/member/name,read,+,local,->`
2. Information about the name of public projects *must* be publicly accessible.
   `<<Public,*,*>,dtd.xml:/division/res_activity/project[./@domain="public"]/name, read,+,local,hard>`
3. Information about report of public projects *must* be publicly accessible.
   `<<Public,*,*>,dtd.xml:/division/res_activity/project[./@domain="public"]/report, read,+,recursive,hard>`

Computer Science department's policy Specified at the DTD level and instance level to complement or override the organization's policy.

4. Information about members of any division in the department is accessible to all members of the organization (`OrgMembers` group) unless otherwise stated by the organization.
   `<<OrgMembers,*,*>,dtd.xml:/division/about_div/member,read,+,recursive,soft>`
5. Information on funds of any division is accessible only to the members of `Admin` group connected from network `145.*`.
   `<<Admin,145.*,*>,dtd.xml:/division//fund,read,+,recursive,->`
   `<<Public,*,*>,dtd.xml:/division//fund,read,-,recursive,->`
6. Information about public seminars of any division is publicly accessible.
   `<<Public,*,*>,dtd.xml:/division/seminar[./@category="public"],read,+,recursive,->`
7. Information about seminars of the `Security` division is accessible only to users connected from network `145.*`.
   `<<Public,145.100.*,*>,sec.xml:/division/seminar,read,+,recursive,->`
   `<<Public,*,*>,sec.xml:/division/seminar,read,-,recursive,->`
8. Topics and description of research activities of the `Security` division are publicly accessible.
   `<<Public,*,*>,sec.xml:/division/res_activity/topic,read,+,recursive,->`
   `<<Public,*,*>,sec.xml:/division/res_activity/description,read,+,recursive,->`
9. Contact information about the `Security` division is publicly accessible unless otherwise stated by the organization.
   `<<Public},*,*>,sec.xml:/division/about_div/contact,read,+,local,soft>`
10. `Bob` cannot access information about the `Security` division projects.
    `<<Bob},*,*>,sec.xml:/division//project,read,-,recursive,->`
11. Information about projects can be accessed by members of the `Security` division when connected from hosts in the domain `*.com`.
    `<<Security,*,*.com>,sec.xml:division//project,read,+,recursive,->`

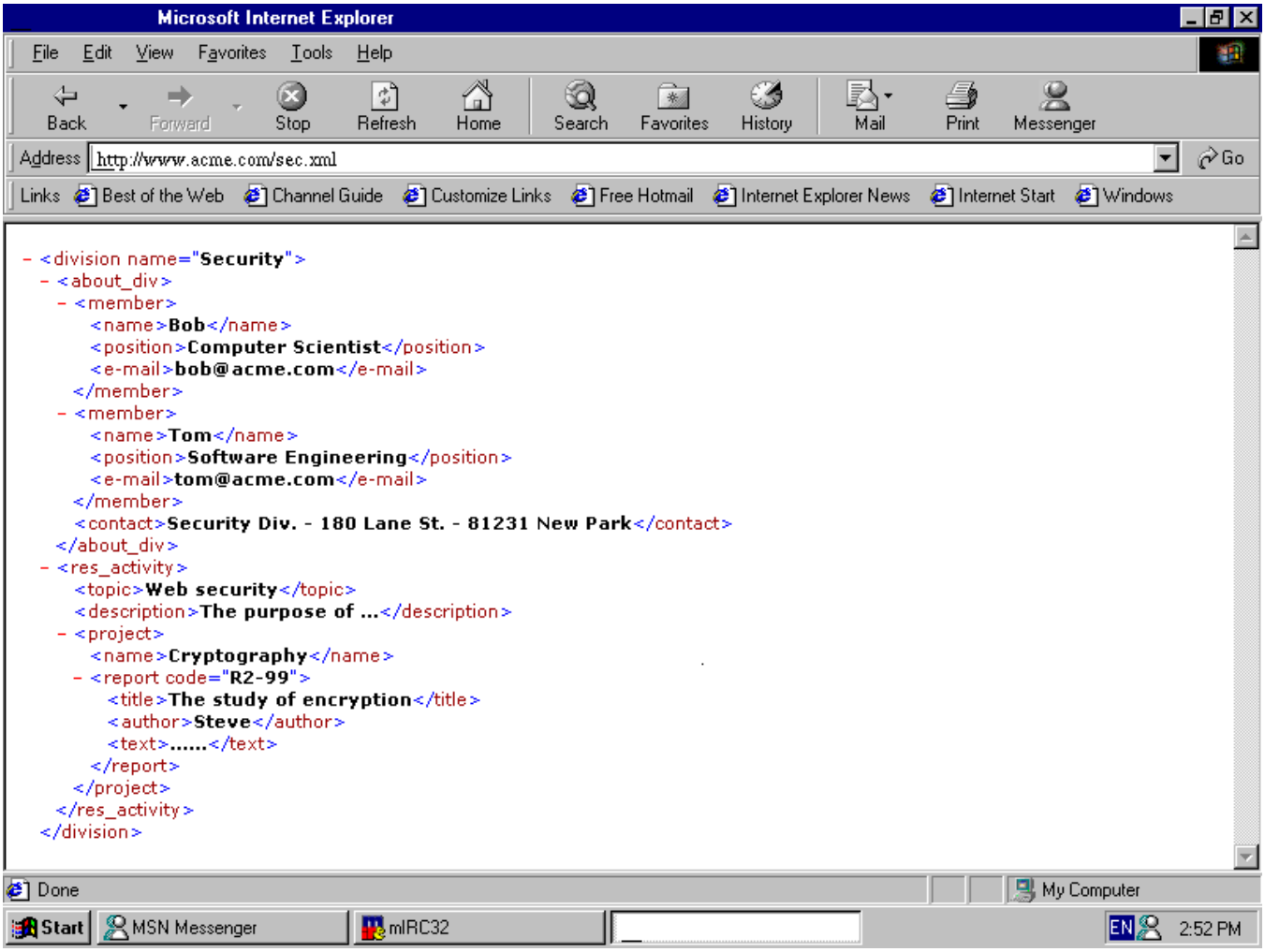

Figure 8: An example of view on the document in Figure 7

Document view

We now illustrate an example of document view visible to a requester in obedience to the authorizations specified. Consider a request to read the document `http://www.acme.com/sec.xml` describing the `Security` division (Figure 7). The request is submitted by user `Bob`, who is a member of the `Security` group, connected from machine `cslab.uniacme.edu` with numeric IP `150.100.80.3`. According to DTD-level authorizations 1 and 4, `Bob` can access information about the members of the division. According to document-level authorization 10, `Bob` cannot access information on projects. However, for public projects, this denial is overridden by hard authorizations 2 and 3 stated by the organization. Finally, `Bob` cannot access seminars information, since this is visible only to connections from network `145.*` (authorizations 7). The resulting view on the document of Figure 7 as returned to Bob is illustrated in Figure 8.

8 Conclusions

We have presented an access control system providing fine-grained access control for XML documents. The approach proposed is focused on enforcing and resolving fine grained authorizations with respect to the data model and semantics. Although presented in association with a specific approach to authorization

specification and subject identification, as supported in the current prototype, its operation is independent from such approaches and could then be applied in combination with different admninistrative policies. For instance, it can be combined with the treatment of roles [15, 21] and of authentication/authorization certificates [7, 9]. We are currently exploring such extensions.

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] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture - A System of Patterns*. Wiley and Sons Ltd., 1996.

[4] S. Castano, M.G. Fugini, G. Martella, and P. Samarati. *Database Security*. Addison-Wesley, 1995.

[5] 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.

[6] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Securing XML documents. In *Proc. of the 2000 International Conference on Extending Database Technology (EDBT2000)*, Konstanz,Germany, March 2000. (to appear).

[7] B. Gladman, C. Ellison, and N. Bohm. Digital signatures, certificates and electronic commerce. http://www.clark.net/pub/cme/html/spki.html.

[8] 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.

[9] J. Kahan. WDAI: A simple World Wide Web distributed Authorization infrastructure. In *Proc. of the 8th Int. World Wide Web Conference*, May 1999.

[10] S. Lewontin and M.E. Zurko. The DCE project: Providing authorizations and other distributed services to the world-wide web. In *Proc. of the 2nd World Wide Web Conference*, October 1994. http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/Security/lewontin/Web_DCE_Conf_94.html.

[11] 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.

[12] S. Mazzocchi. *Cocoon User Manual*. http://www.apache.org/java/cocoon.

[13] J. D. Moffett and M. Sloman. Policies hierarchies for distributed systems management. *IEEE Journal of Selected Areas in Communications*, 11(9):1404–1414, 1993.

[14] P. Samarati, E. Bertino, and S. Jajodia. An Authorization Model for a Distributed Hypertext System. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):555–562, August 1996.

[15] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[16] World Wide Web Consortium (W3C). *Document Object Model (DOM) Level 1 Specification Version 1.0*, October 1998. http://www.w3.org/TR/REC-DOM-Level-1.

[17] World Wide Web Consortium (W3C). *Document Object Model (DOM) Level 2 Specification Version 1.0.*, September Working Draft 1999. http://www.w3.org/TR/WD-DOM-Level-2.

[18] World Wide Web Consortium (W3C). *Extensible Stylesheet Language (XSL) Specification*, April 1999. http://www.w3.org/TR/WD-xsl.

[19] World Wide Web Consortium (W3C). *Resource Description Framework (RDF) Model and Syntax Specification*, February 1999. http://www.w3.org/TR/REC-rdf-syntax.

[20] World Wide Web Consortium (W3C). *XML Path Language (XPath)*, November 1999. http://www.w3.org/TR/xpath.

[21] M. E. Zurko, R. Simon, and T. Sanfilippo. A user-centered, modular authorization service built on an RBAC foundation. In *Proc. of the 20th IEEE Symposium on Security and Privacy*, pages 57–71, Oakland, May 1999.

Vitae

- **Ernesto Damiani** holds a Laurea Degree in Ingegneria Elettronica from Università di Pavia and a PhD degree in Computer Science from Università di Milano. He is currently an assistant professor at the campus located in Crema of Università di Milano. His research interests include distributed and object oriented systems, semi-structured information processing and soft computing.

- **Sabrina De Capitani di Vimercati** is an Assistant Professor at Dipartimento di Elettronica per l' Automazione of the University of Brescia. Her research interests are in the area of information security, databases, and information systems. She has been an international fellow in the Computer Science Laboratory at SRI, CA (USA). She is co-recipient of the ACM-PODS'99 Best Newcomer Paper Award.

- **Stefano Paraboschi** is an associate professor at the Dipartimento di Elettronica e Informazione of Politecnico di Milano. He received the Laurea Degree in Ingegneria Elettronica in 1990, and a PhD in Ingegneria Informatica in 1994, both from Politecnico di Milano. His main research interests are in the area of databases, with a focus on active databases, data warehouses, and the construction of data-intensive Web sites. He is the author, together with Paolo Atzeni, Stefano Ceri, and Riccardo Torlone, of the book "Database Systems: Concepts, Languages and Architectures" (McGraw-Hill 1999).

- **Pierangela Samarati** is an Associate Professor at the Department of Computer Science of the University of Milan. Her main research interests are in data and application security. She has been Computer Scientist in the Computer Science Laboratory at SRI, CA (USA). She has been a visiting researcher at the Computer Science Department of Stanford University, CA (USA), and at the ISSE Department of George Mason University, VA (USA). She is co-author of the book "Database Security," Addison-Wesley, 1995. She is co-recipient of the ACM-PODS'99 Best Newcomer Paper Award.