Generalized XML Security Views

Gabriel Kuper
kuper@acm.org

Fabio Massacci
Fabio.Massacci@unitn.it

Nataliya Rassadko
Nataliya.Rassadko@dit.unitn.it

Università di Trento
Via Sommarive 14, 38050 Povo, Trento, Italy

ABSTRACT

We investigate a generalization of the notion of XML security view introduced by Stoica and Farkas [17] and later refined by Fan et al. [8]. The model consists of access control policies specified over DTDs with XPath expression for data-dependent access control policies. We provide the notion of *security views* for characterizing information accessible to authorized users. This is a transformed (sanitized) DTD schema that can be used by users for query formulation and optimization. Then we show an algorithm to materialize "authorized" version of the document from the view and an algorithm to construct the view from an access control specification. We also propose a number of generalizations for security policies [1].

Categories and Subject Descriptors

H.2.7 [**Database Administration**]: Security, integrity and protection—*Access control*

General Terms

Algorithms, Security

Keywords

XML access control, XML views, XPath

1. INTRODUCTION

XML [3] has become the prime standard for data representation and exchange on the Web. In light of the sensitive nature of many business data applications, this also raises the issue of security in XML and the selective exposure of information to different classes of users based on their access privileges.

To address this issue we need simple, powerful, fine grained authorization mechanisms that

1. can control access to both content and structure;
2. can be enforced without annotating the entire document;
3. still provide a "sanitized" schema information to users.

While specifications and enforcement of access control are well understood for traditional databases [7, 13, 15, 16], the study of security for XML is less established. Although a number of security models are proposed for XML [2, 4, 6, 12, 14], these models do not meet criterion 3 above and, to a lesser extent, criterion 2. More specifically, these proposed models enforce security constraints at the document level by fully annotating the entire XML document/database [4, 2, 6]; these require expensive view materialization, and complicate the consistency and integrity maintenance. To overcome this limitations, the notion of XML security views was initially proposed by Stoica and Farkas [17] and later refined by Fan et al. [8].

The most important limitation of the mainstream models is the lack of support for authorized users to query the data: they either do not provide schema information of the accessible data, or expose the entire original DTD (or its loosened variant). In both cases, the solution is hardly practical for large and complex documents. Furthermore, fixing the access control policies at the instance level without providing or computing a schema, makes it difficult for the security officer to understand how the authorized view of a document for a user or a class of users will actually look like.

On the other side, revelation of excessive schema information might lead to security breaches: an unauthorized user can deduct or infer confidential information via multiple queries (essentially if the authorization specifications are not closed under intersection) and analysis of the schema even if just accessible nodes are queried.

1.1 Our Contribution.

We generalize the notion of XML security views to arbitrary DAG DTDs and to conditional constraints expressed in a very expressive XPath fragment. For each view, a security specification is a simple extension of the document DTD D with security annotations and security policies exploited to obtain full annotation from partial one. This specification has the advantage that can be easily implemented with little or no modification to state-of-the-art DTD parsers and offer security officers an intuitive feeling of the actual look of sanitized document.

From the specification, we derive a security view V consisting of a *view DTD* D_v and a function σ defined via XPath queries. The view DTD D_v shows only the data that is accessible according to the specification. The view is provided to the users so that they can formulate their queries over the view. The function σ is withheld from the users, and is used to extract accessible data from the actual XML documents to populate a structure conforming to D_v.

Query optimization can then be performed by users (using security view) and then by the system (by expanding and optimizing

[1] An extended version of this paper can be found at http://www.dit.unitn.it/∼rassadko/publications/kupe-mass-rass-04-long.pdf

the selection function). Thus, it is no longer necessary to process an entire document and only relevant data is retrieved. Moreover, the users can only access data via D_v, and no information beyond the view can be inferred from (multiple) queries posed on D_v.

More specifically, the main contributions of the paper include:

- A refined version of access policies over XML documents using conditional annotations at DTD level;
- A notion of security view that enforces the security constraints at the schema level and provides a view DTD characterizing them;
- An efficient algorithm for materializing security views, which ensures that views conform to view DTDs;
- An algorithm for deriving a security view from a specification of security annotations.

1.2 Plan of the paper

The rest of the paper is organized as follows. First we present preliminary notions on XML and XPath in Sec. 2. In Sec. 3 we provide a motivating example. Next we introduce the notion of security specification (Sec. 4) and the notion of view (Sec. 5). We show how to materialize a view and that using views is equivalent to annotating directly the document (Sec. 6). In Sec. 7 we describe classification of security policies with respect to consistency and completeness properties. Some extensions of our model are outlined in Sec. 8. Next we present implementation issues (Sec. 9). Finally we conclude the paper in Sec. 10.

2. A PRIMER ON XML AND XPATH

We first review DTDs [3] and XPath [5] queries.

Definition 2.1: A DTD D is a triple $(Ele, P, \texttt{root})$, where Ele is a finite set of *element types*; `root` is a distinguished type in Ele, and P is a function defining element types such that for each A in Ele, $P(A)$ is a regular expression over $Ele \cup \{\texttt{str}\}$, where `str` is a special type denoting PCDATA, We use ϵ to denote the empty word, and "$+$", "$,$", and "$*$" to denote disjunction, concatenation, and the Kleene star, respectively. We refer to $A \rightarrow P(A)$ as the *production* of A. For all element types B occurring in $P(A)$, we refer to B as a *subelement type* (or a *child type*) of A and to A as a *generator* (or a *parent type*) of B. $\Box$

We assume that DTD is non-recursive, i.e., that the graph has no cycles. Sec. 8 discusses this limitation.

Definition 2.2: An XML tree T *conforms to* a DTD D iff

1. the root of T is the unique node labelled with `root`;
2. each node in T is labelled either with an Ele type A, called an A *element*, or with `str`, called a *text node*;
3. each A element has a list of children of elements and text nodes such that their labels form a word in the regular language defined by $P(A)$;
4. each text node carries a `str` value and is a leaf of the tree.

We call T an *instance* of D if T conforms to D. $\Box$

Example 2.1: An XML database consists of a list of *applications* for PhD/MS program. Each application is initiated by a student described via *student-data* with an attribute *id* uniquely identifying student and representing student's login name. *Student-data* is composed of *name*, desired *degree* (PhD or MS) *department*, and *waiver*. The latter field may take values "true" or "false" and means that student does (does not) waive his/her right to inspect the content of recommendation letters. Application is supported by several letters of recommendation (*recomm-letter*), some of them can be classified as *unreliable* under some *reason*. Each letter has *letter* body and is provided by a separate *evaluator* having *name*, *title* and *institution* attributes. Evaluator places comments on applicant's skills in *free-text* field, which is either *PDF* or *TXT* file, and rates applicant's *English* proficiency, achievements during *MS* program and possible contribution in *PhD* program. Letters of recommendation are reviewed by admission committee and are assigned to a category *favorable* or *unfavorable* depending on the context.

The corresponding DTD is depicted on Fig. 1 and we show below some of the rules of this DTD. $\Box$

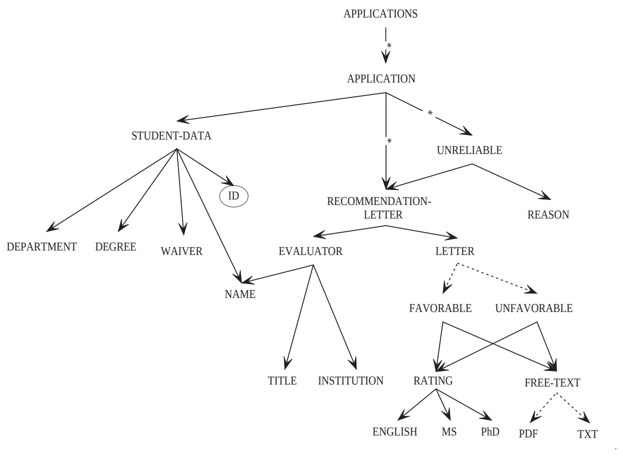


Figure 1: The graph representation of the document DTD D

```
application → (student-data,
   recomm-letter*, unreliable)
letter      → (favorable|unfavorable)
unreliable  → (recomm-letter, reason)
```

We consider a class of XPath queries, which corresponds to the CoreXPath of Gottlob et al. [11] augmented with the union operator and atomic tests and which is denoted by Benedict et al. [8] as $\mathcal{X}$.

The XPath axes we consider as primitive are `child`, `parent`, `ancestor-or-self`, `descendant-or-self`, `self`. We denote by θ one of those primitive axes and by θ^{-1} its inverse.

Definition 2.3: An XPath expression in $\mathcal{X}$ is defined by the following grammar:

$$\begin{array}{rcl}
\langle xpath \rangle & ::= & \langle path \rangle \ \mid \ `/` \langle path \rangle \\
\langle path \rangle & ::= & \langle step \rangle \, (`/` \langle step \rangle)^* \\
\langle step \rangle & ::= & \theta \ \mid \ \theta `[` \langle qual \rangle `]` \ \mid \ \langle path \rangle ` \cup ` \langle path \rangle \\
\langle qual \rangle & ::= & A \ \mid \ ` * ` \ \mid \ op\ c \ \mid \ \langle xpath \rangle \ \mid \\
 & & \langle qual \rangle \ \texttt{and} \ \langle qual \rangle \ \mid \ \langle qual \rangle \ \texttt{or} \ \langle qual \rangle \ \mid \\
 & & \texttt{not} \ \langle qual \rangle \ \mid \ `(` \langle qual \rangle `)`
\end{array}$$

where θ stands for an axis, c is a `str` constant, A is a label, op stands for one of $=, <, >, \leq, \geq$. The result of the $qual$ production is called *qualifier* and is denoted by q. We denote by $\mathcal{X}_{NoTest}$ the fragment build without the $op\ c$ test. $\Box$

We ignore the difference between $xpath$ and $path$ and we denote both with p, we abbreviate `self` with ϵ, `child`$[A]/p$ with A/p, `descendant-or-self`$[A]/p$ with $//A/p$, $q[op\ c]$ with $q\ op\,c$ and $p = p_1/p_2$, where p_2 is $//p_2'$, is written p as $p_1//p_2'$. The ancestor axis is abbreviated as $../$.

$$
\begin{gathered}
\mathcal{S}_{\rightarrow}\left[|/p|\right](N) = \mathcal{S}_{\rightarrow}\left[|p|\right](\{\texttt{root}\}) \\
\mathcal{S}_{\rightarrow}\left[|\theta[q]|\right](N) = \theta(N) \cap \mathcal{E}\left[|q|\right] \\
\mathcal{S}_{\rightarrow}\left[|\theta[q]/p|\right](N) = \theta(\mathcal{S}_{\rightarrow}\left[|p|\right](N)) \cap \mathcal{E}\left[|q|\right] \\
\mathcal{S}_{\rightarrow}\left[|p_1 \cup p_2|\right](N) = \mathcal{S}_{\rightarrow}\left[|p_1|\right](N) \cup \mathcal{S}_{\rightarrow}\left[|p_2|\right](N) \\
\mathcal{S}_{\rightarrow}\left[|(p_1 \cup p_2)/p|\right](N) = \mathcal{S}_{\rightarrow}\left[|p_1/p|\right](N) \cup \mathcal{S}_{\rightarrow}\left[|p_2/p|\right](N) \\
\mathcal{S}_{\leftarrow}\left[|/p|\right] = \begin{array}{ll} \{n \text{ occurs } inT\} & \text{if } \texttt{root} \in \mathcal{S}_{\leftarrow}\left[|/p|\right] \\ \emptyset & \text{otherwise} \end{array} \\
\mathcal{S}_{\leftarrow}\left[|\theta[q]|\right]N = \theta^{-1}(N \cap \mathcal{E}\left[|q|\right]) \\
\mathcal{S}_{\leftarrow}\left[|\theta[q]/p|\right]N = \theta^{-1}(\mathcal{S}_{\leftarrow}\left[|p|\right] \cap \mathcal{E}\left[|q|\right]) \\
\mathcal{S}_{\leftarrow}\left[|p_1 \cup p_2|\right] = \mathcal{S}_{\leftarrow}\left[|p_1|\right] \cup \mathcal{S}_{\leftarrow}\left[|p_2|\right] \\
\mathcal{S}_{\leftarrow}\left[|(p_1 \cup p_2)/p|\right] = \mathcal{S}_{\leftarrow}\left[|p_1/p|\right] \cup \mathcal{S}_{\leftarrow}\left[|p_2/p|\right] \\
\mathcal{E}\left[|A|\right] = \mathcal{T}(A) \\
\mathcal{E}\left[|q_1 \texttt{and} q_2|\right] = \mathcal{E}\left[|q_1|\right] \cap \mathcal{E}\left[|q_2|\right] \\
\mathcal{E}\left[|q_1 \texttt{or} q_2|\right] = \mathcal{E}\left[|q_1|\right] \cup \mathcal{E}\left[|q_2|\right] \\
\mathcal{E}\left[|\texttt{not} q|\right] = \{n \text{ occurs in } T\} \setminus \mathcal{E}\left[|q_2|\right] \\
\mathcal{E}\left[|p|\right] = \mathcal{S}_{\leftarrow}\left[|p|\right]
\end{gathered}
$$

Figure 2: The Semantics of Operators

The semantics of XPath is obtained by adapting to our fragment the $\mathcal{S}_{\rightarrow}$, $\mathcal{S}_{\leftarrow}$, $\mathcal{E}$ operators proposed by Gottlob et al. [11] and is identical to proposal of Benedickt et al. [1]. Intuitively $\mathcal{S}_{\rightarrow}\left[|p|\right](N)$ gives all nodes that are reachable from a node in N using the path p. The $\mathcal{S}_{\leftarrow}\left[|p|\right]$ functions gives all nodes from which a path p starts to arrive to queried node. The $\mathcal{E}\left[|q|\right]$ function evaluates qualifiers and returns all nodes that satisfy q.

The θ-symbol stands for both the semantics and the syntax of axes. So given a set of nodes N of a document T, $\theta(N)$ returns the nodes that are reachable according the axis from a node in N. By $\mathcal{T}(A)$ we denote the set of nodes that have element type A. For the semantics of qualifier see [11].

The semantics of the other operators is shown in Fig. 2

3. A MOTIVATING EXAMPLE

The need to provide users with a schema-level security view is illustrated by the access control requirements in Example 3.1.

Example 3.1: The applicant can access only his/her own data located under field `student-data`. Access to fields `favorable` and `unfavorable` is forbidden, while visibility of `rating` and `free-text` is established according to the accessibility to field `letter`. The latter is accessible if the `waiver` is *true* (data-dependent access). Moreover, the applicant should not be aware of reliability of the recommendation letters as the leakage of this information to recommenders might lead to diplomatic incidents. □

How can such constraints be enforced? Cho et al. [4] and Bertino et al. [2] enforce these constraints directly on the XML document. Damiani et al. [6] express their security specifications as sets of XPath expressions. However they transform their XPath specifications into an annotation of the entire document. So we have systems that do specify how to restrict access at the *data level*.

An important question remains unanswered: what schema information should be provided to the user? To formulate and process queries, the user needs a schema describing the accessible data. One solution, suggested by Damiani et al. [6], is to *loosen* the original DTD (make forbidden nodes optional). In some cases it is unacceptable to expose even the loosened DTD to final user, since highly confidential information, such as "unreliable" letters, can be deduced anyway.

In traditional relational databases users access a *View* of the data and permissions are assigned to views [13, 15]. A user may be denied the knowledge of the existence of an attribute of a relational schema. What we need here is a view of the document (at the schema level) that the user can use for queries, but that hides not only data but also structural information.

We borrow from Stoica and Farkas [17] the notion of access control model for XML that specifies and enforces security constraints at the *schema* level. For the actual notation we refine and generalize the proposal from Fan et al. [8]: authorizations are defined on a document DTD by annotating element types with Y/N or XPath qualifiers, indicating their accessibility.

From such a specification we can then infer a *view DTD* D_v and a *selection function* σ defined via XPath queries. The view DTD D_v shows only the data that is accessible according to the specification. The function σ is withheld from the users, and is used to extract accessible data from the actual XML documents.

4. SECURITY SPECIFICATIONS

Here we present our access-control specification language.

Definition 4.1: A *authorization specification* S is a pair (D, ann), where D is a DTD, ann is a partial mapping such that, for each production $A \rightarrow P(A)$ and each child element type B in $P(A)$, $\mathsf{ann}(A, B)$, if defined, is an annotation of the form:

$$\mathsf{ann}(A,B) \quad ::= \quad \mathsf{Q}[q] \quad | \quad \mathsf{Y} \quad | \quad \mathsf{N}$$

where $[q]$ is a qualifier in our fragment $\mathcal{X}$ of XPath. A special case is the root of D, for which we define $\mathsf{ann}(root) = \mathsf{Y}$ by default. □

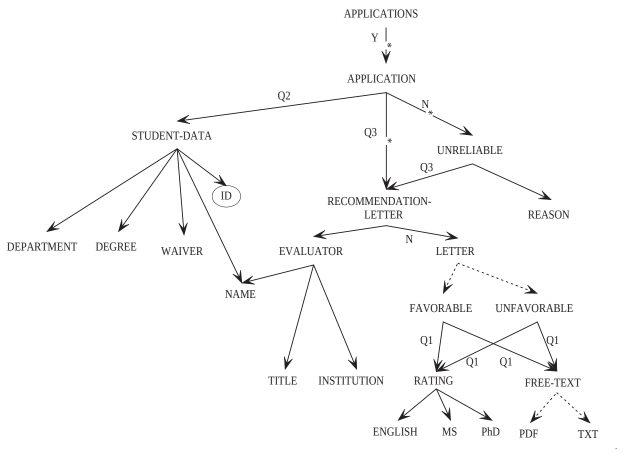
Intuitively, annotating production rule $P(A)$ of the DTD with an unconditional annotation is a security constraint expressed at the schema level: Y or N indicates that the corresponding B children of A elements in an XML document conforming to the DTD will always be accessible (Y) or always inaccessible (N), no matter what the actual values of these elements in the document are. If $\mathsf{ann}(A, B)$ is not explicitly defined, then B *inherits* the accessibility of A. On the other hand, if $\mathsf{ann}(A, B)$ is explicitly defined it may *override* the accessibility of B obtained via propagation.

At the data level, the intuition is the following: given an XML document T, the document is typed with respect to the DTD, and the annotations of the DTD are attached to the corresponding nodes of the document, resulting in a *partially annotated* XML document. Then we convert the document T to a *fully annotated* one by labelling all of the unlabelled nodes with Y or N. This is done by evaluating the qualifiers and replacing them by Y or N annotations, and then by a suitable policy for completing the annotation of the yet labelled nodes of the tree. When everything is labelled we remove all N-labelled nodes from T.

We should emphasize that semantics of qualifiers presented in this paper is *different* from that of in [8]. According to [8] a false evaluation of the qualifier is considered as "no label" and requires the inheritance of an access from ancestors, while we assume that once evaluated on the document, a qualifier is mapped to either Y or N. This greatly simplifies the intuition of the annotation for a security administrator.

Example 4.1: In Fig. 3(a) we show an example of security specification: paths to unconditionally allowed (forbidden) element types from their corresponding parents are marked with Y(N), and conditionally accessible element types are marked by qualifiers q_1, q_2 and q_3 (Fig. 3(b)). `$login` is a dynamic variable that is assigned at run time and depends on the student's login name. □

The construction of a fully annotated document depends heavily on the overall security policy that is chosen to get completeness [7].

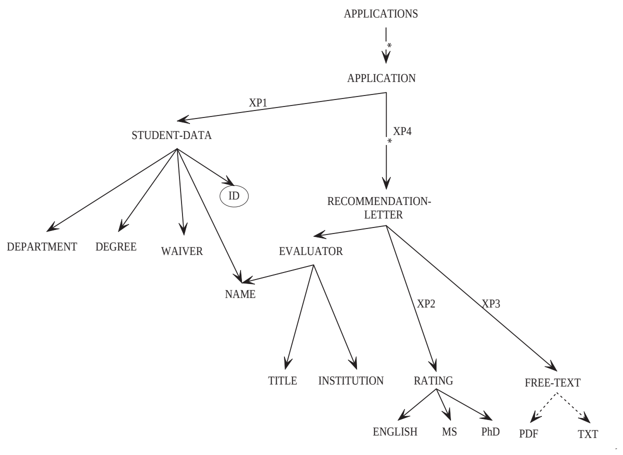


(a) Security annotation defined at DTD level

$$q_1 \doteq ancestor\text{::}\texttt{application}[./\texttt{student-data}[\texttt{@id} = \texttt{\$login}]/\texttt{waiver}/\texttt{text()} = \text{“}true\text{”}]];$$
$$q_2 \doteq ./\texttt{student-data}[./\texttt{@id} = \texttt{\$login}]]$$
$$q_3 \doteq ancestor\text{::}\texttt{application}[./\texttt{student-data}[\texttt{@id} = \texttt{\$login}];$$

(b) Meaning of security annotation qualifiers

Figure 3: Security annotation for competing student



(a) Security DTD view

$xp_1 = ./\texttt{student-data}[\texttt{@id} = \texttt{\$login}]$
$xp_2 = ./\texttt{letter}/(\texttt{favorable} \cup \texttt{unfavorable})/\texttt{rating}[q]$
$xp_3 = ./\texttt{letter}/(\texttt{favorable} \cup \texttt{unfavorable})/\texttt{free-text}[q]$
$xp_4 = ./(\epsilon \cup \texttt{unreliable})/\texttt{recomm-letter}$
where $q = \texttt{applications}/\texttt{application}/\texttt{student-data}/\texttt{waiver} = \text{“}true\text{”}$

(b) Meaning of XPath expressions

Figure 4: Security view for competing student

The top-down procedure that we describe next is the result of *most-specific-takes-precedence* policy which simply says that an unlabelled node takes the security label of its first labelled ancestor. Damiani et al. [6] use a *closed* policy as default: if a node is not labelled then label it as N. We return to this issue in Sec. 7, where we extend our model to allow alternate propagation techniques.

Definition 4.2: Let (D, ann) be a authorization specification and T a XML document conforming to D. The *authorized version* T_A of T according the authorization specification is obtained from T as follows:

1. Type T with respect to D and label nodes with ann values;
2. Evaluate qualifiers top down starting from the root and replace annotations by Y or N depending on the result;
3. For each unlabelled node, label it with the annotation of its nearest labelled ancestor;
4. Delete all nodes labelled with N from the result, making all children of a deleted node v into children of v's parent.

The annotation of the document, before deleting nodes in the last step, is called the *full annotation* of T. □

Example 4.2: Fig. 4(a) shows the security view generated from the security specification in Fig. 3(a). It hides confidential information. Fig. 4(b) lists some of the XPath annotations that are used to populate the appropriate element types form the original document DTD. □

Since T is a tree (a node has only one ancestor) it is not possible to have a conflict on labelling.

The pruning algorithm is more severe than that used by Damiani et al. [6] who delete only subtrees that are entirely labelled N, and delete only the data from nodes labelled N with some descendant labelled Y. As a consequence, the authorized view T_A no longer conforms to the original DTD D, not even to its loosened variant.

Example 4.3: In example 3.1 since `unreliable` is forbidden, the user should not even know that it exists. So he receives documents without it. □

More sophisticated ways of annotation are presented in [10, 18]. In particular, [10] uses XQuery to define derivation access control rules from the existing ones that are organized as XACL privilege triples <*object, subject, access-right*> [14]. The proposal of [18] is based on the conception of Role Graph Model merged with the conception of RBAC for object-oriented databases.

5. SECURITY VIEWS

We now turn to the enforcement of an access specification. To this end, we introduce the notion of *security view* which consists of two parts. The first part is a schema that is seen by the user, while the second part is a function that is hidden from the user, which describes how the data in the new schema should be derived from the original data. The intuition behind our approach is similar to that of security views for relational databases in multi-level security [13] and the notation is borrowed from [8].

Definition 5.1: Let D be a DTD. A *security view* for D is a pair (D_v, σ) where D_v is a DTD and σ is a function from pairs of element types such that for each element type A in D_v and element type B occurring in $P(A)$, $\sigma(A, B)$ is an expression in $\mathcal{X}$. □

Definition 5.2: Let $\mathcal{S} = (D_v, \sigma)$ be a security view. The semantics of $\mathcal{S}$ is a mapping from documents T conforming to D to documents $T_\mathcal{S}$ such that

1. $T_\mathcal{S}$ conforms to D_v
2. The nodes of $T_\mathcal{S}$ are a subset of the nodes of T, and their element type is unchanged.
3. For any node n of T which is in $T_\mathcal{S}$, let A be the element type of n, and let $B_1, \ldots, B_m$ be the list of element types that occur in $P(A)$. Then the children of n in $T_\mathcal{S}$ are
$$\bigcup_{1 \le i \le m} \mathcal{S}_\rightarrow [|\sigma(A, B_i)|](\{n\}) \ .$$
These nodes should be ordered according to the document order in the original document.

$T_\mathcal{S}$ is called the *materialized version* of T w.r.t. the view $\mathcal{S}$. □

Definition 5.3: A *valid* security view is one for which the semantics are always well-defined, i.e., if for every document T, its materialized version conforms to the security view DTD. □

Algorithm: MATERIALIZE
Input: a document T conforming to DTD D, a DTD View (D_v, σ)
Output: a materialized view $T_\mathcal{S}$ of T or $\perp$ (there is no such view)

```
 1: for all nodes $n$ of type $A$ in $T$ do
 2:     let $A \rightarrow P(A)$ the corresponding rule in $D_v$
 3:     for all $B$ occurring in $P(A)$ do
 4:         precompute $\mathcal{S}_\rightarrow [|\sigma(A \rightarrow P(A), B)|](\{n\})$
 5:     assign to $T_\mathcal{S}$ the root of $T$ and mark it as unprocessed
 6:     while there are unprocessed nodes in $T_\mathcal{S}$ do
 7:         select an unprocessed node $n$ of type $A$ with rule $A \rightarrow P(A)$ in $D_v$
 8:         make the nodes in
                $\bigcup_{B \text{ occurs in } P(A)} \mathcal{S}_\rightarrow [|\sigma(A \rightarrow P(A), B)|](\{n\})$
            in $T$ as unprocessed children of $n$ in $T_\mathcal{S}$
 9:         if a child of $n$ already occurs as a processed node in $T_\mathcal{S}$ then
10:             return $\perp$ (invalid view)
11:         make $n$ as processed
```

Figure 5: Algorithm MATERIALIZE

Not all views are valid: wrong typing, violated cardinality constraints, and other problems could be all causes of of a view to be invalid. However, the views that we construct from an annotated DTD are valid.

Security specification and views are related as follows.

Definition 5.4: Let (D, ann) be an authorization specification, and let $\mathcal{S} = (D_v, \sigma)$ be a security view for D. $\mathcal{S}$ is *data equivalent* to (D, ann) iff for every document T conforming to D, the materialized version $T_\mathcal{S}$ coincides with the authorized version T_A. □

Given a security view $\mathcal{S} = (D_v, \sigma)$ and document T conforming to a DTD D, we show how to construct $T_\mathcal{S}$ in Fig. 5.

Proposition 5.1: *If $\mathcal{S} = (D_v, \sigma)$ is a valid view for D, then the result of Algorithm* MATERIALIZE *is a document $T_\mathcal{S}$ that is the materialized version of T.* □

We now study the complexity of the algorithm. Let $f(n, d)$ be the complexity of evaluating an XPath expression of size n on a document of size d. Let $|\sigma|$ be the size of the largest XPath expression in the range of σ. Gottlob et al. [11] have shown that for CoreXPath (i.e. $\mathcal{X}$ without union and test) it is $f(|\sigma|, |T|) = |\sigma| \times |T|$. We extend their result to $\mathcal{X}$ without test and, with an increase in complexity, to the full $\mathcal{X}$ fragment.

Theorem 5.2: *Algorithm* MATERIALIZE *computes a materialized view in time $O(f(|\sigma|, |T|) \times |T|)$.* □

Lemma 5.3: *Every XPath query $p \in \mathcal{X}_{NoTest}$ over a document T can be evaluated in time $O(|p| \times |T|)$.* □

The naive implementation of union $\mathcal{S}_\rightarrow [|p_1/(p_2 \cup p_3)|](N)$ would lead to an exponential blow up. To avoid it we use a query DAG instead of a query tree.

Lemma 5.4: *Every XPath query $p \in \mathcal{X}$ over a document T can be evaluated in time $O(|p| \times |T|^2)$.* □

The test operation increases slightly the complexity because the computation of the $\mathcal{O}(c)$ operator requires the comparison of the `str` value c with the `str` value at every node of the tree.

Corollary 5.5: *Every valid DTD view whose annotations are in $\mathcal{X}$,* *respectively in $\mathcal{X}_{NoTest}$, can be materialized in $O(|\sigma| \times |T|^3)$, resp. $O(|\sigma| \times |T|^2)$, by Algorithm* MATERIALIZE. □

6. FROM AUTHORIZATION SPECIFICATIONS TO VIEWS

Our main result is to show how to construct a security view, given a document DTD and an authorization specification on it. The idea behind our algorithm is to eliminate qualifiers by expanding each qualifier into a union of two element types: one is the original element type, which is annotated Y, and the other is a new type, essentially a copy of the original type, which is annotated N. Since the tag of an element uniquely determines the type, it follows that new type names cannot match any nodes in a document that conforms to the original DTD. This is not a serious problem, as all of these new type names are deleted in the final security view.

The next step expands the annotation to a "full annotation". The notion of a full annotation was defined on annotated documents, and we showed that every document has a unique full annotation. At the schema level, however, this is not the case, as there may be several "paths" in the DTD that reach the same element type, each of which results in a different annotation. We use a similar technique to the way we handle qualifiers, i.e., we introduce new element types, and label the original one Y and the "copy" N. Finally, we delete all the element types that are labelled N, modifying the regular expressions and the σ functions correspondingly.

We show the algorithm ANNOTATE VIEW in Fig. 6 and algorithm BUILD VIEW in Fig. 7.

Definition 6.1: Let $\mathcal{S} = (D, \mathsf{ann})$ be an authorization specification. The DTD constructed by ANNOTATE VIEW algorithm is the *fully annotated* DTD corresponding to (D, ann). □

Theorem 6.1: *Let (D, ann) be a security specification where D is non-recursive. Algorithms* ANNOTATE VIEW *and* BUILD VIEW *terminate and produce a valid security view.* □

Theorem 6.2: *Let (D, ann) be a authorization specification, D is non-recursive, let (D_v, σ) the security view constructed by Algorithms* ANNOTATE VIEW *and* BUILD VIEW. *Let T be a document, T_A the authorized version of T and $T_{\mathcal{S}}$ the materialized version of T with respect to (D_v, σ). Then T_A is isomorphic to $T_{\mathcal{S}}$.* □

The proof is done by a top-down induction on T. The root of T is clearly in both T_A and $T_{\mathcal{S}}$. By induction, assume that n is of type A, and is in both T_A and $T_{\mathcal{S}}$. We must show that each child n in T_A is also a child of n in $T_{\mathcal{S}}$, and vice versa. The result then follows, as the order of the children of n is the same in both documents. Note, that it is essential that nodes in A should be ordered with the document order.

Let, therefore, m be a child of n in T_A, of type B. Assume, first, that m is a child of n in the original document T. Consider the fully annotated DTD (D_F, ann'). Since n is in $T_{\mathcal{S}}$, $\mathsf{ann}'(A) = \mathsf{Y}$. Since m is in T_A, it follows that $\mathsf{ann}(B)$ cannot be equal to N, and hence $\mathsf{ann}'(B) = \mathsf{Y}$, and so element type B is in D_v. Furthermore, if $\mathsf{ann}(B) = \mathsf{Q}[q]$, then q must hold at m.

We must show that m is in $\mathcal{S}_{\rightarrow} [|\sigma (A \rightarrow P(A), B)|] (\{n\})$. Let $p = \sigma(A, B)$. The algorithm ANNOTATE VIEW initially sets $p = B$ (step 2), may replace p by $B[q]$ in step 12, and may add additional disjuncts in step 2 of algorithm BUILD VIEW. In all cases m is clearly in the result.

Finally we consider the case when m is a descendant (not a child) of n in T and show that m is in $\mathcal{S}_{\rightarrow} [|\sigma(A, B)|] (\{n\})$. For the converse, we consider the case that m is a child of n in $T_{\mathcal{S}}$ and show that m is a child of n in T_A. The tricky bit is that at an intermediate step we introduce types (the one annotated with N) that have no correspondence with the document. To have them typed appropriately, we extend the notion of typing so that the new types will also match the corresponding old type from which they are generated.

```
Algorithm: ANNOTATE VIEW
Input: A authorization specification $(D, \mathsf{ann})$
Output: Fully annotated DTD $D$
 1: Initialize $D_v := D$ where ann is defined on $D_v$ as on $D$;
 2: for all production rules $A \rightarrow P(A)$ in $D_v$ do
 3:     for all element types $B$ occurring in $P(A)$ do
 4:         initialize $\sigma(A \rightarrow P(A), B) := B[\epsilon]$
 5:     //Below we will eliminate qualifier annotation
 6:     for all element types $B$ with $\mathsf{ann}(B) = \mathsf{Q}[q]$ do
 7:         add to $D_v$ a new element type $B'$ and a production rule
                $B' \rightarrow P(B')$
 8:         set $P(B') := P(B)$
 9:         for all element types $C$ occurring in $P(B')$ do
10:             $\sigma(B' \rightarrow P(B'), C) := \sigma(B \rightarrow P(B), C)$
11:         set $\mathsf{ann}(B) = \mathsf{Y}$ and $\mathsf{ann}(B') = \mathsf{N}$
12:         for all production rules $A \rightarrow P(A)$ do
13:             if $B$ occurs in $P(A)$ then
14:                 $\sigma(A \rightarrow P(A), B) := B[q]$;
15:                 $\sigma(A \rightarrow P(A), B') := B[\neg q]$;
16:                 replace $B$ by $B + B'$ in $P(A)$
17: //Below we will get fully annotated DTD $D$
18: while $\mathsf{ann}(B)$ of some element types $B$ is undefined do
19:     if all generators $A$ of $B$ have defined $\mathsf{ann}(A)$ then
20:         if all $\mathsf{ann}(A) = \mathsf{Y}$ then
21:             set $\mathsf{ann}(B) := \mathsf{Y}$;
22:         else if all $\mathsf{ann}(A) = \mathsf{N}$ then
23:             set $\mathsf{ann}(B) := \mathsf{N}$;
24:         else
25:             add to $D_v$ a new element type $B'$ and a production
                    rule $B' \rightarrow P(B')$
26:             set $P(B') := P(B)$
27:             for all element types $C$ occurring in $P(B')$ do
28:                 $\sigma(B' \rightarrow P(B'), C) := \sigma(B \rightarrow P(B), C)$
29:             set $\mathsf{ann}(B) = \mathsf{Y}$, $\mathsf{ann}(B') = \mathsf{N}$,
30:             for all generators $A$ of $B$ do
31:                 if $\mathsf{ann}(A) = \mathsf{N}$ then
32:                     replace $B$ with $B'$ in $P(A)$
```

Figure 6: Algorithm ANNOTATE VIEW

```
Algorithm: BUILD VIEW
Input: Fully annotated DTD $D$
Output: A security view $(D_v, \sigma)$
 1: for all element types $B$ with $\mathsf{ann}(B) = \mathsf{N}$ do
 2:     for all production rules $A \rightarrow P(A)$ do
 3:         if $B$ occurs in $P(A)$ then
 4:             for all $C$ that occurs in $P(B)$ do
 5:                 set
                        $\sigma(A \rightarrow P(A), C) :=$
                        $\sigma(A \rightarrow P(A), B) / \sigma(B \rightarrow P(B), C) \cup$
                        $\sigma(A \rightarrow P(A), C)$
 6:             replace $B$ by $P(B)$ in $P(A)$ if $B \rightarrow P(B)$ exists
                    and by $\epsilon$ otherwise
 7: $D_v$ consists of all the element types $A$ for which $\mathsf{ann}(A) = \mathsf{Y}$,
        with the $\sigma$ function restricted to these types.
```

Figure 7: Algorithm BUILD VIEW

The complexity of the algorithm follows from Theorem 6.3:

Theorem 6.3: *Let (D, ann) be a authorization specification for a non-recursive DTD, let P be size of the largest production rule in D. Let n_{Y} be the number of element types annotated with Y, and let n_{other} the number of element types otherwise annotated or not annotated. Then the size of the select function σ generated by the algorithm is bounded by $O(n_{other} \times |\mathsf{ann}|)$ and the size of the View DTD D_v is bounded by $O(n_{\mathsf{Y}} \times P^{n_{other}+1})$.* □

The above upper bound is tight as the following example shows:

Example 6.1: Consider DTD with the production $\mathtt{root} \rightarrow A_0$ and $A_i \rightarrow A_{i+1}A_{i+1}$ for $i = 0 \ldots n-1$ and where $\mathsf{ann}(A_0) = \mathsf{N}$, $\mathsf{ann}(A_n) = \mathsf{Y}$. Then the DTD View D_v has only one rule

$$\mathtt{root} \rightarrow \overbrace{A_n \ldots A_n}^{2^n\text{times}},$$

and the select function is $\sigma(\mathtt{root}, A_n) = A_0/\cdots/A_n$. □

7. OTHER SECURITY POLICIES

Our model is based on a specific policy, used for determining a complete authorization specification of a document based on a partial specification. This is the *most-specific-takes-precedence* policy [7]. Different applications may have different requirements, and we now look at alternative approaches:

Local Propagation Policy: "open", "closed", or "none";

Hierarchy Propagation Policy: "topDown", "bottomUp", or "none";

Structural Conflict Resolution: "localFirst", "hierarchyFirst", or "none";

Value Conflict Resolution: "denialTakesPrecedence", "permissionTakesPrecedence", or "none".

The Local Propagation Policy is similar to traditional policies for access control: in the case of "open", if a node is not labelled N then it is labelled by Y; in the case of "closed", a node not labelled Y is labelled by N.

The Hierarchy Propagation Policy specifies node annotation inheritance in the tree. In the case of "topDown", an unlabelled node with a labelled parent inherits the label of its parent. In the case of "bottomUp" an unlabelled node inherits the label from a labelled children. Note that the "bottomUp" case can result in conflicts, and they should be addressed by the Value Conflict Resolution Policy.

The Structural Conflict Resolution Policy specifies whether the local or hierarchy rule takes precedence ("localFirst" or "hierarchyFirst" respectively); while "none" means that the choice depends on the values and on the Value Conflict Resolution Policy. The latter specifies how to resolve conflicts for unlabelled nodes that are assigned different labels by the preceding rules: N always has precedence over Y ("denialTakesPrecedence"); Y always has precedence over N ("permissionTakesPrecedence"), and no choice ("noneTakesPrecedence").

Definition 7.1: A policy is *complete* and *consistent* if every partially annotated tree can be extend to a fully annotated tree. □

A comprehensive analysis of all possible policy combinations gives the theorem:

Theorem 7.1: *In Table 1, where * means "any", policies following the conditions of lines 1–7 are sound and complete, policies following the conditions of line 8 are not complete, policies following the conditions of lines 9–11 are not consistent.* □

8. EXTENSIONS

One restriction in our current proposal is related to nonrecursive DTDs. For authorization specification of recursive DTD it is possible to derive a fully annotated DTD by modifying step 18 of the algorithm ANNOTATE VIEW, but one cannot construct a select function in XPath, because XPath lacks the full Kleene-star operator. Using the present algorithm, we can obtain an approximate solution: *by stopping the modified* ANNOTATE VIEW *after a finite number of iterations of step 1 of* BUILD VIEW *we have a* secrecy preserving *view*.

The second extension is related to policies over XML documents expressed as XPath queries [6] tagged to principals. If the principal in the specification is matched to the actual requester then XPath queries are used to select the subset of nodes that are labelled with some security attribute. It is possible to translate that security specification into our framework.

9. IMPLEMENTATION

We have implemented a preliminary version of a Java tool that outputs a "sanitized" XML document, i.e. document that contains only permitted nodes and the DTD view.

Firstly, we use Xerces Java DOM parser [2] and Wutka DTD parser [3]. The latter we modified to distinguish security policy attributes located at root element and security annotations over the rest of DTD.

Then partially annotated DTD is extended to a full annotated one according to the algorithm ANNOTATE VIEW. Next we apply BUILD VIEW to produce D_v which is used to materialize view of XML document $T_{\mathcal{S}}$ according to the algorithm MATERIALIZE.

10. RELATED WORK AND CONCLUSIONS

A number of security models have been proposed for XML (see [9] for a recent survey). Specifying security constraints with XPath on top of document DTDs was discussed in [6]. The semantics of access control to a user is a specific view of the document determined by the XPath access-control rules. A view derivation algorithm is based on tree labelling. Issues like granularity of access, access-control inheritance, overriding, and conflict resolution are studied in [2, 6].

A different approach is explored in [4]. In a nutshell, access annotations are explicitly included in the actual element nodes in XML, whereas DTD nodes specify "coarse" conditions on the existence of security specifications in corresponding XML nodes. Only elements with accessible annotations appear in the result of a query.

Stoica and Farkas [17] proposed to produce single-level views of XML when conforming DTD is annotated by labels of different confidentiality level. The key idea lies in analyzing semantic correlation between element types, modification of initial structure of DTD and using cover stories. Altered DTD then undergoes "filtering" when only element types of the confidentiality lever no higher that the requester's one are extracted. However, the proposal requires expert's analysis of semantic meaning of production rules, and this can be unacceptable if database contains a large amount of schemas which are changed occasionally.

This paper elaborates on certain issues left open in [8]. In particular, we studied access control and security specifications defined over general DTDs in terms of regular expressions rather than normalized DTDs of [8]. Furthermore, we developed a new algorithm for deriving a security view definition from more intuitive access control specification (w.r.t. a non-recursive DTD) without introduc-

[2]http://xml.apache.org/xerces2-j/
[3]http://www.wutka.com/dtdparser.html

Table 1: Policy conditions

	hierarchy	local	structural conflict	value conflict	condition
1	topDown	≠none	hierarchyFirst	$*$	$*$
2	topDown	none	$*$	$*$	root is annotated
3	bottomUp	≠none	hierarchyFirst	≠none	$*$
4	bottomUp	none	$*$	≠none	leaves are annotated
5	$*$	≠none	localFirst	$*$	$*$
6	none	≠none	$*$	$*$	$*$
7	≠none	≠none	noneFirst	≠none	$*$
$\underline{8}$	none	none	$*$	$*$	$*$
$\underline{9}$	≠none	≠none	none	none	$*$
$\underline{10}$	bottomUp	$*$	hierarchyFirst	none	$*$
$\underline{11}$	bottomUp	none	≠hierarchyFirst	none	$*$

ing dummy element types, and thus preventing inference of sensitive information from the XML structure revealed by dummies.

We have presented a refined model for securing XML data, based on the novel notion of security views. A salient feature of our model is that it specifies and enforces access-control policies at the schema level. This yields an effective and efficient approach to enforcing security without materializing and maintaining views.

Acknowledgments. This project has been partially supported by the MIUR-FIBR project ASTRO and the MIUR-COFIN "Web-based management and representation of spatial and geographical data".

11. REFERENCES

[1] M. Benedikt, W. Fan, and G. M. Kuper. Structural properties of XPath fragments. In *Proceedings of the International Conference on Database Theory*, 2003.

[2] E. Bertino and E. Ferrari. Secure and selective dissemination of XML documents. *ACM Transactions on Information and System Security*, 5(3):290–331, 2002.

[3] T. Bray, J. Paoli, and C. M. Sperberg-McQueen. *Extensible Markup Language (XML) 1.0*. W3C, Feb. 1998.

[4] S. Cho, S. Amer-Yahia, L. Lakshmanan, and D. Srivastava. Optimizing the secure evaluation of twig queries. In *Proceedings of the International Conference on Very Large Data Bases*, 2002.

[5] J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendation. http://www.w3.org/TR/xpath, November 1999.

[6] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. A fine-grained access control system for XML documents. *ACM Transactions on Information and System Security*, 5(2):169–202, 2002.

[7] S. De Capitani di Vimercati and P. Samarati. Access control: Policies, models, and mechanism. In R. Focardi and F. Gorrieri, editors, *Foundations of Security Analysis and Design - Tutorial Lectures*, volume 2171 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.

[8] W. Fan, C.-Y. Chan, and M. Garofalakis. Secure XML querying with security views. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 587–598. ACM Press, 2004.

[9] I. Fundulaki and M. Marx. Specifying access control policies for XML documents with XPath. In *Proceedings of the 9th ACM symposium on Access control models and technologies*, pages 61–69. ACM Press, 2004.

[10] S. K. Goel, C. Clifton, and A. Rosenthal. Derived access control specification for XML. In *Proceedings of the 2nd ACM Workshop On XML Security*, pages 1–14. ACM Press, 2003.

[11] G. Gottlob, C. Koch, and R. Pichler. Efficient algorithm for processing XPath queries. In *Proceedings of the International Conference on Very Large Data Bases*, 2002.

[12] S. Hada and M. Kudo. XML Access Control Language: Provisional Authorization for XML Documents. http://www.trl.ibm.com/projects/xml/xacl/, 2000.

[13] T. F. Lunt, D. E. Denning, R. R. Schell, M. Heckman, and W. R. Shockley. The SeaView security model. *IEEE Transactions on Software Engineering*, 16(6):593–607, 1990.

[14] M. Murata, A. Tozawa, M. Kudo, and S. Hada. XML access control using static analysis. In *Proceedings of the 10th ACM conference on Computer and communication security*, pages 73–84. ACM Press, 2003.

[15] X. Qian. View-based access control with high assurance. In *Proceedings of the 15th IEEE Symposium on Security and Privacy*, pages 85–93. IEEE Computer Society Press, 1996.

[16] P. D. Stachour and B. Thuraisingham. Design of LDV: A multilevel secure relational database management system. *IEEE Transactions on Knowledge and Data Engineering*, 2(2):190–209, 1990.

[17] A. Stoica and C. Farkas. Secure XML views. In *Research Directions in Data and Applications Security, IFIP WG 11.3 Sixteenth International Conference on Data and Applications Security*, volume 256, pages 133–146. Kluwer, 2003.

[18] J. Wang and S. L. Osborn. A role-based approach to access control for XML databases. In *Proceedings of the 9th ACM symposium on Access control models and technologies*, pages 70–77. ACM Press, 2004.