

Query Rewriting Algorithm Evaluation for XML Security Views

Nataliya Rassadko

The University of Trento, via Sommarive 14, 38050 Povo(TN), Italy
rassadko@dit.unitn.it

Abstract. We investigate the experimental effectiveness of query rewriting over XML security views. Our 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 is used by users for query formulation. To avoid the overhead of view materialization in query answering, these queries later undergo rewriting so that they are valid over the original DTD schema, and thus the query answer is computed from the original XML data. We provide an algorithm for query rewriting and show its performance compared with the naive approach, i.e. the approach that requires view materialization.

Key words: query rewriting, XML views, XPath annotation, algorithm, evaluation, security

1 Introduction

Specification of access control models for XML data has been a fairly active field of research in recent years [5], [6], [7], [10], [12], [13], [14], [15], [16], [18], [20], [21], [24], [26], [32]. All this previous work (except [14], [21], [26]) enforces security constraints at the document level by fully annotating the entire XML document.

As a result, one major limitation of these 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 so-called “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 deduce or infer confidential information via multiple queries and analysis of the schema even if only the accessible nodes are queried.

To overcome this limitations, the notion of XML security views was initially proposed by Stoica and Farkas [32] and later refined by Fan et al. [14] and Kuper

et al. [21]. The basic idea is to provide a schema that describes the data that can be seen by the user, as well as a (hidden) set of XPath expressions that describe how to compute the data in the view from the original data.

In the current paper, we implement and test experimentally the performance of the security view model of [21]. To this end, we define a rewriting algorithm that takes a user query over the a security view, and rewrites the query into a query over the original database. We then compare the cost of evaluating this query with that of evaluating the original query over a materialized view of the data, and show that significant performance improvements.

The paper is organized as follows. In Sec. 2 we present preliminary notions on XPath and XML security views. The algorithm of query rewriting is described shortly in Sec. 3. Evaluation of rewriting algorithm is provided in Sec. 4. The discussion of related work is located in Sec. 5. Finally, we conclude the paper in Sec. 6.

2 Background

We first review the fragment of XPath [11] that may be used by a user in query formulation.

Definition 1. *An XPath expression in \mathcal{X} is defined by the following grammar:*

$$\begin{aligned}
\langle xpath \rangle &::= \text{'/?'} \langle path \rangle \mid \langle path \rangle (\text{' \cup ' } \langle path \rangle) * \\
\langle path \rangle &::= \langle step \rangle (\text{'/' } \langle step \rangle) * \\
\langle step \rangle &::= \langle test \rangle (\text{'[' } \langle qual \rangle \text{']' } *) \\
\langle test \rangle &::= \theta \text{' :: ' } A \mid \theta \text{' :: *' } \\
\langle qual \rangle &::= \langle xpath \rangle \mid \langle path \rangle \text{ op } c \mid \langle qual \rangle \text{ and } \langle qual \rangle \mid \\
&\quad \langle qual \rangle \text{ or } \langle qual \rangle \mid \text{not } \langle qual \rangle \mid \text{'(' } \langle qual \rangle \text{' }
\end{aligned}$$

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 filtering is called qualifier and is denoted by q .

We must note the semantic difference between $\langle xpath \rangle$ and $\langle path \rangle$: the former may contain unions, while the latter may not. This is because XPath 1.0 [11] does not allow unions in location steps. For the sake of readability, we ignore the syntactic difference between $xpath$ and $path$; we denote both with p . We also abbreviate **self** by ϵ , **child** $:: A/p$ with A/p , **descendant-or-self** $:: A/p$ by $//A/p$, and $p = p_1/p_2$ with $p_2 = //p'_2$ is written p as $p_1//p'_2$. The parent axis is also abbreviated as $../$.

The intuition behind XML security view is similar to that of multi-level security view for relational databases [23], [29], [31], [34], discretionary access control over relational databases [4] and object-oriented databases [8]. A typical approach to specifying and enforcing access control for traditional databases is to define views on which security permissions should be applied. For instance, Lunt et al. [22] showed how to use standard SQL queries to implement the SeaView multi-level secure database. In contrast, the hierarchical structure and

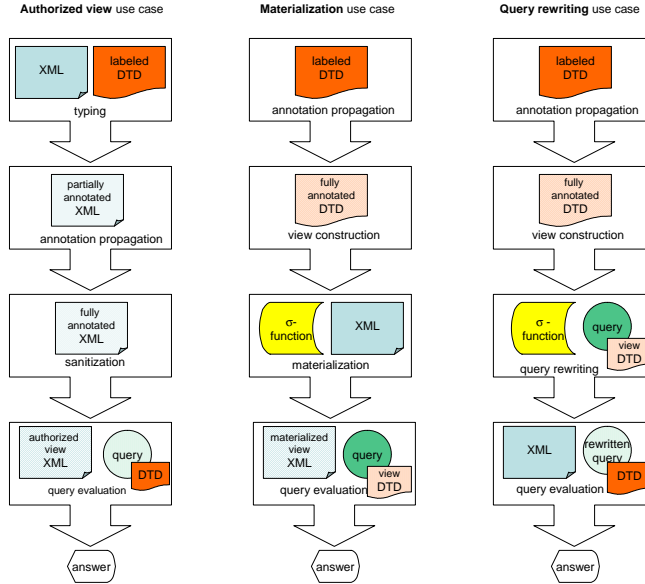


Fig. 1. Query answering use cases

the dependency (e.g., ancestor and descendants) of XML data as well as the presence of disjunction and recursion in DTDs make it impossible to define a security view via a single query. Moreover, they introduce new challenges in how to generate a view that conforms to a DTD view. The challenges were observed in [3], which showed that even for XML views of relational data, it is highly nontrivial to ensure that the views typecheck.

The insight of XML security view construction was found in [29] where it was demonstrated how to compute a full database labelling from a partial one implied by security views. From the XML viewpoint, a partial assignment of security labels to XML document nodes can be also extended to a full assignment. From the latter, it is easy to compute an XML analogue of relational view by means of “sanitization” operation that hides (e.g., deletes or encrypts) nodes with negative authorizations, but reveals their permitted children. The resulting XML tree is called *authorized*. This scenario is depicted on the left part of Fig. 1.

Unlike solutions presented in e.g., [5], [13], [15], [20], we employ schema-level policy enforcement, resulting in the DTD schema of permitted data (or in other words, DTD view like in [21], [32]) and hidden rules of *DTD-to-DTD view* transformation. This pair (DTD view, hidden rules) can be used to construct a *materialized* version of XML document by deleting forbidden nodes so that the *materialized view* be isomorphic to the *authorized view*. Finally, the *materialized view* is used for evaluation of user queries. We call such a query answering schema *view materialization use case*. It is conceptually represented on the central part of Fig. 1. There are two ways to implement this use case in

client-server architecture (where *client* tries to get an access to the XML document and possesses a schema of available data, and *server* stores the initial XML document and must satisfy requests of *client* to this document): server either stores every materialized view or recalculates a corresponding view on the fly every time client issues a query. In the former case, the integrity maintenance becomes unfeasible. In the latter case, the view materialization can be very time- and memory- consuming that is unacceptable in the presence of many users requesting the database simultaneously.

To overcome limitations of the *view materialization* use case, we developed a *query rewriting use case* when *user's* queries are answered without view materialization: we rewrite a user query formulated for the DTD view into an equivalent query formulated for the original DTD using *hidden rules* [30] and evaluate the resulting query over the original XML database. The schema of such an approach is shown on the right part of Fig. 1.

Hidden rules are defined for every edge of DTD view and describe the path between nodes of that edge in the original DTD. For example, if hidden rule $\sigma(A, B) = B$, node B is a child of node A in the original DTD; if $\sigma(A, B) = C[D]/B$, B is a grandchild of A in the original DTD and the parent of B should be C with a child D .

3 Query Rewriting Algorithm Description

The algorithm for query rewriting has two phases: query parsing and further translation of the parsed query into σ -functions. Query parsing phase implies that user query is represented as a tree of subqueries (*parse tree*) according to the grammar that we have shown in Def. 1. The graphic representation of a generic parse tree is show in Fig. 2. Namely, *basic block* is one of $\langle xpath \rangle$, $\langle path \rangle$, $\langle step \rangle$, $\langle qual \rangle$, or $\langle test \rangle$ which is $\theta::\lambda$ where λ is either an element type name or an asterisk *; *optional part* is self-explaining (i.e., $\langle xpath \rangle$ any consist of one or more

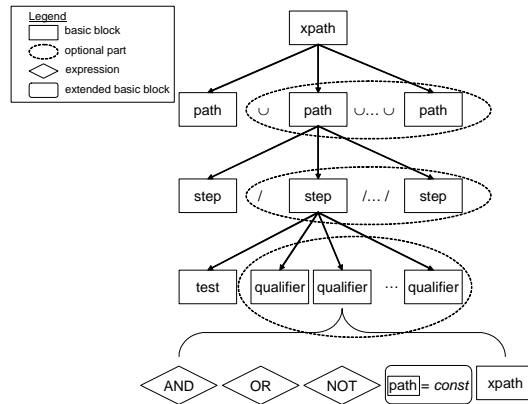


Fig. 2. Generic parse tree

$\langle path \rangle$ s that may consist of one or more $\langle step \rangle$ s that may include zero or more $\langle qual \rangle$ s); *extended basic block* is related to a qualifier of the form $\langle path \rangle = const$; *expression* is related to qualifiers including **and**, **or**, and **not** operators with operands as other qualifiers. If $\langle xpath \rangle$ or $\langle path \rangle$ occur in a qualifier, they are parsed as basic blocks according to Def. 1. Leaves of the parse tree are $\langle test \rangle$ s.

The translation of the parsed query starts from the leaves of the parse tree and moves up to the root $\langle xpath \rangle$. In particular, for each subquery p and some element A , the algorithm calculates $QR_{(p,A)}$ using $QUERY\ REWRITE(p_i, B_j)$, where p_i is a direct subquery (child in a parse tree) of p and B_j is a node reachable from A via p_i in D_v . At the same time, the algorithm calculates $reach(p, A)$ representing the set of nodes reachable from node A via the path p . To obtain a rewriting of the initial user query q , we invoke $QUERY\ REWRITE(q, root)$.

We do not provide here a formal presentation of $QUERY\ REWRITE$ algorithm since it was presented in [30]. Basically, the following rules characterize a process of query rewriting:

1. rewriting of $\langle xpath \rangle$ is a union of rewritten $\langle path \rangle$ s that constitute it;
2. rewriting of $\langle path \rangle$ is a rewriting every its step w.r.t. a $\langle test \rangle$ of a previous $\langle step \rangle$. Namely
 - $test_1/child::test_2 \rightarrow \sigma(test_1, test_2)$;
 - $test_1/parent::test_2 \rightarrow \sigma^{-1}(test_2, test_1)$;
 - $test_1/descendant-or-self::test_2 \rightarrow QUERY\ REWRITE(test_1, path(test_1, test_2))$,
 $path(test_1, test_2)$ is a path from $test_1$ to $test_2$ expressed by **child** axes;
 - $test_1/ancestor-or-self::test_2 \rightarrow QUERY\ REWRITE(test_1, path^{-1}(test_1, test_2))$,
 $path^{-1}(test_1, test_2)$ is a path from $test_1$ to $test_2$ expressed by **parent** axes.

Here, if $test_2$ is $*$ then $test_2$ is any node v that is in a corresponding relation with $test_1$. The resulting rewritten expression, then, is a union of rewritten expressions as if v where at the position of $test_2 = *$.

3. rewriting of $\langle test \rangle$ with $\langle qual \rangle$ is a rewriting of $\langle qual \rangle$ w.r.t. $\langle test \rangle$;
4. rewriting of **and/or/not** expressions is respectively **and/or/not** expression of rewritten subexpressions of the initial expression.

Here $\sigma^{-1}(A, B)$ is, basically, $\sigma(A, B)$ that is inverted with the help of **parent** axis so that to represent a path from B to A (which may be an ancestor in the original DTD) instead of from A to B . For example, if $\sigma(A, B) = A/C[D]/B$, user query contains a fragment $B/parent::A$, then this fragment will be rewritten as $B/parent::C[D]/parent::A[QUERY\ REWRITE(root, path(root, A))]$. Note that $parent::A$ is extended with a qualifier that guarantees that A is indeed accessible.

If a set of steps of $\langle path \rangle$ can be represented as an array then we can divide that array in two equal subarrays (one contains the left subpath, another - the right one), we can recursively rewrite subpath corresponding to those subarrays and join them in $O(n)$ time, where n is a number of nodes reachable by the left subpath from **root** node. If the path contains a reverse **parent** axes, the inversion of a corresponding σ -function will take $O(k)$ operations, where k is the number of steps in σ -function. Hence, the overall complexity will be no more than $O((n \times k \times \log m))$, where m is a number of steps in path including those in


```

1 : ann(site, regions)           = N
2 : ann(regions, namerica)      = Q[/site/people/person[@id = $login]/address/country/text() = 'UnitedStates'
   or /site/people/person[@id = $login]/address/country/text() = 'Canada']
3 : ann(namerica, item)        = Q[location/text() = 'United States' and
   shipping/text() != 'Will ship only within country']
4 : ann(open_auction, seller)   = Q[@person = $login]
5 : ann(open_auction, bidder)   = Q[personref/@person = $login]
6 : ann(bidder, increase)       = Y
7 : ann(closed_auction, price)  = Q[parent :: closed_auction/(buyer/personref or seller)/@person = $login]
8 : ann(closed_auction, buyer)  = Q[personref/@person = $login]
9 : ann(people, person)         = Q[@id = $login]
10 : ann(person, profile)       = Q[business/text() = 'Yes']
11 : ann(profile, business)     = N

```

Fig. 4. Annotation for a registered user of an auctions portal

```

q1 : regions/africa
q2 : namerica/item
q3 : people/person/profile/business
q4 : open_auctions/open_auction[initial/text() < '50' and current/text() > '100']/increase

q5 : people/person/*
q6 : open_auctions/open_auction/*/*

q7 : people/person/profile[@income > 85000]/parent :: person/name
q8 : people/person/name/parent :: person/address/parent :: person/profile[@income > 85000]/parent :: person/name

```

Fig. 5. Set of queries

More precisely, the first rule (1) prohibits an access to the information about items and their location in regions. Then, this rule is overridden so that items located in North America are visible for people from United States or Canada (policy 2) and items from United States that can be shipped not only within the country are visible for everyone (policy 3).

Next, information about seller/bidder of any open auction (4 and 5) is visible if the viewer is seller/bidder himself. In the same way information on buyer is protected in closed auctions (8). All increases of any particular auction is publicly visible (7).

Rule 9 says that **person** may get a personal information only about himself/herself, i.e. person id should be equal to user login. The latter is expressed by a dynamic variable **\$login**. The instantiation of this variable is hardcoded, i.e. if qualifier contains substring **\$login**, it should be replaced by the login name passed as a program parameter or taken as a system variable. A personal profile, instead, can be publicly available if field **business** has a text value *Yes*. However, **business** is not visible in its turn.

Queries. We have defined a set of queries (see Fig. 5) to test both the algorithm QUERY REWRITE and the advantages of query evaluation w.r.t. different use cases (see Fig. 1).

Query q_1 and q_2 will try to get an access to items in African and North American regions respectively. Next, q_3 should always return empty set because

```

q2 : regions/namerica[/site/people/person[@id = $login]/address/country/text() = ' UnitedStates' or
      /site/people/person[@id = $login]/address/country/text() = ' Canada']/item[location/text() = ' United States'
      and shipping/text() != ' Will ship only within country']
q4 : open_auctions/open_auction[initial/text() < ' 50' and current/text() > ' 100']
      /bidder[not(personref/@person = $login)]/increase
q5 : people/person[@id = $login]/(address|watches|phone|name|creditcard
      |emailaddress|profile[business/text() = ' Yes']/homepage)
q7 : people/person[@id = $login]/profile[business/text() = ' Yes'][@income > 85000]/
      self :: profile[business/text() = ' Yes']/parent :: person[/site/people/person[@id = $login]]/name

```

Fig. 6. Set of rewritten queries

of policy rule 11. “Long” query q_4 looks for increases of auctions with the particular start and current prices. Queries q_5 and q_6 introduce asterisk *. Reverse **parent** axis is introduced in q_7 that tries to get a name of a person whose income is more than 85K. The result set of this query evaluation should be the same as of the query q_8 that includes three reverse **parent** axes.

Some rewritten queries are depicted on Fig. 6. More precisely, in q_2 before **namerica**, a missing **region** goes. Field **namerica** itself is extended with a qualifier which is $\sigma(\text{regions}, \text{namerica})$. In the same way, after **item**, $\sigma(\text{namerica}, \text{item})$ goes. The rewriting of q_4 shows that the user will receive **increases** that were posed by other users, while his own increase values are located under field **bidder** that is visible for him since its person reference is equal to his login. In q_5 , asterisk * is rewritten into the union of visible nodes that are in the relation **child** with **people**. The rewriting of q_6 has the same idea. In q_7 , we would like to note that **profile** is extended with a security qualifier $\sigma(\text{person}, \text{profile})$ in addition to a user-defined filter on it. The second line of q_7 represents reverse axis rewriting. This part is inserted in q_8 three times. Finally, q_1 and q_3 are rewritten to **null** because the user cannot see either **africa** (policy rule 1) or **business** (policy rule 11).

XML documents. XMark data generator [1] produces XML documents conforming to a DTD **auctions.dtd** depicted on Fig. 3. Number and type of elements in resulting XML depend on parameter called *factor*. The significant feature of XMark benchmark is the generation of one unique XML document for one factor value. We generated 20 XML documents with factor $0.0i0$ and $0.0i5$, where $i = \overline{0, 9}$. The size of these XML files varies from 0.3Mb to 9Mb (all together around 100Mb).

Performance results. We tested our implementation on Windows XP platform, Intel Pentium M 1.4GHz, 256Mb DDR SDRAM. For each XML document (recall, we have 20 XML documents), we run evaluation of each query $q_j, j = \overline{1, 8}$ from the viewpoint of 10 users ($\text{login} = \text{person}_i$, where $i = \overline{0, 9}$) measuring time of query rewriting (that includes time of DTD view construction) $t_{i,j}^1$ and time of query answering $t_{i,j}^2$. Finally, an arithmetic means of t_j^2 were calculated and summed for every $j = \overline{1, 10}$.

Query rewriting performance results are shown in Table 1 where each cell contains time (in milliseconds) required to rewrite $q_i, i = \overline{1, 8}$. In all experiments

Table 1. Query rewriting performance results

	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8
<i>person</i> ₀	70	71	60	71	60	60	60	70
<i>person</i> ₁	60	60	60	60	60	60	70	71
<i>person</i> ₂	60	70	60	60	60	70	70	70
<i>person</i> ₃	60	60	60	70	60	70	80	70
<i>person</i> ₄	60	60	70	70	60	60	70	80
<i>person</i> ₅	60	60	70	60	70	70	70	80
<i>person</i> ₆	60	60	70	60	70	70	80	61
<i>person</i> ₇	70	60	60	80	80	70	80	70
<i>person</i> ₈	60	60	70	70	70	70	60	70
<i>person</i> ₉	70	81	60	70	60	60	61	80
avg	63	64.2	64	61.1	65	66	69	72.2

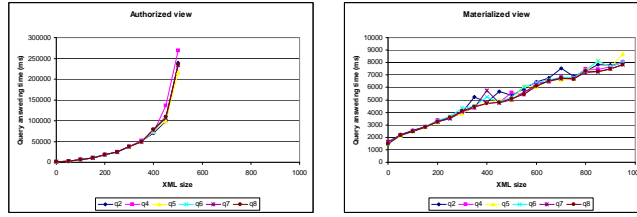
DTD view construction time was between 520 and 620 milliseconds. In the query rewriting part, the measured time includes a query parsing and QUERY REWRITE running. We used SiXPath ¹ processor to parse XPath queries into their tree representation. The processor represents steps of a path in array, so we can apply divide-and-conquer technique that improves efficiency of query rewriting. From Table 3, it can be easily seen that more complicated query requires more time to be rewritten. However, one point is that in reality, the user does not formulate such cumbersome queries like q_8 that, indeed, requires “much” time for rewriting. On the other hand, even if the user issues a query that contains time-consuming elements like * or parent axes, the query rewriting performance degrades insignificantly (less than 10 milliseconds between the easiest and the hardest query in our test case).

Query answering time was evaluated by a Xalan ² XPath evaluator. Unfortunately, XML Task Force evaluator ³ which is shown in [17] as the most efficient and scalable XPath evaluator, could not be used in our test framework because it does not accept queries that contain union, e.g., like queries q'_5 , q'_6 . The overall result computation performance is shown on Fig. 7 for each query q_i , $i = \overline{1, 8}$ and each of 20 XML files. Namely, the Fig. 7(a) shows our experience with XML Access Control Processor (XMLACP) developed by Damiani et al. [13]. Query evaluation includes a preliminary construction of an authorized view which is extremely time consuming. This is because security annotations are propagated on the XML tree. Moreover, security annotation is, basically, a pair $\langle xp, lbl \rangle$, where xp is an XPath expression defining a node being labeled, lbl is Y or N. Since xp may include qualifiers, the step of initial labeling takes much time which grows exponentially with the growth of the initial XML document size.

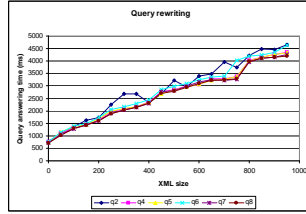
¹ <http://sourceforge.net/projects/sixpath>

² <http://xml.apache.org/xalan-j/>

³ <http://www.dbai.tuwien.ac.at/research/xmltaskforce/>



(a) Authorization view use case (b) Materialized view use case



(c) Query rewriting use case

Fig. 7. Experiments

We have to note that XMLACP approach does not delete forbidden XML nodes if they have permitted descendants, it rather clears them from attributes. Therefore user queries should include even forbidden steps that may reveal sensitive information. Running a few steps forward, we constructed such queries from the output of QUERY REWRITE but without qualifiers of the initial security annotation of DTD. Another issue is related to a set of authorized views. Their number can be enormous (e.g., we can derive 126 personal views from an XML document computed with factor 0.050, i.e. of the size around 500Kb; this number grows exponentially with increasing XML size on 500Kb) but their sizes is about 10% less than the initial XML document. Therefore, storing all individual views is not feasible even if do not consider integrity maintenance problem.

Materialization use case shown in Fig. 7(b) shows a much better result in time performance. On the other hand, space required for view storing is also reduced since we delete all intermediate N-labeled nodes.

Finally, query rewriting use case (Fig. 7(c)) is even better: it is at least two times faster than the previous case and does not require an additional storage for a materialized view.

Unfortunately, we could not retrieve other existing XML securing systems, e.g., Author- \mathcal{X} [5] (it is lost and cannot be recuperated because of the storing computer crash ⁴), Lock-X [9] (for the same reason ⁵), SMOQE [?] (because of experiment conduction for additional research ⁶).

⁴ Personal communication with Elena Ferrari

⁵ Personal communication with SungRan Cho

⁶ Personal communication with Wenfei Fan

5 Related Work

5.1 Runtime policy evaluation

The general scenario of the current category of proposals is the following: The systems defines a set of access control rules of the form $\langle subject, object, action, sign \rangle$ where *subject* is self-explaining, *object* is an XML element/attribute expressed by XPath, *action* is typically **read/write**, $sign \in \{+, -\}$. Different conflict resolution rules and default policy are established as well. With respect to user's request $\langle req_subject, req_object, req_action \rangle$, access control rules applicable to *req_subject* is selected, their *signs* for *req_action* are propagated to *req_object*. Hence, permission is granted to the user if the resulting sign on *req_object* is +; otherwise, the access is denied.

In particular, [20] introduces the notion of *provisional authorization*, when some provisional action (e.g., logging, transcoding) is performed according to the user's request. The proposal of [7] considers the case when the access control is moved to clients, e.g, secure tokens and smart cards that are used as trust components in different mobile devices (e.g., PC, PDA, cellular phone) participating in applications dealing with sensitive information (e.g., certification, electronic voting, e-payment, health care, etc.). Several papers consider the case of evolving access control policies expressed in XQuery [16] and by means of RDF [2], [18]. Such policies can be used for a derivation of new access control rules including content-based constraints of requested and other documents, environmental information like time and place of request initiator, information about possessed privileges.

Run-time policy evaluation can be accelerated by efficient lookup of compressed accessibility map where compression is performed on objects [36], on objects and actions [19], on objects, subjects and actions [37]. Another direction for improving runtime policy evaluation concerns statical analysis of user queries [27], integration of policy into user query [24], matching user query against efficient policy representation as a tree [28]. In the case when mandatory access control is considered, recursive checks can be reduced by adding special predicates to node tests in the user query [10].

An association between XML nodes can be hidden either at the stage of policy definition [18] or after detecting the possibility of information leakage in security view [35]. Finally, access control for XML documents can be strengthened with a role-based concepts [33].

5.2 Security views for XML

This scenario is called *authorization view use case* in the current paper. The details can be seen, e.g., in [5], [15], [13]. The variation of this use case enforces access control policies cryptographically.

For example, the approach in [6] is based on Author- \mathcal{X} [5]. It avoids generation of multiple physical views for each use by means of different keys for encrypting different portions of the same document. One and only one key is

responsible for encryption of each portion of the source XML document. To minimize the number of encryption keys, the portions of the document protected with the same set of policies are encrypted with the same keys. The consequent scenario is key distribution and periodical broadcast of the encrypted document.

Miklau and Suciu extend the Bertino's idea of secure and selective dissemination of XML documents with the notion of *conditional access control rules* [25], which generalizes the term "subject", i.e. authorization is based not on network identifier or user name, but on knowledge presented by the user.

The ideas of [6] and [25] are refined and extended with RBAC in [12].

5.3 Schema-based security views

Stoica and Farkas [32] proposed to produce single-level views of XML when conforming DTD is annotated by labels of different confidentiality levels. 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 level no higher than 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. No query rewriting is discussed.

Another view-based approach is proposed by Fan et al in [14]. In [21], we refined this solution with other DTD view derivation and XML view materialization algorithms. In this paper, we underlined the similarities and the differences with our approach to query rewriting.

The recent approach to schema-based security views was presented in [26]. The solution allows a complete restructuring of a DTD and relies on a command-like specification language. However, it was mentioned in [26] that many operations are not commutative and have restrictions that means a possibility of errors while designing access control policies. A sophisticated query rewriting is provided.

6 Conclusion

In this paper, we have studied the performance of answering queries on an XML database, subject to access control annotations applied on the original DTD. We show that the query rewriting approach compared to the case of authorized/materialized view is more efficient in sense of time and space.

Time effectiveness takes place because we avoid view materialization which is a time consuming operation. In our experimental benchmark the query rewriting strategy retrieves answer for user query at least 2 times faster than in the case the materialized view and at least 100 faster than in the case of authorized view. Another mentioned point is the space preserving property of advanced method. We recall that the number of views can be extremely large and their size may, in the very good case, be 50% smaller w.r.t. the initial XML document. This may

cause problems with the disc space allocation and with the maintenance of data integrity.

Our future research direction is related to drawbacks of the suggested labeling mechanisms outlined below:

1. Hardcoding of user attributes in qualifiers. If at some stage behavior of the system is changed, e.g. from identity based authorization to credential based one (i.e. login may be substituted by private key or social security number, or even by a boolean combination of these attributes), we need to rewrite and recompile the source code responsible for parsing a set of program parameters.
2. The suggested approach of security specification lacks flexibility in defining access rules based on user credentials (i.e. distributed environment with multiple users unknown in advance) and purposes for data storage/access (i.e. privacy issues).
3. Multiple labels of the same semantics but of the different syntax complicate variation of policies.
4. Multiple labels of the same syntax. This problem is subsumed by the previous remark because the same syntactically equal qualifiers have the same semantics. Therefore, a change in one qualifier requires revisiting of all the same qualifiers.

We are going to investigate the issue of ID/IDREF attributes. Namely, we want to answer the question: what happens if ID attribute should be deleted but its referencing IDREF attribute should exist? Answering this question will help us to extend XML security views to a distributed/federated/shared environment. Finally, we are planning to extend our view derivation and query rewriting algorithms for the case of recursive DTDs.

References

1. XMark – An XML Benchmark Project. <http://monetdb.cwi.nl/xml/index.html>.
2. Chutiporn Anutariya, Somchai Chatvichienchai, Mizuho Iwaihara, Vilas Wongse, and Yahiko Kambayashi. A rule-based XML access control model. In *RuleML*, pages 35–48, 2003.
3. M. Benedikt, C. Chan, W. Fan, R. Rastogi, S. Zheng, and A. Zhou. DTD-directed publishing with attribute translation grammars. In *Proceedings of the 28th Conference on Very Large Data Bases (VLDB'02)*, 2002.
4. E. Bertino, S. Jajodia, and P. Samarati. A flexible authorization mechanism for relational data management systems. *ACM Transactions on Information Systems (TOIS)*, 17(2):101–140, 1999.
5. Elisa Bertino, M. Braun, Silvana Castano, Elena Ferrari, and Marco Mesiti. Author-X: A Java-based system for XML data protection. In *Proceedings of the IFIP TC11/ WG11.3 Fourteenth Annual Working Conference on Database Security*, pages 15–26. Kluwer, B.V., 2001.
6. Elisa Bertino and Elena Ferrari. Secure and selective dissemination of XML documents. *ACM Transactions on Information and System Security (TISSEC)*, 5(3):290–331, 2002.

7. Luc Bouganim, François Dang Ngoc, and Philippe Pucheral. Client-based access control management for xml documents. In *Proceedings of the 30th Conference on Very Large Data Bases (VLDB'04)*, pages 84–95, 2004.
8. N. Boulahia-Cuppens, F. Cuppens, A. Gabillon, and K. Yazdanian. Multiview model for object-oriented database. In *Proceedings of the Annual Computer Security Applications Conference*, pages 222–231, 1993.
9. SungRan Cho, Sihem Amer-Yahia, Laks V. S. Lakshmanan, and Divesh Srivastava. LockX: a system for efficiently querying secure XML. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data (SIGMOD'03)*, pages 669–669, San Diego, California, 2003. ACM Press.
10. SungRan Cho, Sihem Amer-Yahia, Laks V.S. Lakshmanan, and Divesh Srivastava. Optimizing the secure evaluation of twig queries. In *Proceedings of the 28th Conference on Very Large Data Bases (VLDB'02)*, pages 490–501, 2002.
11. J. Clark and S. DeRose. XML path language (XPath) version 1.0. w3c recommendation, november 1999. <http://www.w3.org/TR/xpath>.
12. Jason Crampton. Applying hierarchical and role-based access control to XML documents. In *Proceedings of ACM Workshop on Secure Web Services (SWS'04)*, Fairfax, VA, USA, 2004. ACM Press.
13. 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 (TISSEC)*, 5(2):169–202, 2002.
14. Wenfei Fan, Chee-Yong Chan, and Minos Garofalakis. Secure XML querying with security views. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD'04)*, pages 587–598. ACM Press, 2004.
15. Alban Gabillon and Emmanuel Bruno. Regulating access to XML documents. In *Proceedings of the IFIP TC11/WG11.3 fifteenth annual working conference on Database and application security*, pages 299–314, Niagara, Ontario, Canada, 2001. Kluwer Academic Publishers.
16. Siddhartha K. Goel, Chris Clifton, and Arnon Rosenthal. Derived access control specification for XML. In *Proceedings of the 2nd ACM Workshop On XML Security (XMLSEC'03)*, pages 1–14. ACM Press, 2003.
17. G. Gottlob, C. Koch, and R. Pichler. Efficient algorithm for processing XPath queries. In *Proceedings of the 28th Conference on Very Large Data Bases (VLDB'02)*, 2002.
18. Vaibhav Gowadia and Csilla Farkas. RDF metadata for XML access control. In *Proceedings of the 2nd ACM Workshop On XML Security (XMLSEC'03)*, pages 39–48, Fairfax, Virginia, 2003. ACM Press.
19. Mingfei Jiang and Ada Wai-Chee Fu. Integration and efficient lookup of compressed XML accessibility maps. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 17(7):939–953, July 2005.
20. Michiharu Kudo and Satoshi Hada. XML document security based on provisional authorization. In *Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS'00)*, pages 87–96, New York, NY, USA, 2000. ACM Press.
21. Gabriel Kuper, Fabio Massacci, and Nataliya Rassadko. Generalized XML security views. In *Proceedings of the tenth ACM symposium on Access control models and technologies (SACMAT'05)*, pages 77–84. ACM Press, June 2005.
22. T. F. Lunt, R. R. Schell, W. R. Shockley, M. Heckman, and D. Warren. A near-term design for the SeaView multilevel database system. In *Proceedings of IEEE Symposium on on Security and Privacy (SSP-88)*, pages 234–244. IEEE Computer Society Press, 1988.

23. Teresa F. Lunt, Dorothy E. Denning, Roger R. Schell, Heckman, Mark, and William R. Shockley. The SeaView security model. *IEEE Transactions on Software Engineering (TOSE)*, 16(6):593–607, 1990.
24. Bo Luo, Dongwon Lee, Wang-Chien Lee, and Peng Liu. QFilter: Fine-grained runtime XML access control via NFA-based query rewriting. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management (CIKM'04)*, pages 543–552, New York, NY, USA, 2004. ACM Press.
25. Gerome Miklau and Dan Suciu. Controlling access to published data using cryptography. In *Proceedings of the 29th Conference on Very Large Data Bases (VLDB'03)*, pages 898–909, September 2003.
26. Sriram Mohan, Arijit Sengupta, Yuqing Wu, and Jonathan Klinginsmith. Access control for XML - a dynamic query rewriting approach. In *Proceedings of the 32th Conference on Very Large Data Bases (VLDB'06)*, pages 1–12, Seoul, Korea, 2006. VLDB Endowment.
27. Makoto Murata, Akihiko Tozawa, Michiharu Kudo, and Satoshi Hada. XML access control using static analysis. In *Proceedings of the 10th ACM Conference on Computer and Communication Security (CCS'03)*, pages 73–84. ACM Press, 2003.
28. Naizhen Qi and Michiharu Kudo. XML access control with policy matching tree. In *Proceedings of the 10th European Symposium on Research in Computer Security (ESORICS'05)*, volume 3679 of *Lecture Notes in Computer Science*, pages 3–23. Springer-Verlag, 2005.
29. Xiaolei Qian. View-based access control with high assurance. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy (SP'96)*, page 85, Washington, DC, USA, 1996. IEEE Computer Society.
30. Nataliya Rassadko. Policy classes and query rewriting algorithm for XML security views. In E. Damiani and P. Liu, editors, *Proceedings of the IFIP TC11/WG11.3 20th annual working conference on Database and application security (IFIP'06)*, volume 4127 of *Lecture Notes in Computer Science*, pages 104–118. Springer-Verlag, July 2006.
31. P. D. Stachour and B. Thuraisingham. Design of LDV: A multilevel secure relational database management system. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2(2):190–209, 1990.
32. Andrei G. Stoica and Csilla Farkas. Secure XML views. In *Proceedings of the 16th International Conference on Data and Applications Security (IFIP'02)*, volume 256 of *IFIP Conference Proceedings*, pages 133–146. Kluwer, July 2002.
33. Jingzhu Wang and Sylvia 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 (SACMAT'04)*, pages 70–77. ACM Press, 2004.
34. J. Wilson. Views as the security objects in a multilevel secure relational database management system. In *Proceedings of IEEE Symposium on Security and Privacy (SSP'88)*, pages 70–84. IEEE Computer Society Press, 1988.
35. Xiaochun Yang and Chen Li. Secure XML publishing without information leakage in the presence of data inference. In *Proceedings of the 30th Conference on Very Large Data Bases (VLDB'04)*, pages 96–107, 2004.
36. Ting Yu, Divesh Srivastava, Laks V. S. Lakshmanan, and H. V. Jagadish. A compressed accessibility map for XML. *ACM Transactions on Database Systems (TODS)*, 29(2):363–402, 2004.
37. Huaxin Zhang, Ning Zhang, Kenneth Salem, and Donghui Zhuo. Compact access control labeling for efficient secure XML query evaluation. In *Proceedings of the 21st International Conference on Data Engineering Workshops (ICDEW'05)*, page 1275, 2005.