# Supporting XML Security Models using Relational Databases: A Vision

Dongwon Lee, Wang-Chien Lee, and Peng Liu

Penn State University

`dongwon@psu.edu, wlee@cse.psu.edu, pliu@ist.psu.edu`

**Abstract.** As the *secure* distribution and sharing of information over the World Wide Web becomes increasingly important, the needs for flexible and efficient support of access control systems naturally arise. Since the eXtensible Markup Language (XML) is emerging as the format of the Internet era for storing and exchanging information, there have been, recently, many proposals to extend the XML model to incorporate security aspects. To the lesser or greater extent, however, such proposals neglect the fact that the data for XML documents will most likely reside in relational databases, and consequently do not utilize various security models proposed for and implemented in relational databases.
In this paper, we take a rather different approach. We explore how to support security models for XML documents by leveraging on techniques developed for relational databases. More specifically, in our approach, (1) Users make XML queries against the given XML view/schema, (2) Access controls for XML data are also specified in the XML model, but (3) Data are stored in relational databases, and (4) Security check and query evaluation are also done in relational databases. Instead of re-inventing wheels, we take two representative methods in both XML security model and XML to relational conversion problems, and show how to glue them together in a seamless manner to efficiently support access controls for the XML model using relational databases.

## 1 Introduction

Since the eXtensible Markup Language (XML) was invented for exchanging and storing information over the World Wide Web (Web) [4], its usage has exploded significantly. As more information is exchanged and processed over the Web, the issues of security become increasingly important. Such issues are diverse, spanning from data level security using cryptography to network transport level security to high-level access controls. In this paper, our focus is on how to support high-level *access controls* for XML documents.

Table 1 illustrates the current development of XML security models. First row refers to (research-oriented) security models developed for XML and relational models, respectively, while second row refers to database products for each model. In general, not all features proposed by research in the first row are implemented in the real implementations in the second row yet. For instance,

**Table 1.** The overview of XML and Relational security model supports.

| **XML** | **Relational** | |
|---|---|---|
| XML Security Models ([6], [2], etc) | Relational Security Models ([13], etc) | **Models** |
| XML Databases (Xindice, Tamino, etc) | Relational Databases (Oracle, DB2, SQL Server, etc) | **Products** |

most XML database products currently do not have any support for access controls. Similarly, commercial relational products have implemented only minimal features of access controls via authorizations such as `GRANT` and `REVOKE`.

Recently, many access control methods extending the XML model to incorporate security aspects have been proposed (e.g., XACML [11], [6], [2], [23]). To the lesser or greater extent, however, such proposals neglect the fact that the most data for XML documents still reside in relational databases behind the scenes, and consequently do not utilize various security models that have been proposed for and implemented in relational databases. We believe that current XML security research (i.e., upper-left column of Table 1) is re-inventing wheels without utilizing existing relational security models (i.e., upper-right column of Table 1) or security features that are already implemented and being used in relational products (i.e., lower-right column of Table 1).

Therefore, our goal in this research is to study how to support XML security models by utilizing existing security support of relational security models or relational products. More specifically, we assume that

– XML documents are converted into and stored in relational databases.
– Users are given an XML view/schema against which they issue XML queries.
– Access controls are specified by security administrators in the XML schema and documents.
– Security check and query evaluation are done by relational databases (or by a middleware on top of relational databases), and only valid answers are returned to users in the XML format.

In this paper, as a preliminary work, we explore various research issues and a few possible sketches of solutions to achieve the vision of supporting XML security models using relational databases. Furthermore, we present an illustrative example that shows a complete steps of the vision as a proof of concept. We hope to draw more research interests and efforts onto the direction that we are proposing in this paper.

## 2 Related Work

Since our research relates to two seemingly unrelated works, we first survey those works in two separate categories, and then discuss a few works that are overall similar/dissimilar to our proposal.

## 2.1 XML and Relational Security Models

**XML access control models.** Several authorization-based XML access control models are proposed. In [19], authorizations are specified on portions of a HTML document, however, no semantic context similar to that provided by XML can be supported. In [7], a specific authorization sheet is associated with each XML document/DTD expressing the authorizations on the document. In [6], the model proposed in [7] is extended by enriching the authorization types supported by the model, providing a complete description of the specification and enforcement mechanism. Among comparable proposals, in [2], an access control environment for XML documents and some techniques to deal with authorization priorities and conflict resolution issues are proposed. Finally, the use of authorization priorities with propagation and overriding, which is an important aspect of XML access control may recall approaches in the context of object-oriented databases, like [9] and [18]. Although our proposal is based on existing XML authorization models such as [6], we focus on how to use relational databases to help enforce XML authorization models, and none of the above XML authorization models address the interaction between XML and relational access controls.

**Relational access control models.** Relational access control models can be classified into two categories: *multilevel security models* [15, 24, 20] and *discretionary security models.* Multilevel security models assign each data object (e.g., a tuple) as well as each subject (e.g., a user) a security *level* (or class), and enforce the following two specific access control rules: (1) a level $L_i$ subject can never read a level $L_j$ data object unless $L_i \geq L_j$; (2) a level $L_i$ subject can never write a level $L_j$ data object unless $L_j \geq L_i$. Although multilevel security models are widely used in military applications, they are seldom used in commercial applications for their restrictive nature. By contrast, discretionary security models allow the creator of a data object $x$ to own all the privileges associated with $x$ and to grant some of the privileges to other users in such a way that a variety of access control policies could be enforced. Discretionary security models are dominant in commercial data management. Although several more expressive and flexible discretionary security models are proposed [13, 14], most real world database systems implement a discretionary access control model similar to the one implemented in System R [12], where (1) access control is supported by the GRANT and REVOKE commands; (2) only table or column level authorizations are directly supported; (3) views are used to indirectly support some data dependent access control. Nevertheless, role-based access control [21] is not implemented in System R but implemented by most existing DBMSs such as Oracle. It is clear that our XML-relational access control scheme cannot be directly supported by table or column level authorizations, since each XML path is usually stored in a set of tuples. Although views can be used to support XPath-oriented access control, they are expensive and difficult to manage.

## 2.2 Conversion Methods between XML and Relational Models

Toward conversion between XML and relational models, an array of research has addressed the particular issues lately. On the commercial side, database vendors are busily extending their databases to adopt XML types. Typically, they can handle XML data using BLOB/CLOB formats along with a limited keyword searching or using some object-relational features [5, 1], but not many details have been revealed. On the research side, various proposals have been made recently. Here, we only survey two kinds of works related to XML-to-relational conversion – *structure-based* and *data-based* conversions. The former generates a target relational schema from the given XML schema as a source, while the latter uses XML documents directly to generate relational tuples. Since the data-based conversion methods do not require an XML schema as input, the methods work for arbitrary XML schema. However, it cannot capture semantics that appear in XML schema, but hidden in XML documents.

**Structure-based conversions.** Work done in STORED [8] is one of the first significant and concrete attempts to this end and deals with non-valid XML documents. STORED uses a data mining technique to find a representative DTD whose support exceeds the pre-defined threshold and convert XML documents to relational format using the DTD. [3] discusses template language-based conversion from DTD to relational schema which requires human experts to write an XML-based conversion rule. [22] presents three inlining algorithms that focus on the table level of the schema conversions. On the contrary, [16] proposes a method where the hidden semantic constraints in DTD are systematically found and translated into relational formats. Since the method is orthogonal to the structure-oriented conversion methods, it can be used along with algorithms [8, 3, 22, 10] with little change. [17] proposes an algorithm for mapping a DTD to the Entity-Relationship (ER) model (and thus the relational model) and examine some of the issues in loading XML data into the generated model.

**Data-based conversions.** [10] studies different performance issues among eight algorithms that focus on the attribute and value level of the schema. [25] proposes a DTD-independent, path-based mapping algorithm. While ignoring specific characteristics hidden in each DTD, [25] decomposes XML documents into element, attribute, text and path tables, so that the changes of DTDs of the XML documents do not necessarily result in invalid mapping as found in examples [8, 22].

## 2.3 Supporting XML Security Models using Relational Databases

To our best knowledge, the only work that is directly related to our proposal is [23]. [23] also proposes an idea of using RDBMS to handle access controls for XML documents, in a rather limited setting. According to our taxonomy of Table 8, [23] roughly uses the following methods: model-to-RDBMS conversion, schema-level XML security model, structure-based data conversion, and external

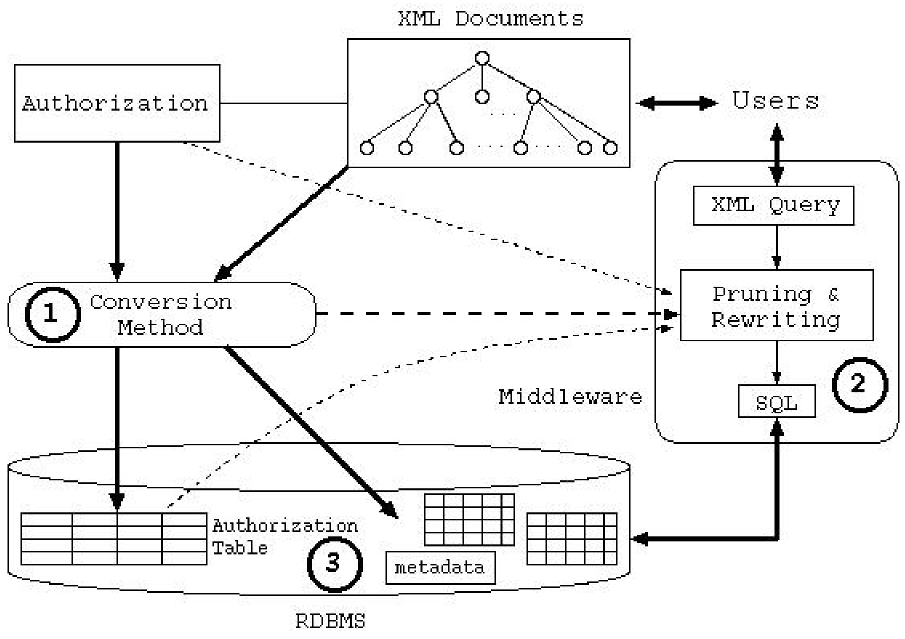

**Fig. 1.** The framework for supporting XML security models using relational databases.

and pre-pruning security check. In this research, however, we aim at conducting a much more extensive and systematic research than [23].

## 3 Framework and Research Issues

In this section, we discuss various research issues that arise in supporting XML security models using relational databases. Figure 1 illustrates an overall framework for our vision. From the users' perspective, they are accessing data in a given set of XML documents defined by associated DTDs or XML schema. The access controls of the data are governed by authorization rules specified in accordance with a given XML security model.

Behind the scene, the XML documents and the associated authorization rules are actually preprocessed and converted into relational tables, metadata, and authorization tables in relational databases. This process is illustrated by circle (1) in the figure. To access the XML data, a user submits an XML query against the known XML schema. Based on the authorization rules, a user's query is first pruned and rewritten into SQL. This process is illustrated by circle (2) in the figure. Please note that, in our proposal, the pruning process here can be adapted to generate various execution plans. Thus, access control can be enforced purely based on security pruning, internal security mechanism of relational databases, or a mixture of these two. Thus, when the SQL query is processed in the relational databases, authorization table may be checked (as illustrated in (3)) before the valid answers are returned to users in the XML format.

### 3.1 XML to Relational Conversion

As shown in Figure 1, two of the initial but critical tasks for secure storage and access of XML data in relational databases are to (1) map XML authorization rules into the existing access control mechanism in relational databases; and (2) map XML documents into tables in relational databases. In the following, we discuss some of the issues.

1. **Theoretical study of XML and Relational security models.** To fully realize our vision, a thorough study on the expressive power of XML and relational security models must be done. Since there is not a unified security model for both models, nor a single standard agreed upon in community, one must first understand the pros and cons of different security models (e.g., security propagation, conflict resolution, etc) and their relationships among others. Furthermore, a finding of theoretical mapping (i.e., complete and sound algorithms) from the source XML security model to the target relational security model would be challenging tasks.

2. **Schema-level vs. Instance-level.** When XML security authorizations are specified based on XML schema, it is called a *schema-level* access control. When an XML document element or attribute can carry additional tag, specifying security information that can overwrite access control specified in the XML schema, it is called as *instance-level* access control. Depending on which scheme is used in the XML security model, how to convert such authorization rules into relational format becomes challenging. This issue also affects security evaluation strategy significantly.

3. **Which XML-to-relational conversion method is appropriate to use?** Recently many conversion algorithms have been proposed (e.g., [8, 22, 10, 25]), each of which has different pros and cons for different applications. When the schema-level access control is specified, for instance, we believe the path-based conversion methods such as XRel [25] are good candidates. This is because in XML model, using XPath to specify the scope of the objects is quite natural. For instance, a statement "*manager* has `read` and `write` accesses for `//employees/salary`" indicates that a subject (i.e., manager) can read and write all objects (i.e., elements) as long as the objects are `salary` under `employee` element. When path-based XML-to-relational conversion methods are used, such XPath-based security scope can be easily captured into single authorization table in relational databases. If other structure-oriented conversion methods such as hybrid inlining [22] are used for conversion, then authorization information in the XML model would be scattered into several table in relational databases, making difficult or inefficient to do security evaluation. However, in general, a question of which XML-to-relational conversion algorithms suits best for the given application is non-trivial to answer.

## 3.2 Security Evaluation

In this context, we explore three dimensions – *where*, *how*, and *when* to evaluate and enforce security information.

1. **Where to evaluate?** Incoming queries are in XML format such as XPath, while query processing is done by relational databases. Therefore, at some point, input queries must be rewritten to SQL format. One extreme to support security evaluation is to do all the necessary security check outside of relational databases, while the other extreme is to push all security check down to database engine, utilizing built-in features in relational databases. Let us consider three strategies for instance. (1) In the *external evaluation*, users' XML query is compared against authorization rules from both XML and relational models and pruned such that only query nodes that are valid against the given authorization rules remain at the end. Since pruning stage guarantees only valid data access, relational database can do normal SQL query processing, without worrying about insecure access. (2) In the *internal evaluation*, authorization rules are first converted and stored in relational databases via some conversion method, and security check is conducted inside relational databases. This approach explores features such as GRANT, REVOKE or view implemented in relational databases. (3) For some cases, it might be better to combine two extremes (i.e., external and internal security evaluation approaches) to strike a balance, thus *hybrid evaluation*. It is not clear how to split such a task – what part of security check is best done externally, and what is best done inside of relational databases?
2. **How to evaluate?** Second dimension of security evaluation is *how* to enforce the security. In the *strict evaluation* approach, no data protected by authorization rules can be accessed or returned. However, in the *relaxed evaluation* approach, during query processing, any data can be accessed, but only secure data must be returned to users. Interesting question is, then, when would be such a relaxed evaluation useful? Some XML data might be tagged as "accessible during query processing", but not "returnable". In such a situation, a question of if one can utilize such a relaxed evaluation for faster and secure query processing is interesting.
3. **When to evaluate?** Last dimension of security evaluation is *when* to evaluate. In the most primitive *post-processing* approach, XML query is processed like a normal query, then at the end, a portion of answers that violate security check is pruned and the remains are returned to user. However, one can think of other approaches such as *pre-pruning* (e.g., [6]) or *interleaving*. Investigating when one approach is better than another and, if so, in what situation would be a challenging task.

## 3.3 Authorization in Relational Databases

In representing authorizations in relational databases, the typical method is to use the *authorization table* that essentially contains a tuples of subject/object pairs and its allowed actions. Although varied, most authorization table schemes restrict the granularity of access control for relational model to either table-level or column-level. On the contrary, in XML security model, the finer access control is possible (e.g., nodel-level). Therefore, due to this differences of granularities of two models, problem occurs. Consider the following DTD, for instance:

```
<!ELEMENT A (B+, C, X)>
<!ELEMENT B (D, E)>
<!ELEMENT C (#PCDATA)>
<!ELEMENT D (#PCDATA)>
<!ELEMENT E (#PCDATA)>
<!ELEMENT X (#PCDATA)>
```

If one used the hybrid inlining method [22] for XML-to-relational conversion, one essentially would have the following two tables generated, `A(C,X)`, and `B(D,E,fk_A)`, where `fk_A` is the foreign key referencing the primary key of the table `A`. Now, consider the following four authorization rules:

```
A1:  (Admin, /A/B, read, +)
A2:  (Admin, /A/C, read, +)
A3:  (Admin, /A/B[./D>5], read, +)
A4:  (Admin, /A/B[./D>5]/E, read, +)
```

For instance, the first rule `A1` states that the subject (i.e., Admin) can read all nodes `B` under `A`, and the fourth rule `A4` states that the subject can read all nodes `E` under `B` under `A` as long as `B` has a child node `D` whose value is greater than 5. Figure 2 illustrates pictorial representation of the scope of objects covered by each authorization rule. For instance, the rule `A1` covers all the nodes `B` of XML document. According to the hybrid inlining conversion, all the information related to the nodes `B` would be stored into the table `B(D,E,fk_A)`. Therefore, by applying the *table-level* SQL GRANT statement shown below, one can achieve the same security enforcement as dictated in the `A1`.

```
GRANT SELECT TO USER Admin ON B;
```

Similarly, the rule `A2` can be enforced by *column-level* GRANT statement as follows:

```
GRANT SELECT TO USER Admin ON A(C);
```

However, the rules `A3` and `A4` cannot be enforced using GRANT statement since most relational databases do not have *tuple-level* or *cell-level* authorization yet. One may support such an authorization by first creating a view and then issue a GRANT statement to the created view. For instance, the following statements enforce the rule `A3`.

```
CREATE VIEW tmp AS SELECT * FROM B WHERE D>5;
GRANT SELECT TO USER Admin ON tmp;
```

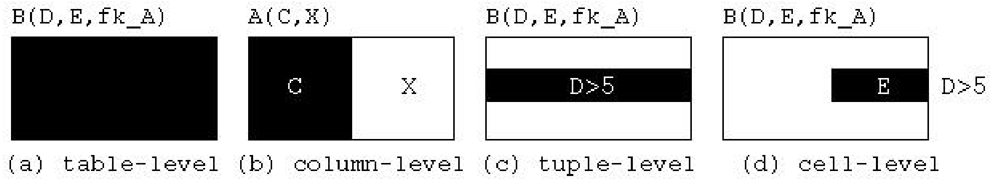

**Fig. 2.** Granularity discrepancy. Dark area refers to the objects being covered by authorization rules. (a) `A1` covers the complete information of the table `B`, (b) `A2` covers the attribute `C` of the table `A`, (c) `A3` covers all the attributes but only tuples satisfying `D>5`, and (d) `A4` covers the attribute `E` of the table `B` but only tuples satisfying `D>5`.

The rule `A4` can be supported similarly. Although finer-grained access controls of XML security models (e.g., tuple-level or cell-level) can be supported using views in relational databases, this scheme shares the same limitations of handling a large number of views – difficulty or inefficiency of maintenance and update. Therefore, it is not entirely clear how to efficiently support such finer-grained access controls in relational databases.

## 4 Illustrative Example

In this section, we will go over an example of XML document with authorization rules and illustrate how that might be supported using one of the conversion methods. Tables 2 and 3 are the DTD for `AllDepts` and an XML document conforming to the DTD. Furthermore, Table 4 contains several authorization rules in the XML security model proposed by [6]. Note that the scope of the authorization rules is specified by XPath expressions, and thus node-level fine-grained access control is allowed in that model. In that table, sign "-" and "+" mean the action is "prohibited" and "allowed", respectively, and type "R" and "L" mean the propagation of the rule is "recursive" and "local", respectively. That is, the recursive propagation implies the rule applies to nodes $N$ specified by the XPath expression as well as all the descendants of $N$. In the local propagation, the authorization rule applies to only nodes specified by the XPath expression. For instance, the rule `R1` states that "No public read/write is allowed for the attribute `dname` under the element `AllDepts`, as a child or descendant." Also, the rule `R4` states that "Manager can read/write the `Budget` element under the `Proj` element when the `Proj` has a `private` as a value for the attribute `type`."

Now, suppose the XML document in Table 3 is converted and stored in relational databases using XRel method [25] as shown in Table 5. Note that what is shown in Table 5 is a simplified version of the original XRel algorithm for simplicity. In XRel, all root-to-leaf paths of XML document, where leaf is either attribute or element, are assigned a unique ID (i.e., pathID) and stored in the path table (d). Then, each node (i.e., text string, attribute value, and element) of the XML document is captured in the appropriate table separately (i.e., text table (b), attribute table(a), and element table (c)).. For instance,

**Table 2.** XML side: A DTD for `AllDepts`.

```
<!DOCTYPE AllDepts [
  <!ELEMENT Dept    (Manager,Staff+,Proj*)>
  <!ATTLIST Dept     dname   ID          #REQUIRED>
  <!ELEMENT Manager (Name,Addr,Salary)>
  <!ATTLIST Manager  eid     ID          #REQUIRED>
  <!ELEMENT Staff   (Name,Addr,Salary)>
  <!ATTLIST Staff    eid     ID          #REQUIRED>
  <!ELEMENT Proj    (Year,Budget)>
  <!ATTLIST Proj     pname   ID          #REQUIRED
                     type   (public|private) #IMPLIED>
  <!ELEMENT Year    (#PCDATA)>
  <!ELEMENT Budget  (#PCDATA)>
]>
```

**Table 3.** XML side: An XML document for `AllDepts`. Note that `[XX]` in front of elements, attributes, or text string below is not part of the XML document, but a node ID number added for the discussion of this paper.

```
<[1]AllDepts>
  <[2]Dept [3]dname='CS'>
    <[4]Manager [5]eid='m10'>
      <[6]Name>[7]Tom</Name> <[8]Addr>[9]110 Foster Ave.</Addr>
      <[10]Salary>[11]70K</Salary>
    </Manager>
    <[12]Staff [13]eid='e10'>
      <[14]Name>[15]Jane</Name> <[16]Addr>[17]54 Union St.</Addr>
      <[18]Salary>[19]45K</Salary>
    </Staff>
    <[20]Proj [21]pname='XML' [22]type='public'>
      <[23]Year>[24]2003</Year>  <[25]Budget>[26]100K</Budget>
    </Proj>
    <[27]Proj [28]pname='Stream' [29]type='private'>
      <[30]Year>[31]2002</Year>  <[32]Budget>[33]300K</Budget>
    </Proj>
  </Dept>
</AllDepts>
```

**Table 4.** XML side: Authorization rules for `AllDepts`.

| No. | Subject | Object | Action | Sign | Type |
|---|---|---|---|---|---|
| R1 | Public | `/AllDepts/*/@dname` | read,write | - | L |
| R2 | Public | `//Dept/*/Name` | read | + | L |
| R3 | Manager | `//Dept/Staff` | read | + | R |
| R4 | Manager | `//Dept/Proj[./@type='private']/Budget` | read,write | + | L |
| R5 | Staff,Manager | `//Dept/Proj[./@type='public']/Budget` | read | + | L |

the node with nodeID=4 (i.e., `Manager`) is captured in the third tuple of the element table (c) using the `/AllDepts/Dept/Manager` path. Similarly, the node with nodeID=17 (i.e., `54 Union St.`) is captured in the fifth tuple of the text table (b) using the `/AllDepts/Dept/Staff/Addr` path.

**Table 5.** RDBMS side: Four (i.e., attribute, text, element, and path) tables generated from the XML document of Table 3 by XRel.

(a) Attribute table

| pathID | value | nodeID |
|---|---|---|
| 3 | CS | 3 |
| 5 | m10 | 5 |
| 10 | e10 | 13 |
| 15 | XML | 21 |
| 16 | public | 22 |
| 15 | Stream | 28 |
| 16 | private | 29 |

(b) Text table

| pathID | value | nodeID |
|---|---|---|
| 6 | Tom | 7 |
| 7 | 110 Foster Ave. | 9 |
| 8 | 70K | 11 |
| 11 | Jane | 15 |
| 12 | 54 Union St. | 17 |
| 13 | 45K | 19 |
| 17 | 2003 | 24 |
| 18 | 100K | 26 |
| 17 | 2002 | 31 |
| 18 | 300K | 33 |

(c) Element table

| pathID | index | rindex | nodeID |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 2 |
| 4 | 1 | 1 | 4 |
| 6 | 1 | 1 | 6 |
| 7 | 1 | 1 | 8 |
| 8 | 1 | 1 | 10 |
| 9 | 1 | 1 | 12 |
| 11 | 1 | 1 | 14 |
| 12 | 1 | 1 | 16 |
| 13 | 1 | 1 | 18 |
| 14 | 1 | 2 | 20 |
| 17 | 1 | 1 | 23 |
| 18 | 1 | 1 | 25 |
| 14 | 2 | 1 | 27 |
| 17 | 1 | 1 | 30 |
| 18 | 1 | 1 | 32 |

(d) Path table

| pathID | pathExpr |
|---|---|
| 1 | `/AllDepts` |
| 2 | `/AllDepts/Dept` |
| 3 | `/AllDepts/Dept/@dname` |
| 4 | `/AllDepts/Dept/Manager` |
| 5 | `/AllDepts/Dept/Manager/@eid` |
| 6 | `/AllDepts/Dept/Manager/Name` |
| 7 | `/AllDepts/Dept/Manager/Addr` |
| 8 | `/AllDepts/Dept/Manager/Salary` |
| 9 | `/AllDepts/Dept/Staff` |
| 10 | `/AllDepts/Dept/Staff/@eid` |
| 11 | `/AllDepts/Dept/Staff/Name` |
| 12 | `/AllDepts/Dept/Staff/Addr` |
| 13 | `/AllDepts/Dept/Staff/Salary` |
| 14 | `/AllDepts/Dept/Proj` |
| 15 | `/AllDepts/Dept/Proj/@pname` |
| 16 | `/AllDepts/Dept/Proj/@type` |
| 17 | `/AllDepts/Dept/Proj/Year` |
| 18 | `/AllDepts/Dept/Proj/Budget` |

Note that no authorization information of Table 4 is captured by the XRel conversion. Therefore, to support access controls of XML model using XRel method, one needs to *somehow* carry over the authorization rules of Table 4 into relational forms, too.

Among many possible approaches, one extreme is *not* to store any accessibility information and instead directly use Table 4 in evaluating access controls. That is, this process of security evaluation is entirely conducted outside of relational databases (i.e., external security evaluation) and has to first evaluate the XPath expressions of Table 4 to find out all XML nodes being enforced. Another variation is to store a list of XML nodes for each authorization rule in an auxiliary table. For instance, for the rule `R2` of Table 4, the XPath expression `//Dept/*/Name` is evaluated first. Then, two path expressions (i.e., pathID is 3 and 11) stored in the path table (d) of Table 5 are identified as matches and stored instead in a table. An example auxiliary table is shown in Table 6.

**Table 6.** RDBMS side: A pathID-based auxiliary table for `AllDepts` of Table 4.

| Subject | pathID | Action | Sign | Type |
|---|---|---|---|---|
| Public | 3 | SELECT,UPDATE | - | L |
| Public | 6, 11 | SELECT | + | L |
| Manager | 9,10,11,12,13 | SELECT | + | R |

One limitation of this approach is that this cannot handle the so-called *twig query* such as rules `R4` and `R5` of Table 4 that have the filtering condition. That is, in the `R4`, the XPath expression `//Dept/Proj[./@type='private']/Budget` is essentially a tree with two branches, `//Dept/Proj/Budget` and `//Dept/Proj/@type='private'`, with the node `Budget` being projected at the end. Since the path table (d) of Table 5 has only root-to-leaf paths, it is not straightforward to handle such a case. Therefore, to support such a twig case of authorization rules, one can instead store node IDs in the auxiliary table as shown in Table 7, where two rules `R4` and `R5` of Table 4 are captured with proper node IDs. This approach is essentially similar to the *materialization* method (e.g., [26]) where each user (or role) separately keeps a list of XML nodes that he/she is allowed to access, according to access controls. This approach becomes problematic when the number of rules in Table 4 is huge, although it can be alleviated using a space-efficient method like CAM of [26]. The more serious problem of this approach is that after the original XML data are stored in a relational database, the XML nodes to materialze are scattered among tables, making it difficult to efficiently keeping track of.

Note that neither of two presented schemes can be implemented in the basic relational security features, since these require *value-based* or *content-based* security constraint. Therefore, it would be challenging to investigate how to support such schemes using table-level or column-level relational security constraint.

Suppose a manager "Tom" wants to retrieve department names as follows:

```
/AllDepts/Dept//Name
```

Since the `Name` element is accessible to public, no security check is needed and the input XPath is translated to the following SQL according to XRel algorithm:

**Table 7.** RDBMS side: A nodeID-based auxiliary table for `AllDepts` of Table 4.

| Subject | nodeID | Action | Sign | Type |
|---|---|---|---|---|
| Manager | 33 | SELECT,UPDATE | + | L |
| Staff,Manager | 26 | SELECT | + | L |

**Table 8.** Main issues and choices used in the example throughout Section 4. Choices made are $\boxed{\text{boxed}}$ .

| Issue | Choice |
|---|---|
| Target model | model-to-model vs. $\boxed{\text{model-to-RDBMS}}$ |
| XML security model | $\boxed{\text{schema-level}}$ vs. instance-level vs. both |
| XML-to-Relational conversion | structure-based vs. $\boxed{\text{data-based}}$ |
| Security check location | external vs. $\boxed{\text{internal}}$ vs. hybrid |
| Security check time | post-pruning vs. $\boxed{\text{pre-pruning}}$ vs. intermixed |

```
SELECT e.nodeID
FROM   Element e, Path p
WHERE  p.pathExpr LIKE '/AllDepts/Dept%/Name' AND
       e.pathID = p.pathID
```

Secondly, suppose a regular user "Jane" wants to retrieve salary information of staffs as follows:

```
/AllDepts/Dept/Staff/Salary
```

When an auxiliary table such as Table 6 is given, then, this XPath could be re-written to the following SQL query, where the last WHERE condition "`a.pathID = p.pathID`" ensures valid access control.

```
SELECT e.nodeID
FROM   Element e, Path p, Auxilary a
WHERE  p.pathExpr LIKE '/AllDepts/Dept/Staff/Salary' AND
       e.pathID = p.pathID AND a.pathID = p.pathID
```

The presented example so far and its choices made from the main issues discussed in Section 3 can be summarized as shown in Table 8. As illustrated, choices that we pick in this example is only one of many possible approaches, and more study is needed to understand detailed pros/cons and behaviors of them.

## 5 Conclusion

In this paper, we explore the research issues on how to support access controls of XML data by leveraging existing techniques in relational databases. We envisage an XML data management system in which (1) users make XML queries against a given XML schema; (2) access controls for XML data are also specified in a XML security model; and (3) Data are stored in relational databases. In such a system, while users view and access data based on an XML data model, the fact that the data is stored and processed in a relational database system is made transparent to the users. We present a framework for processing XML queries in the above system and examine various research issues appeared in the framework. In this paper, we discuss several important problems in terms of converting XML data to relational representation and storage, converting XML security models to relational access control mechanisms and metadata, and security evaluations. We sketch how to resolve the above problems and glue various components in our framework to efficiently support access controls for the XML model using relational databases. We also use examples to illustrate the various issues we discussed in the paper. While this paper, as a preliminary work, points out a vitally important research direction, we believe that more in-depth studies are needed to realize our vision.

## References

1. S. Banerjee, V. Krishnamurthy, M. Krishnaprasad, and R. Murthy. "Oracle8i - The XML Enabled Data Management System.". In *IEEE ICDE*, San Diego, CA, Feb. 2000.
2. E. Bertino and E. Ferrari. "Secure and Selective Dissemination of XML Documents". *IEEE Trans. on Information and System Security (TISSEC)*, 5(3):290–331, Aug. 2002.
3. R. Bourret. "XML and Databases". Web page, Sep. 1999. http://www.rpbourret.com/xml/XMLAndDatabases.htm.
4. T. Bray, J. Paoli, and C. M. Sperberg-McQueen (Eds). "Extensible Markup Language (XML) 1.0 (2nd Edition)". W3C Recommendation, Oct. 2000. http://www.w3.org/TR/2000/REC-xml-20001006.
5. J. M. Cheng and J. Xu. "XML and DB2". In *IEEE ICDE*, San Diego, CA, Feb. 2000.
6. E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. "A Fine-Grained Access Control System for XML Documents". *IEEE Trans. on Information and System Security (TISSEC)*, 5(2):169–202, May 2002.
7. E. Damiani, S. De Capitani Di Vimercati, S. Paraboschi, and P. Samarati. "Design and Implementation of an Access Control Processor for XML Documents". *Computer Networks*, 33(6):59–75, 2000.
8. A. Deutsch, M. F. Fernandez, and D. Suciu. "Storing Semistructured Data with STORED". In *ACM SIGMOD*, Philadephia, PA, Jun. 1998.
9. E. Fernandez, E. Gudes, and H. Song. "A Model of Evaluation and Administration of Security in Object-Oriented Databases". *IEEE Trans. on Knowledge and Data Engineering (TKDE)*, 6(2):275–292, 1994.
10. D. Florescu and D. Kossmann. "Storing and Querying XML Data Using an RDBMS". *IEEE Data Eng. Bulletin*, 22(3):27–34, Sep. 1999.
11. S. Godik and T. Moses (Eds). "eXtensible Access Control Markup Language (XACML) Version 1.0". OASIS Specification Set, Feb. 2003. http://www.oasis-open.org/committees/xacml/repository/.

12. P. P. Griffiths and B. W. Wade. “An Authorization Mechanism for a Relational Database System”. *ACM Trans. on Database Systems (TODS)*, 1(3):242–255, Sep. 1976.
13. S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. “Flexible Support for Multiple Access Control Policies”. *ACM Trans. on Database Systems (TODS)*, 26(2):214–260, Jun. 2001.
14. S. Jajodia, P. Samarati, V. S. Subrahmanian, and E. Bertino. “A Unified Framework for Enforcing Multiple Access Control Policies”. In *ACM SIGMOD*, pages 474–485, May 1997.
15. S. Jajodia and R. Sandhu. “Toward a Multilevel Secure Relational Data Model”. In *ACM SIGMOD*, May 1990.
16. D. Lee and W. W. Chu. “Constraints-preserving Transformation from XML Document Type Definition to Relational Schema”. In *Int'l Conf. on Conceptual Modeling (ER)*, pages 323–338, Salt Lake City, UT, Oct. 2000.
17. W.-C. Lee, G. Mitchell, and X. Zhang. “Integrating XML Data with Relational Databases”. In *IEEE Int'l Workshop on Knowledge Discovery and Data Mining in World Wide Web*, Taipei, Taiwan, Apr. 2000.
18. F. Rabitti, E. Bertino, and G. Ahn. “A Model of Authorization for Next-Generation Database Systems”. *ACM Trans. on Database Systems (TODS)*, 16(1):89–131, 1991.
19. P. Samarati, E. Bertino, and S. Jajodia. “An Authorization Model for a Distributed Hypertext System”. *IEEE Trans. on Knowledge and Data Engineering (TKDE)*, 8(4):555–562, 1996.
20. R. Sandhu and F. Chen. “The Multilevel Relational (MLR) Data Model”. *IEEE Trans. on Information and System Security (TISSEC)*, 1(1), 1998.
21. R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. “Role-Based Access Control Models”. *IEEE Computer*, 29(2), 1996.
22. J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. “Relational Databases for Querying XML Documents: Limitations and Opportunities”. In *VLDB*, Edinburgh, Scotland, Sep. 1999.
23. K.-L. Tan, M. L. Lee, and Y. Wang. “Access Control of XML Documents in Relational Database Systems”. In *Int'l Conf. on Internet Computing (IC)*, Las Vegas, NV, Jun. 2001.
24. M. Winslett, K. Smith, and X. Qian. “Formal Query Languages for Secure Relational Databases”. *ACM Trans. on Database Systems (TODS)*, 19(4):626–662, 1994.
25. M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemura. “XRel: A Path-Based Approach to Storage and Retrieval of XML Documents using Relational Databases”. *ACM Trans. on Internet Technology (TOIT)*, 1(2):110–141, Nov. 2001.
26. T. Yu, D. Srivastava, L. V.S. Lakshmanan, and H. V. Jagadish. “Compressed Accessibility Map: Efficient Access Control for XML”. In *VLDB*, Hong Kong, China, 2002.