Storing XML documents and XML policies in Relational Databases

A. A. Abd EL-Aziz
Research Scholar
Dep. of Information Science & Technology
Anna University
Email: zizoah2003@gmail.com

A.kannan
Professor
Dep. of Information Science & Technology
Anna University
Email: kannan@annauniv.edu

***Abstract*—In this paper, We explore how to support security models for XML documents by using relational databases. Our model based on the model in [6], but we use our algorithm to store the XML documents in relational databases.**

I. INTRODUCTION

There are a lot of researches about XML databases, but the proposed approaches are not yet practical. However, these researches do not consider the fact that the great amount data and access control information for XML documents is still stored into relational databases. The problem in native XML databases is that most native XML databases can only return the data as XML. (A few support the binding of elements or attributes to application variables.) If an application needs the data in another format (which is likely), it must parse the XML before it can use the data. This is clearly a disadvantage for local applications that use a native XML database instead of a relational database, as it incurs overhead not found in (for example) an ODBC application. The limitations of native XML security models are summarized as follows[6]:

- **Poor Practicality**: The great amount of data is still stored into relational databases and there are few commercial XML databases.
- **Lack Stability**: Native XML security policy is poor in stability. However, RDB-based security models are proved to be stable by many researches and practical uses.
- **Simple User-based Security Policy**: Native XML security models only provide simple user-based security policy.

Therefore, our goal in this paper is to study how to support XML security models more effectively than conventional RDB-based XML security models by utilizing security support of relational security models.

II. RELATED WORK

To solve limitations of native XML security models mentioned above, many researches about XML security model using relational database have been executed. [5] presents a XENA (XML sEcurity eNforcement Architecture) which stores XML documents as relational tables with pre-processing method. [4] suggests XML access control method with XACT (XML Access Control Tree). XACT is a tree which stores access control information for each node. However, because XACT must be created for each node in XML documents, if the size of XML documents is increased, the creating costs of XACT can be exorbitant. [2], [3] suggests QFilter which provides XML access control by shared NFA and comparison evaluation between pre-processing and post-processing.

III. TRANSFORMING DTD TO RELATIONAL SCHEMA

In this section, we describe our technique to map a given XML DTD to a relational schema. Our technique consists of eight steps: (1) transform the DTD to Xschema, (2) Simplify the Xschema constraints, (3) inlining of elements and attributes, (4) handling key constraints, (5) mapping collection types, (6) mapping IDREF and IDREFS attributes, (IDREFS attributes are treated similar to child elements), (7) handling the union type(or), (8) capturing the order specified in the XML model.

A. Transforming DTD to Xschema

First we define XSchema, a language independent formalism to specify XML schemas[7]. To define XSchema, we first assume the existence of a set $\hat{E}$ of element names, a set $\hat{A}$ of attribute names and a set $\hat{\tau}$ of atomic data types (e.g., ID, IDREF, IDREFS, string, integer, date, etc). Xschema is a structural specification of an XML schema with specification of data types, attribute definitions, and inclusion dependency constraints. Further attributes of types IDREF and IDREFS identify the target types referred to by the values.

Definition 1. An XSchema is denoted by 5-tuple X =(E, A, M, P, r), where:

- *E is a finite set of element names in* $\hat{E}$,
- *A is a function from an element name* $e \in E$ *to a set of attribute names* $a \in \hat{A}$,
- *M is a function from an element name* $e \in E$ *to its element type definition: i.e., M(e) =*α*, where* α *is a regular expression:* $\alpha ::= \epsilon \mid \tau \mid \alpha + \alpha \mid \alpha,\alpha \mid \alpha^*$*, where* ϵ *denotes the empty element,* $\tau \in \hat{\tau}$*, "+" for the union, "," for the concatenation,* α^* *for the Kleene star,* $\alpha^?$ *for* $(\alpha + \epsilon)$ *and* α^+ *for* (α,α^*),
- *P is a function from an attribute name a to its attribute type definition: i.e., P(a) =* β*, where* β *is a 4-tuple (*τ*, n, d, f), where* $\tau \in \hat{\tau}$*, n is either "?" (nullable) or "*$\neg$*?"*

(not nullable), d is a finite set of valid domain values of a or ϵ if not known, and f is a default value of a or ϵ if not known. Further more, if τ is IDREF or IDREFS, then τ also specifies the target type or types that the attribute value should refer to using the symbol "$\longrightarrow$"

- *$r \subseteq E$ is a finite set of root elements,*

Example 1. The following is the DTD for a Conference:

```
⟨!DOCTYPE Conference [
  ⟨!ELEMENT conf (title,date,editor?,papers*)⟩
  ⟨!ATTLIST  conf  id   ID    # REQUIRED⟩
  ⟨!ELEMENT title (# PCDATA)⟩
  ⟨!ELEMENT date EMPTY ⟩
  ⟨!ATTLIST date  year  CDATA  # REQUIRED
                  mon   CDATA  # REQUIRED
                  day   CDATA  # IMPLIED
  ⟨!ELEMENT editor (person*)⟩
  ⟨!ATTLIST editor eids IDREFS # IMPLIED⟩
  ⟨!ELEMENT paper(title,contact?,author,cite?)⟩
  ⟨!ATTLIST  paper  id    ID  # REQUIRED⟩
  ⟨!ELEMENT contact EMPTY ⟩
  ⟨!ATTLIST contact aid IDERF# REQUIRED⟩
  ⟨!ELEMENT author (person+)⟩
  ⟨!ELEMENT person (name,(email|phone)?)⟩
  ⟨!ATTLIST person id  ID # REQUIRED⟩
  ⟨!ELEMENT name EMPTY ⟩
  ⟨!ATTLIST name fn  CDATA # IMPLIED⟩
                 ln CDATA  # REQUIRED⟩
  ⟨!ELEMENT email (# PCDATA) ⟩
  ⟨!ELEMENT phone (# PCDATA) ⟩
  ⟨!ELEMENT cite (papers*) ⟩
  ⟨!ATTLIST cite id  ID #REQUIRED⟩
          format (ACM|IEEE) #IMPLIED⟩
   ]⟩
```

The Xschema for DTD conference is as follows:
E = {conf, title, date, editor, paper, contact, author, person, name, email, phone, cite}, A(conf)={id}, M(conf)=(title, date, editor$^?$ paper*), P(conf.id)=(ID, $\neg$?, ϵ, ϵ), M(title)=(string), A(date)={year, mon, day}, M(date)=ϵ, A(editor)={eids}, M(editor)=(person*), P(editor.eids)=(IDREFS$\longrightarrow$person*, ?, ε, ϵ), A(paper)={id}, M(paper)=(title, contact$^?$, author, cite$^?$), P(paper.id)=(ID, $\neg$?, ε, ε), A(contact)={aid}, M(contact)=ϵ, P(contact.aid)=(IDREF $\longrightarrow$ person, $\neg$?, ϵ, ϵ), M(author)=(person*), A(person)={id}, M(person)=(name,(email + phone)$^?$), P(person:id)=(ID,$\neg$?, ϵ, ϵ), A(name)={fn, ln}, M(name)=ϵ, P(fn)=(string, $\neg$?, ϵ, ϵ), P(ln)=(string, $\neg$?, ϵ, ϵ), M(email)=(string), M(phone)=(string), A(cite)={id, format}, M(cite)=(paper*), P(cite:id)=(ID,$\neg$?, ϵ, ϵ), P(format)=(string, $\neg$?, (ACM|IEEE), ϵ), r={conf}

B. Simplify the Xschema constraints

Since the relational model cannot capture all the constraints specified in the XSchema, then we try to simplify the Xschema to transform it to relational schema. Our schema simplification step is based on the following principles[10], [1]:

$$(e1,e2)^* \longrightarrow e1^*, e2^*$$
$$(e1,e2)^? \longrightarrow e1^?, e2^?$$
$$(e1|e2) \longrightarrow e1^?, e2^?$$
$$e1^{**} \longrightarrow e1^*$$
$$e1^{*?} \longrightarrow e1^*$$

Where e1, e2 and a are subelements.
Example 2. In the above Xschema M(person) = (name, (email + phone)$^?$) is simplified to be M(person) = (name, email$^?$, phone$^?$).

C. Inlining

Our inlining technique creates one relation for an element instead of creating more relations corresponding to one element which is performed in [7]. Inlining is used to generate more meaningful and efficient relational schemas. In inlining, we consider attributes of descendants of an element as attributes in the relation corresponding to that element. Inlining for an element (e) is done recursively using the inline technique described below. Inline technique returns a relation that should be generated for an input element currEl. The inlining technique also takes as input attSet which is used to maintain the list of attributes of (e) that should be present in the relation generated for (e). To inline the element (e), we call inline, where the initialization is: currEl= e, attSet = ϕ.

```
inline : currEl, attSet $\longrightarrow$ ResultSet
1. Assign the set of attributes in A(currEl) except IDREF and IDREFS  attributes to attSet.
2. Set ResultSet = $\phi$.
3. Let the elements which occurs in M(currEl) with occurrence constraints (1,1) or (1,0) after simplification be $\{e_1,e_2,...e_n\}$.
    3.1. For each $e_i$, do the following.
        3.1.1. if $M(e_i) \in \hat{\tau}$, then attSet = attSet $\cup$ $\{e_i\}$.
        3.1.2. else attSet = attSet $\cup$ inline$\{e_i, \phi\}$.
    3.2. if attSet = $\phi$, attSet = {currEl}.
4. ResultSet = ResultSet $\cup$ attSet.
5. Return ResultSet.
```

The inlining technique is applied to the top elements which are determined by the following rules[8]:
Rule1: An element which does not appear in any other element type definition (such as *conf*).
Rule2: A non#PCDATA element which appears in more than one other element type definition.
Rule3: A non#PCDATA element B which appears in another element type definition A with "*" or "+" operators (such as paper, person).
Rule4: If recursion occurs, one of the elements in the recursion is selected as a top element.
Example 3. According to the above rules we find that the element nodes are `conf` (according to rule 1), `paper` (according to rule 3, 4) and `person` (according to rule 3), so by performing inlining on `conf`, `paper`, and `person`, we obtain the following relation definitions
`conf`(id, title, year, mon, day, editor), `paper`(id, title, contact, author, cite_id, cite_format), `person`(id, fn, ln, email, phone).

D. Handling key constraint

In each relation that is created from the previous step, add an attribute (code), its values are 1,2,3...,etc., as a primary key

for each relation.
Example 4. The relation definitions will be conf(code, id, title, year, mon, day, editor), paper(code, id, title, contact, author, cite_id, cite_format), person(code, id, fn, ln, email, phone).

E. Mapping collection types

1. If there is a table corresponding to the collection type, then adds a foreign key refers to the table that represents or contains its parent.
2. Else create a new table corresponding to the collection type, and add a foreign key refers to the table that represents or contains its parent.
3. If the parent of the collection type say (e) is an attribute in a table and A(e)=Φ, then remove it form that table.

Example 5.
1. Since M(conf)=(title, date, editor$^?$ paper*), and there is a table corresponding to paper element, then we add a foreign key to the conf table in paper table. So the paper table will be paper(code, id, title, contact, author, cite_id, cite_format, conf).
2. Since M(editor)=(person*), and there is a table corresponding to person element, then we add a foreign key to the conf table in person table . So the person table will be person(code, id, fn, ln, email, phone, conf).
3. Since M(cite)=(paper*), and there is a table corresponding to paper element, then we add a foreign key to the paper table in paper table. So the paper table will be paper(code, id, title, contact, author, cite_id, cite_format, conf, paper).
4. Since M(author)=(person*), and there is a table corresponding to person element, then we add a foreign key to the paper table in person table and remove the author attribute form paper table. So the person table will be person(code, id, fn, ln, email, phone, conf, paper) and paper table will be paper(code, id, title, contact, cite_id, cite_format, conf, paper).

F. Mapping IDREF and IDREFS attributes

1. An IDREF attribute is mapped by replacing it by a foreign key.

Example 6. We have an IDREF attribute defined for contact, which refers to person. So the result of our mapping is defining the contact attribute in paper table as a foreign key refers to person table.
2. IDREFS attributes are mapped by creating a new table contains a foreign key to the referenced table and a foreign key for the table that represent the element which contains the IDREFS attribute, then removing the IDREFS attribute form that table.
Example 7. We have conf(code, id, title, year, mon, day, editor), A(editor)= {eids}, p(eids)=(IDREFS $\longrightarrow$ person*, ¬?, ε, ε), and person(code, id, fn, ln, email, phone, conf, paper). So we create a new table editor(conf, person) and remove the editor attribute form conf table, so conf table will be conf(code, id, title, year, mon, day).

G. Handling the union type

In this step, if we find more than one attribute that may be NULL in a table say a1 and a2, do the following step:
1. Replace them with two attributes, one of them as a flag attribute and the second for values its name is a1_ a2
2. Add the flag attribute to the key of the table.

Example 8. We have person(code, id, fn, ln, email, phone, conf, paper), since email and phone attributes may be NULL, so we replace them with a flag attribute that added to the key and email_phone attribute, then we will have person(code, flag, id, fn, ln, email_phone, conf, paper).

H. Capturing order specified in the XML model

To capture document order in relational database system, we encode each element's position in an XML document as a data value by using Dewey order method[11]. With Dewey order, each element is assigned a vector that represents the path from the document's root to the element. We store the order of elements in XML DTD in two tables, these tables are meta data that will be used in mapping relational query result to XML documents.
Example 9. For DTD conference, We create the following tables:

pathId	*ele_name*	*parentId*
1	conf	
1.1	title	1
1.2	date	1
1.3	editor	1
1.3.1	person	1.3
1.4	paper	1
1.4.1	title	1.4
1.4.2	contact	1.4
1.4.3	author	1.4
1.4.4	cite	1.4
1.4.3.1	person	1.4.3
1.4.4.1	paper	1.4.4

parent	*sub_ele*	*order*
person	name	1
person	email	2
person	phone	3

IV. Storage Schema of Access Control

The storing of XML access control rules is done as the following in table 1 [6]:

TABLE I
STORAGE SCHEMA OF ACCESS CONTROL RULES DEFINED BY THE SECURITY ADMINISTRATOR

R ID	Subj	Role	Type	Path	Att	Value	Level
1	Bob	R+	R	Order/Order _info /Ad //dr	City	Cairo	.
2	Bob	W+	R	Order/Customer_info	.	.	2
3	Jane	R-	L	Order/Customer_info /Credit_card	Name	Not Jane Jane	.

The table for access control rules consists of 8 attributes. The *RuleID* attribute is an identifier of each access control rule and the *subj* attribute is a subject of access control rule. The *Role* attribute is a positive/negative role of the subject and can have R(read) or W(write) or U(update)values. Besides, + or - represent positive or negative roles. The *Type* attribute is an access control scope and can have R(recursive) or L(local) value. The recursive node includes indicated node and all child nodes of that node. However, the local node only includes indicated node. The *Path* attribute is a path information of node that the access control is applied to. The *Att* attribute is attribute information and the Value attribute include values related to the attribute. The *Level* attribute represents that the described rule is applied to the indicated node and child node represented as the level value. That is, level value 1 means that this access control rule is applied to the only indicated node, same as local mode. As the table 1, level value 2 means that this access control rule is applied to the indicated node and child nodes, not descendant. The above storage schema is a beginning schema defined by the security administrator and changed recursive form into local form automatically as Table 2.

TABLE II
AUTOMATICALLY TRANSLATED STORAGE SCHEMA OF ACCESS CONTROL RULES

RuleID	Subj	Role	Type	Path	Att	Value
1	Bob	R+	L	Order/Order _info /Ad //dr	City	Cairo
2	Bob	R+	L	Order/Order _info /Ad //dr/city	City	Cairo
3	Bob	R+	L	Order/Order _info /Ad //dr/Zip	City	Cairo
4	Bob	W+	R	Order/Customer_info//	.	.
5	Bob	W+	R	Order/Customer_info//name	.	.
6	Bob	W+	R	Order/Customer_info//phone	.	.
7	Bob	W+	R	Order/Customer_info/Add/	.	.
8	Bob	W+	R	Order/Customer_info// Credit_card	.	.
9	Jane	R-	L	Order/Customer_info /Credit_card	Name	Jane

V. CONCLUSION

In this paper, we suggested the RDB-based XML access control model . We envisage an XML data management system in which:

1) access control rules for XML data are specified in a relational database.
2) XML data are stored into a relational database.

The technique suggested in this paper can have the following contribution:

1) **Practicality**: The technique can support more practical access control processing by using relational database.
2) **Stability**: The storing of the access control techniques guarantees better stability than conventional XML access control models.
3) **Performance**: because we store XML data into the relational database, when user queries are give, we do not need to load all of XML documents. [9].

REFERENCES

[1] A. Deutsch, M. F. Fernandez, and D. Suciu. Storing Semistructured Data with Stored. *In proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 431–442, 1999.

[2] B. Luo et al. Pragmatic xml access control using offthe- shelf rdbms. *Computers & Security*, 23, 2004.

[3] B. Luo et al. Qfilter: fine-grained run-time xml access control via nfa-based query rewriting. *In Proceedings of International Conference on Information and Knowledge Management (CIKM), Washington, DC, USA*, November 2004.

[4] J. Jeon et al. Filter xpath expressions for xml access control. *Computer Security (ESORICS), Dresden, Germany*, September 2007.

[5] K.L. Tan et al. Access control of xml documents in relational database systems. *In Proceedings of International Conference on Internet Computing (ICIC 01), Las Vegas, NV*, June 2001.

[6] Jinhyung Kim, Dongwon Jeong, and Doo-Kwon Baik. A vision: Rdb-based xml security models considering data levels. *In Proceedings of 2nd International Conference on Future Generation Communication and Networking*, pages 356 – 361, 13-15 Dec. 2008.

[7] M. Mani and D. Lee. XML to Relational Conversion using Theory of Regular Tree Grammars. *In Proceedings of the 28th VLDB Conference,Hong Kong, China*, 2000.

[8] Y. Men-hin and A. Wai chee Fu. From XML to Relational Database. *In Proceedings of the 8th International Workshop on Knowledge Representation meets Databases (KRDB)*, September 15, 2001.

[9] Yuanbo Qu, XiaoGuang Hong, and Ji Feng. An approach to construct secure view for xml. *In Proceedings of the International Conference on Management and Service Science. MASS '09*, pages 1–4, 2009.

[10] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. *In Proceedings of the 25th VLDB Conference, Edinburgh, Scotland*, pages 302–314, 1999.

[11] I. Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and c. Zhang. Storing and Querying Ordered XML Using a Relational Database System. *In proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 204–215, 2002.