Pragmatic XML Access Control using Off-the-shelf RDBMS

Bo Luo, Dongwon Lee, and Peng Liu*

The Pennsylvania State University
{bluo, dongwon, pxl20}@psu.edu

Abstract. As the XML model gets more popular, new needs arise to specify *access control* within XML model. Various XML access control models and enforcement methods have been proposed recently. However, by and large, these approaches either assume the support of security features from XML databases or use proprietary tools outside of databases. Since there are currently few commercial XML databases with such capabilities, the proposed approaches are not yet practical. Therefore, we explore the problem of "Is is possible to fully support XML access control in RDBMS?" We formalize XML and relational access control models using *deep set* operators. Then we show that the problem of XML AC atop RDBMS is amount to the problem of converting XML deep set operators into *equivalent* relational deep set operators. We show the conversion algebra and identify the properties to ensure the correct conversion. Finally, we present three practical implementations of XML access controls using off-the-shelf RDBMS and their performance results.

1 Introduction

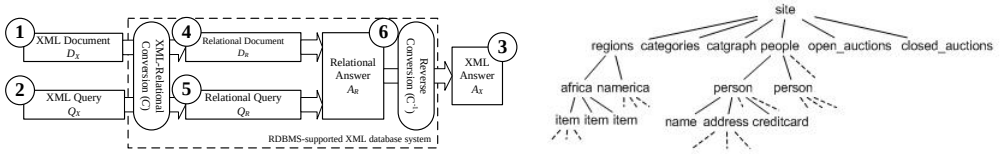
The XML model [9] has emerged as the *de facto* standard for storing and exchanging information in the Internet Age. As more information is exchanged over the Web, the issues of security become increasingly important. Such issues span from data level security to network level security to high-level access controls. In this paper, our focus is on how to support *access control* for XML data. Many access control methods extending the XML model to incorporate security aspects have been proposed recently (e.g., [22, 12, 3, 52]). To the lesser or greater extent, however, XML access control enforcement mechanisms proposed in the research community neglect the fact that the most XML data still resides in RDBMS. In the scenario of RDBMS-backed XML database systems (hereafter *XRDB*), XML documents are stored in RDBMS and query-answering is conducted through a conversion layer. In the scenario of XML publishing, relational data is compiled into XML format for distribution and exchange. For both scenarios, we enjoy the benefit of XML model while taking advantage of the maturity of the off-the-shelf RDBMS. In both scenarios, it is desirable to natively specify access controls on the XML side, but they need to be enforced on the RDBMS. We

* Peng Liu was supported in part by NSF CCR-0233324.

believe that current XML access control enforcement mechanism research is in a sense re-inventing wheels without utilizing existing relational access control models or leveraging on security features that are readily available in relational products. In short, our goal in this paper is to answer the question: *When is it (not) possible to support XML access control using RDBMS? Why? How?*

Challenges. First, the major challenges of supporting XML access control in XRDB systems stem from the inherent discrepancy of XML and relational data models. Relational data model features a structure of two-dimensional table, while XML features a hierarchical data model. When XML data are shredded into relational data model, not all transformation algorithms can fully preserve structural properties of XML model [1]. The inherent incompatibility of two data models leads to the fundamental discrepancy between two access control models. Second, relational access control policies define authorized actions of "cells," where each cell is an impartible element and whose accessibility is explicitly expressed. However, XML nodes are hierarchically nested, and XML data model inherently takes "answer by subtree model" (e.g., querying for `//foo` yields the whole subtree rooting at node `<foo/>`). Therefore, for any XML node, an action could be: authorized/unauthorized to the whole subtree, or partially authorized. The later case does not occur in relational access control model. Finally, in XML model, we can control the access right of each individual node. In traditional relational model, the smallest granularity that we may control is a column via GRANT/REVOKE. Therefore, we need to employ more recent developments of RDBMS access controls (e.g., Oracle VPD) to enable cell level access control.

Key contributions. (1) To our best knowledge, this work is the first one to algebraically formalize XML access control in both native XML (XDB) and XRDB environment. (2) This work takes the first steps to define the *equivalent objects* and *equivalent operations* between native XML and XRDB systems. With this concept, we can migrate all the exciting features of native XML systems into XRDB by converting the atomic operations into equivalent relational counterparts. In this paper, we take the feature of fine-grained XML access control for a pilot study, and the results are encouraging. (3) This work shows for the first time that the "security" of XRDB can be achieved by finding the "equivalent" relational operators for three specific deep-set operators. This finding provides a viable way to build secure XRDB systems. (4) Finally, this work proposes several practical approaches to implement the viable way "discovered" by our theory.



(a) Overview (b) XMark DTD

Fig. 1. Overview architecture and an example XML schema.

2 Related Work

2.1 XML and Relational Access Control

Current access control research can be categorized into two groups: access control modeling and access control enforcement mechanisms.

On the model side, several XML access control models have been proposed. Starting with [47] for HTML documents; [13, 12] describes XML access control with an authorization sheet to each document or DTD. [3] proposed and XML access control model to deal with authorization priorities and conflict resolution. [28] introduced provisional authorization and XACL. [20] formalizes the way of specifying *objects* in XML access control using XPath. Most of the proposals adopt either *role-based access control* (e.g. [53]) or *credential-based access control* (e.g. [4]). The major difference between them is the way they identify users. Credential-based access control is more flexible and powerful in this aspect. However, in the research of access control enforcement mechanisms, people tend to choose a relatively simple access control model to avoid distraction.

XML access control enforcement mechanisms in native XML environment have been intensively studied in recent years. They are categorized into four classes: (1) engine level mechanisms implement security check inside XML database engine; each XML node is tagged with a label [14, 11, 55] or an authorization list [57, 27], and filtered during query processing. (2) view based approaches build security views that only contain access-granted data [51, 16, 29]. [1](3) pre-processing approaches check user queries and enforce access control rules before queries are evaluated, such as the static analysis approach [39, 40], QFilter approach [34], access condition table approach [43], policy matching tree[44], secure query rewrite (SQR) approach [37], etc. (4) [8] considers access control of streaming XML data and apply security check at client side, using a filtering mechanism. More recently, [5, 10] takes encryption issues into consideration, and [18, 38] focus on protecting the privacy and security associated with XML tree structure.

Relational access control models can be classified into two categories: *multi-level security models* [26, 54, 48] and *discretionary security models (DAC)*. Most real world database systems implement a table/column level DAC similar to the one implemented in System R [23]. View-based approaches is the traditional method to enable row-level access control, while Oracle's VPD is the most recent development. Finally, some advanced access control models (e.g., [24, 25]) are proposed in a more theoretical manner.

2.2 XML and Relational Conversion

As illustrated in Figure 1(a), in XRDB: XML data D_X are converted into D_R and stored in RDBMS; user issues XML query Q_X (XPath or XQuery) using published XML schema; Q_X is converted into Q_R (SQL) and evaluated against D_R; relational answer A_R is finally converted to XML (A_X) to return to user.

[1] When a view-based approach implements virtual views without materializing them, it is inherently a pre-processing approach.

Toward conversion between XML and relational models, an array of research has addressed the particular issues lately. On the industry side, database vendors are busily extending their databases to adopt XML types. Shredding and non-shredding are two major pathes that followed by commercial products. Oracle provides both un-shredded (CLOB) and shredded storage options [41]. Microsoft supports XML shredding and publishing through mid-tier approach in SQL Server 2000, and adds CLOB storage in SQL Server 2005 [46]. IBM proposes the first native XML storage in DB2 9, but shredded XML storage (through schema decomposition) is still kept as an important feature [42, 6]. On the research side, various proposals have been made recently, mainly either schema-based (e.g., [15, 50, 30]) or schema-oblivious (e.g., [19, 56]) approaches.

In terms of access control, some commercial products apply existing column level access control of RDBMS on XML data stored in CLOB columns. None of these approaches supports or discusses fine-grained access control. Finally, to our best knowledge, the only work that is directly relevant to our proposal is [52]. [52] proposes an idea of using RDBMS to handle XML access controls, in a rather limited setting. In our vision paper [31], we addressed some issues and challenges of enforcing XML access control atop RDBMS. We provide the algebraic analysis and explore practical solutions in this paper.

Our framework is not tied to a particular conversion method. Throughout this paper, we use shared-inlining [50] and XRel [56] as the examples of schema-based and schema-oblivious conversion methods, respectively. Briefly, XRel decomposes XML documents into document, element, attribute, text, and path tables. In this approach, each node is stored as one record in the `element` table, and each distinct path is stored as one record in the `pth` table. As a simple example, we decompose an XMark ([49], Figure 1(b)) document using XRel and show part of `element` table in Figure 2 (b). As we can see, element 252 is a node of path 164 ("`/site/people`", as stored in the `path` table); which starts from offset 33996 (byte) and ends at 36229 in the original XML document.

3 Preliminaries

3.1 XML Access Control Policy

Access control models define the semantics and syntax of access control policies. Although they could be very complicated, the essential is to describe *subjects*, *objects*, *actions* and all the variations around it. Fortunately, there is no discrepancy in identifying *subjects* and defining *actions* in XML and relational environment, e.g., they both could adopt RBAC or CBAC to identify users. As we described in Section 1, shredding XML access control models into relational ones is a challenging task, because of the fundamental discrepancies of XML and Relational data models. Therefore, challenges reside in *object*-related components of access control models, while issues that only relate to *subjects* and *actions* are trivial. Thus, our subsequent discussion focuses more on *object* part. In this paper, we adopt the model proposed in [12] as the basis; other models like [39, 8, 34, 45] can be used as well with a reasonable change.

Definition 1 (XML Access Control Rule) *An XML access control policy is specified by a set of* **access control rules***:* $R_X = \{\mathbf{subject}, \mathbf{object}, \mathbf{action}, \mathbf{sign}\}$*, where* subject *is to whom an authorization is granted,* object *is a set of XML nodes (in XPath) to which the policy is applied,* action *is one of read, write, or update, and* sign $\in \{+, -\}$ *refers to access granted or denied, respectively.* □

In this model, access is prohibited by default. Negative rule takes precedence when it conflicts with positive rules. All access controls propagate to the entire subtree rooting at *object*, complying with the answer-by-tree XML semantics. If a rule applies to context node only, we add "`/text()`" to its *object* field.

3.2 XML to Relational Conversion

Remark 1 *A relational to XML conversion method contains: (1)* $C_D()$ *to convert XML to relational data, (2)* $C_Q()$ *to convert XML query (XQuery or XPath) to SQL, and (3)* C^{-1} *to convert relational answer back to XML.* □

That is, $Q_R = C_Q(Q_X)$, $D_R = C_D(D_X)$, and $A_X = C^{-1}(A_R)$. From this, the process of "evaluating XML query on XRDB" can be modeled as:

$$A_X = C^{-1}(A_R) = C^{-1}(Q_R\langle D_R\rangle) = C^{-1}(C_Q(Q_X)\langle C_D(D_X)\rangle) \quad (1)$$

Remark 2 *An X2R conversion algorithm is* lossless *iff: (1) (lossless node conversion)* $\forall$ *XML node* $x_i, C_D^{-1}(C_D(x_i)) = x_i$*; (2) (lossless node set decomposition)* $\forall$ *XML node set* $\{x_1, ..., x_n\}, C_D^{-1}(C_D(\{x_1, ...x_n\})) = C_D^{-1}(\{C_D(x_1), ...C_D(x_n)\}) = \{C_D^{-1}(C_D(x_1)), ...C_D^{-1}(C_D(x_n))\}$*; and (3) (exclusive conversion)* $C_D(x_1) = C_D(x_2)$ *only when* $x_1 = x_2$*, and* $C_D^{-1}(r_1) = C_D^{-1}(r_2)$ *only when* $r_1 = r_2$*.* □

Remark 3 *An X2R conversion algorithm is* correct *iff:* $\forall$ *query* Q *and* $\forall$ *document* X*,* $Q\langle X\rangle = C^{-1}(Q_R\langle D_R\rangle) = C^{-1}(C_Q(Q_X)\langle C_D(X)\rangle)$*.* □

Definition 2 (Soundness) *An X2R conversion algorithm A is* ***sound*** *iff it is* lossless *and* correct*.* □

In the remainder of the paper, we assume that the conversion algorithm being used is *sound*. We ignore the order of XML nodes when we compare the correctness, since this feature is not supported in most X2R conversion algorithms.

In the research community, most X2R conversion algorithms support a subset of XQuery/XPath (e.g., /, //, * and predicates). Our approach does not alter the query or data conversion algorithm. Therefore, for a particular X2R conversion method X, we support everything that X supports. For ease of understanding, we do not use predicates in the examples, however, we test queries with predicates in our experiments.

3.3 Deep set operators

In [35], we propose deep set operators for XML, as extensions of conventional set operators defined in XPath [2] and XQuery [7]. Here, we briefly revisit them, and later demonstrate how they are used to formalize XML access control.

Definition 3 (deep set operators) *The* `deep-union` *operator* ($\overset{D}{\cup}$) *takes node sequences* $\langle P\rangle$ *and* $\langle Q\rangle$ *as operands, and returns a sequence of nodes (1) who exist as a node or as a descendant in* ***either*** *operand sequences, and (2) whose parent does not satisfy (1). Formally,* $\langle P\rangle \overset{D}{\cup} \langle Q\rangle = \{n|(n \in \langle P_d\rangle \vee n \in \langle Q_d\rangle) \wedge (n :: parent() \notin \langle P_d\rangle \wedge n :: parent() \notin \langle Q_d\rangle)\}$ *where* $P_d = P/descendant-or-self()$. *The* `deep-intersect` *operator* ($\overset{D}{\cap}$) *takes node sequences* $\langle P\rangle$ *and* $\langle Q\rangle$ *as operands, returns a sequence of nodes (1) who exist as a node or as a descendant in* ***both*** *operand sequences, and (2) whose parent does not satisfy (1). Formally,* $\langle P\rangle \overset{D}{\cap} \langle Q\rangle = \{n|(n \in \langle P_d\rangle \wedge n \in \langle Q_d\rangle) \wedge (n :: parent() \notin \langle P_d\rangle \vee n :: parent() \notin \langle Q_d\rangle)\}$. *Finally, the* `deep-except` *operator* ($\overset{D}{-}$) *takes node sequences* $\langle P\rangle$ *and* $\langle Q\rangle$ *as operands, for each node* $\langle p_i\rangle$ *in* $\langle P\rangle$, *it remove* $\langle p_i\rangle \overset{D}{\cap}_X \langle Q\rangle$ *from the subtree of* $\langle p_i\rangle$ *and return the remaining.* □

3.4 XML access control in XDB and XRDB

XML access control is to ensure that only *safe answer* (SA) is returned. As in [33, 35], safe answer of Q includes all the XML nodes n such that: (1) $n \in \langle Q\rangle$, (2) the access of n is granted by positive rules, and (3) the access of n is *not* denied by negative rules. Therefore, the precise semantics of "safe answer," SA_X is:

$$SA_X = \langle Q_X\rangle \overset{D}{\cap}_X [(\langle R^+_{X_1}\rangle \overset{D}{\cup}_X ... \overset{D}{\cup}_X \langle R^+_{X_n}\rangle) \overset{D}{-}_X (\langle R^-_{X_1}\rangle \overset{D}{\cup}_X ... \overset{D}{\cup}_X \langle R^-_{X_m}\rangle)] \quad (2)$$

Equation (1) models XML query evaluation in XRDB. Similarly, (2) models how only safe XML answers, SA_X, are returned. Combine them, we have:

Definition 4 (Secure XRDB) *An XRDB is secure iff* $\forall$ *ACR set* ACR_X *and* $\forall$ *query* Q_X, *it always returns the safe answer:* $A_X \equiv SA_X$. *Therefore,*

$$C^{-1}(\{C_Q(Q_X)\langle C_D(D_X)\rangle\}') \equiv \langle Q_X\rangle \overset{D}{\cap}_X [(\langle R^+_{X1}\rangle \overset{D}{\cup}_X ...) \overset{D}{-}_X (\langle R^-_{X1}\rangle \overset{D}{\cup}_X ...)] \quad (3)$$

□

Note that $\{C_Q(Q_X)\langle C_D(D_X)\rangle\}'$ indicates that access control mechanism intervenes in relational query processing. Our goal in this paper is to enforce XML access controls on RDBMS so that Equation 3 holds in *XRDB* setting. In this way, we need to convert *access control rules* R_X and *deep set operators* into their *equivalent* relational counterpart.

4 XML Access Control in XRDB: The Theory

All entities of the 4-tuple XML access control model, except *object*, can be directly adopted to relational access control model. We apply an X2R algorithm $C(R_X.object)$ to get $R_R.object$. Therefore, we get "equivalent" relational ACR:

$$\mathbf{R_R} = \{\mathbf{R_X.subject}, \mathbf{C(R_X.object)}, \mathbf{R_X.action}, \mathbf{R_X.sign}\}$$

1. {user, /site/people/person, read, +}
2. {user, /site/people/person/credicard, read, -}

(a)

DOCID	ELEMENTID	PATHID	ST	ED	
0	252	164	33996	36229	<people>
0	293	165	35592	35826	<person>
0	299	165	35832	36217	<person>
0	303	188	35989	36032	<creditcard>

(b)

```
SELECT  e0.DOCID, e0.ELEMENTID, e0.PATHID, e0.ST, e0.ED
FROM document d, element e0, pth p0
WHERE p0.pathexp LIKE '#%/people'
AND e0.pathid = p0.pathid AND d.docid = e0.docid
```

(c)

```
SELECT  e0.DOCID, e0.ELEMENTID, e0.PATHID, e0.ST, e0.ED
FROM document d, element e0, pth p0
WHERE p0.pathexp LIKE '#%/people#/person'
AND e0.pathid = p0.pathid AND d.docid = e0.docid
```

(d)

Fig. 2. Naive enforcement of "equivalent" relational ACR leads to wrong answer.

However, naive enforcement of the converted relational access control rules may lead to security leakage, as demonstrated in the following example:

Example 1. Consider two rules of Fig. 2(a) with XRDB(XRel) – an XRDB employing XRel [56] as the conversion algorithm. The "element" table is partly shown in Fig. 2(b). Rule 1 indicates that a user is allowed to access `<person>` nodes, i.e., nodes 293 and 299 (record 2 and 3 in Fig. 2 (b)), and rule 2 indicates that a user cannot access `<credicard>` nodes, i.e., node 303. Naive enforcement will grant access to the record 2, 3; and revoke the access to record 4.

Query "`//people`" is desired to yield an answer containing two `<person>` nodes, since they are the accessible descendants of the requested node. However, the converted SQL query (Fig. 2(c)) yields no answer since access to record 1 is prohibited by default. Moreover, for a query "`//person`", the converted SQL (Fig. 2(d)) returns both `<person>` nodes to the user (with the unauthorized `<creditcard>` node). This is so because both records of element 293 and 299 are accessible, while revoking access to element 303 does not affect its ancestor. □

4.1 Object and Operation Equivalency

To solve the problem illustrated in Example 1, we propose our framework of supporting access control in XRDB systems. First, we define object and operation equivalency between XML and relational.

Definition 5 (Object Equivalency) *When both $R = C(X)$ and $X = C^{-1}(R)$ hold for XML node set X and relation R, we consider X and R equivalent w.r.t. C/C^{-1}, and denote as $X \equiv R$.* □

Note that, when we talk about equivalency of X and R, we have to predefine the context, i.e., select the X2R conversion algorithm C/C^{-1}. For a XML node set X, $C(X)$ may be different under different X2R conversion algorithms.

Definition 6 (Operation Equivalency) *Suppose $X_1 \equiv R_1$ and $X_2 \equiv R_2$ w.r.t. C/C^{-1}. Then, an XML operation OP_X is equivalent to a relational operation OP_R (denoted as $OP_X \equiv OP_R$) w.r.t. C and C^{-1} if:*

$$C(X_1\ OP_X\ X_2) = C(X_1)\ OP_R\ C(X_2) = R_1\ OP_R\ R_2$$ □

Note that XML operator takes node sets as operands while its equivalent relational counterpart may not take two generic relations as operands. Each operand

is the equivalent objects of corresponding XML node set, which may be tables, columns, records, etc. Relational operations require operands to be domain compatible (e.g., intersect, union etc.). We loosen this requirement for OP_R.

With the concept of operation equivalency, we can migrate all the exciting features of XML into XRDB by converting the atomic operations into equivalent relational operation. Our secure XRDB problem is articulated as follows:

Lemma 1. *In XRDB(C), if we can find relational operators,* $\overset{D}{\cup}_R$, $\overset{D}{\cap}_R$, *and* $\overset{D}{-}_R$, *which are equivalent to XML deep set operators,* $\overset{D}{\cup}_X$, $\overset{D}{\cap}_X$, *and* $\overset{D}{-}_X$, *w.r.t. the X2R conversion algorithm C, we are able to enforce XML access control in XRDB(C) such that Equation (3) always holds.* ∎

Please refer to [36] for detailed proof. Now we need to find equivalent operations such that $\overset{D}{\cup}_R \equiv \overset{D}{\cup}_X$, $\overset{D}{\cap}_R \equiv \overset{D}{\cap}_X$ and $\overset{D}{-}_R \equiv \overset{D}{-}_X$. Again, equivalency is based on specific X2R conversion method, therefore, the existence and representation of relational deep set operators also depends on the particular X2R conversion. Hereafter, we analyze the role of each deep set operator in (2) and the existence of its equivalent relational counterpart under different X2R conversion algorithms.

4.2 On Equivalent Conversion of Deep Set Operators

Deep-union operator is used to integrate all the accessible nodes defined by individual positive rules (also, all the inaccessible nodes defined by negative rules). With the property $P \overset{D}{\cup} Q \subseteq P \cup Q$ [35], Remark 1 is rewritten into:

$$\langle P\rangle \overset{D}{\cup}_X \langle Q\rangle = \{n|(n \in \langle P\rangle \vee\ n \in \langle Q\rangle)\ \wedge (n \notin \langle P//*\rangle \wedge\ n \notin \langle Q//*\rangle)\} \quad (4)$$

Let $r = C(n)$. When C/C^{-1} is sound according to Definition 2, we have:

$$C(\langle P\rangle \overset{D}{\cup}_X \langle Q\rangle) = \{r|[r \in C(\langle P\rangle) \vee r \in C(\langle Q\rangle)] \wedge [r \notin C(\langle P//*\rangle) \wedge r \notin C(\langle Q//*\rangle)]\}$$

Here, since we are to find $\overset{D}{\cup}_R$ such that $C(\langle P\rangle) \overset{D}{\cup}_R C(\langle Q\rangle) = C(\langle P\rangle \overset{D}{\cup}_X \langle Q\rangle)$:

$$C(\langle P\rangle) \overset{D}{\cup}_R C(\langle Q\rangle) = \{r|[r \in C(\langle P\rangle) \vee r \in C(\langle Q\rangle)] \wedge [r \notin C(\langle P//*\rangle) \wedge r \notin C(\langle Q//*\rangle)]\}$$

The condition of $[r \in C(\langle P\rangle) \vee\ r \in C(\langle Q\rangle)]$ is essentially the regular union. It is composed by set containment and Boolean operations. In XRDB, set containment check is supported when the soundness requirement in Definition 2 is fulfilled, and Boolean operation is generally supported in RDBMS. $[r \notin C(\langle P//*\rangle) \wedge r \notin C(\langle Q//*\rangle)]$ tends to support deep semantics. It requires XRDB to be able to identify if $r \in C(\langle P//*\rangle)$ for any given relational object r and set $C(\langle P\rangle)$.

Lemma 2. *To implement deep-union operator in XRDB(C), the X2R conversion algorithm C should: (1) fulfil the soundness requirement stated in Definition 2; and (2) for given node n and node set* $\langle P\rangle$, *it should be able to check the containment condition of:* $C(n) \in C(\langle P//*\rangle)$, *e.g., it should recognize if* $C(n)$ *is a descendant of any node* $C(p_i)$*;* ∎

At present, all X2R conversion algorithms (we are aware of) fulfill Lemma 2.
Deep-intersect operator is used to calculate the exact overlapping of queried data and accessible data (i.e. $\langle Q \rangle$ and $\langle ACR \rangle$). It is defined as:

$$C(\langle P \rangle \overset{D}{\cap}_X \langle Q \rangle) = \{r | [r \in C(\langle P \rangle) \wedge r \in C(\langle Q \rangle)] \wedge [r \notin C(\langle P//* \rangle) \vee r \notin C(\langle Q//* \rangle)]\} \tag{5}$$

Compare with 4, they only differ in logical operators. Therefore, Lemma 2 could be directly extended to deep-intersect.

Example 2. In Example 1, a query "`//people`" yields `<people>` nodes, i.e. element 252, (record 1 in Fig. 2 (b)). Meanwhile, *object* field of access control rule 1, "`/site/people/person`", yields `<person>` nodes, i.e. element 293 and 299 (record 2 and 3 in Fig. 2 (b). In XRel, each XML node is marked with "*start*" and "*end*" offset. Node containment is checked through comparison of the offsets: for nodes p_1 and p_2, if ($p_1.start < p_2.start$) and ($p_1.end > p_2.end$), p_2 is an descendant of p_1. Here, we can tell that node 293 and 299 are descendants of node 292. Therefore, "`//people` $\overset{D}{\cap}_X$ `//person`" yields node 293 and 299. Comparing with Example 1, "`//people` $\cap$ `//person`" yields $Null$. □

The operands of XML deep-union/intersect operators may contain different nodes. In RDBMS, where domain compatibility is strictly enforced, their relational equivalent counterpart might be domain incompatible (e.g. a row "intersect" a cell). This happens in schema-based X2R conversion (e.g. [15, 50]), where different XML nodes could be converted to tables, rows, etc. To tackle this problem, we employ new RDBMS techniques, e.g. Oracle VPD, to enable fine-control of relational tables to create relational views with any group of cells.

Deep-except is used to remove inaccessible nodes from the answer. Recall that, in our XML access control model, all nodes are inaccessible by default. When a user is prohibited to access a node, there is no need to write a negative rule to revoke accessibility unless the node is covered by positive rules (ACR^+). Thus, negative rules are only used to specify exceptions to global permissions, i.e. "revoke" access granted by ACR^+. Deep except operator is used to enforce negative rules. Regarding whether deep except could be implemented in XRDB with X2R conversion algorithm, it depends upon the characteristics of specific negative rule. In particular, we distinguish two types of negative rules:

Definition 7 (Node elimination vs. Descendant elimination rules) *A negative rule in ACR restricts user from access a set of nodes $\{r_1^-, ...r_n^-\}$. If* ***none*** *of the nodes is a descendant of the context node of a positive rule, i.e.:*

$$r_i^- \notin \langle R^+//* \rangle, \quad \forall r_i^- \in \{r_1^-, ...r_n^-\}; \forall \langle R^+ \rangle \in \langle ACR^+ \rangle$$

then it is called a ***node elimination*** *(**NE**) negative rule. Else, if one of the nodes is a descendant of the context node of a positive rule, i.e.:*

$$r_i^- \in \langle R^+//* \rangle, \quad \exists r_i^- \in \{r_1^-, ...r_n^-\}; \exists \langle R^+ \rangle \in \langle ACR^+ \rangle$$

it is called a ***descendant elimination*** *(**DE**) negative rule.* □

Intuitively, "*Node Elimination*" negative rule removes context node from $\langle ACR^+ \rangle$. For XML nodes covered by node elimination negative rules $\langle ACR_1^- \rangle$, deep-except operator directly removes them from $\langle ACR^+ \rangle$:

$$\langle ACR^+ \rangle \overset{D}{-}_X \langle ACR_1^- \rangle = \{n | n \in \langle ACR^+ \rangle \ \wedge \ n \notin \langle ACR_1^- \rangle\}$$

Essentially, this is the regular except semantics. In this way, in XRDB, we have,

$$C(\langle ACR^+ \rangle) \overset{D}{-}_R C(\langle ACR_1^- \rangle) = \{r | r \in C(\langle ACR^+ \rangle) \ \wedge \ r \notin C(\langle ACR_1^- \rangle)\}$$

To support deep except operator for node elimination negative rules only, the conditions described in Lemma 2 still apply.

On the other hand, "*Descendant Elimination*" negative rule removes descendants from context node of $\langle ACR^+ \rangle$. It takes more burden to process descendant elimination negative rules, where real "deep" semantics is required. That is,

$$\langle ACR^+ \rangle \overset{D}{-}_X \langle ACR_2^- \rangle = \{deepRemove(n, n \overset{D}{\cap}_X \langle ACR_2^- \rangle) | n \in \langle ACR^+ \rangle\}$$

where $deepRemove(p, \langle Q \rangle)$ takes a node and a set of its descendants as operands, removes the descendants from the subtree of the node and return the remaining. This function may not be directly converted to relational.

Lemma 3. *When deep-except operator takes node specified by descendant elimination negative rules as the second operand, it is implemented through deepRemove() operation. To implement deep-except operator that supports descendant elimination negative rules in XRDB(C), the X2R conversion algorithm X should: (1) fully satisfy Lemma 2; and (2) for any node n_1 and its descendant n_2, $C(n_2)$ should be part of $C(n_1)$; and in the reverse conversion of $n_1 = C^{-1}(C(n_1))$, node n_2 in the subtree is entirely converted from $C(n_2)$.* ∎

Example 3. For instance, in Example 1, Rule 2 is a descendant elimination negative rule since it revoke access towards descendants of node (`<person>`).
In XRDB(XRel) [56], descendants are converted to independent records that are stand alone from ancestors. As shown in Fig. 2(b), to reconstruct a `<person>` node, $C_{XRel}^{-1}()$ only takes record 2 to reconstruct a full node. The descendant `<creditcard>` is included in the answer, but the record 4 is not touched by $C_{XRel}^{-1}()$. Therefore, XRel violates condition (2) of Lemma 3, so that we cannot directly implement deep-except operator to support descendant elimination rules. When user requests for "`//person`", we are not able to revoke access towards `<creditcard>` child from RDBMS side.
In Shared-Inlining [50] approach, `<person>` nodes are translated into a table, and `<creditcard>` nodes take a column. The relational schema is [32]:

```
Person(Id, ParentId, Person, Person_address, Person_address_city, ...
..., Person_address_zipcode, Person_creditcard, ......)
```

Here, the ancestor-descendant relationship is kept such that each row represents a "`person`" node, and each cell represents a child node. Therefore, to obtain `//person` $\overset{D}{-}_X$ `//creditcard`, we just mask "`person_creditcard`" column in the table; and the reconstructed XML tree of "`person`" node will not have corresponding child, i.e., "`creditcard`" node is removed from the XML answer. □

5 XML Access Control Enforcement in XRDB

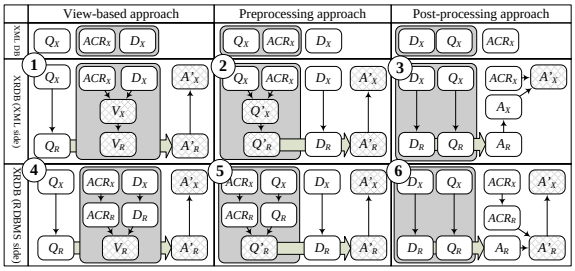


Fig. 3. Access control enforcement approaches in XML DB and XRDB.

Figures 3 shows a general framework for XML access control in XDB and XRDB. There are three categories of XML access control enforcement mechanisms: (1) view-based approach (① ④ in Fig. 3); (2) pre-processing approach (② ⑤ in Fig. 3); and (3) post-processing approach (③ ⑥ in Fig. 3). In this section, we articulate the algebra of these approaches using deep set operators.

5.1 View-based approach

When access control is first enforced on XML documents to create *views*, it is the traditional view-based approach. In this model, XML view V_X (or *safe document* SD) is constructed, and query is evaluated against the view

$$SA = Q\langle V_X\rangle = Q[(\langle R^+_{X1}\rangle \overset{D}{\cup}_X ... \overset{D}{\cup}_X \langle R^+_n\rangle) \overset{D}{-}_X (\langle R^-_1\rangle \overset{D}{\cup}_X ... \overset{D}{\cup}_X \langle R^-_m\rangle)]$$

To convert this approach into XRDB, we can either convert XML view into relational view, as shown in ① of Figure 3; or construct relational view using converted relational ACR, ④ of Figure 3. They are formalized as:

$$SA = C^{-1}(Q_R\langle V_R\rangle) = C^{-1}(Q_R\langle C(V_X)\rangle) \tag{6}$$

and

$$SA = C^{-1}(Q\langle V_R\rangle) = C^{-1}(Q\langle C(ACR_X)\langle D_R\rangle\rangle) \tag{7}$$

5.2 Pre-processing approach

In preprocessing model, *safe query* SQ is constructed as:

$$SQ_X = Q_X \overset{D}{\cap}_X [(R^+_{X1} \overset{D}{\cup}_X ... \overset{D}{\cup}_X R^+_{Xn}) \overset{D}{-}_X (R^-_{X1} \overset{D}{\cup}_X ... \overset{D}{\cup}_X R^-_{Xm})]$$

Safe answer is yielded by evaluating safe query against the original document: $SA_X = SQ_X\langle D_X\rangle$. To extend this approach to XRDB, we have: (1) XML Query

Rewriting: as shown in ② in Fig. 3, we convert the safe XML query into SQL, and answer it with regular XRDB; and (2) Relational Query Rewriting: as shown in ⑤ in Fig. 3, we convert original Q_X into SQL Q_R. and then we rewrite it into safe query SQ_R. They are formalized as

$$SA_X = C^{-1}(SQ_R\langle D_R\rangle) = C^{-1}(C(SQ_X)\langle D_R\rangle) \tag{8}$$

and $$SQ_R = Q_R \overset{D}{\cap}_R [(\ R^+_{R1}\ \overset{D}{\cup}_R ... \overset{D}{\cup}_R\ R^+_{Rn})\ \overset{D}{-}_X\ (R^-_{R1} \overset{D}{\cup}_X ... \overset{D}{\cup}_X R^-_{Rm})] \tag{9}$$

Example 4. Let use revisit the previous examples: we manage XMark document in XRDB(XRel). Suppose we have access control rule `(user, //people, read, +)`, and user submits query `//name`. Figure 4(a) shows the relational query for $C(//people) \overset{D}{\cap}_R C(//name)$, which is implemented according to the definition in Equation 5 (we marked up all the sub-queries). Moreover, this query could be further optimized, as shown in Figure 4(b). □

```
SELECT   docID, pathid, elementID, st, ed FROM element
WHERE  (elementID IN                                          r∈C(//people::self-or-descendant())
          ( SELECT e0.elementID FROM document d, element e0, pth p0
            WHERE   p0.pathexp LIKE '#%/people%'  AND e0.pathid = p0.pathid AND d.docid = e0.docid )
        AND elementID IN                                      r∈C(//name::self-or-descendant())
          (SELECT e0.elementID FROM document d, element e0, pth p0
           WHERE p0.pathexp LIKE '#%/name%' AND e0.pathid = p0.pathid AND d.docid = e0.docid ))
        AND ( NOT elementID IN                                r∈C(//people//*)
          (SELECT e0.elementID FROM document d, element e0, pth p0
           WHERE p0.pathexp LIKE '#%/people#/%' AND e0.pathid = p0.pathid AND d.docid = e0.docid)
        OR NOT elementID IN                                   r∈C(//name//*)
          (SELECT e0.elementID FROM document d, element e0, pth p0
           WHERE p0.pathexp LIKE '#%/name#/%' AND e0.pathid = p0.pathid AND d.docid = e0.docid))
```

(a) SQL query for: //people $\overset{D}{\cap}$ //name

```
SELECT   e0.docID, e0.elementID, e0.st, e0.ed FROM demo.document d, demo.element e0, demo.pth p0
WHERE  (( p0.pathexp LIKE '#%/people%'  AND e0.pathid = p0.pathid AND d.docid = e0.docid)
AND      (p0.pathexp LIKE '#%/name%' AND e0.pathid = p0.pathid AND d.docid = e0.docid ))
AND      ((NOT p0.pathexp LIKE '#%/people#/%' AND e0.pathid = p0.pathid AND d.docid = e0.docid)
OR       (NOT p0.pathexp LIKE '#%/name#/%' AND e0.pathid = p0.pathid AND d.docid = e0.docid))
```

(b) optimized SQL query

Fig. 4. Enforcing XML access control via external pre-processing

Another method is to use Oracle VPD. Oracle version 8.1.5 introduces a new security feature supporting non-view-based fine-grained access control, namely *Row Level Security* or *Virtual Private Database*. It allows users to control accessibility towards row/cell level. With VPD, we are able to tailor relational data into any shape we want. To utilize VPD for access control in XRDB, we first construct relational predicates from the converted relational access control rules ACR_R, then define a VPD policy to enforce the predicates on converted SQL queries. Moreover, cell level access control capability of VPD is of special importance to XRDB systems that use schema-based X2R conversion algorithm, such as Inlining. In those XRDB systems, XML nodes are converted to different types of relational objects: tables, rows and cells. In this way, $\langle ACR\rangle$ may not be conventional relations, e.g. it could be arbitrary combinations of columns, rows and/or individual cells.

5.3 Post-processing based approach

In native XML DB, access control through post-processing described as:

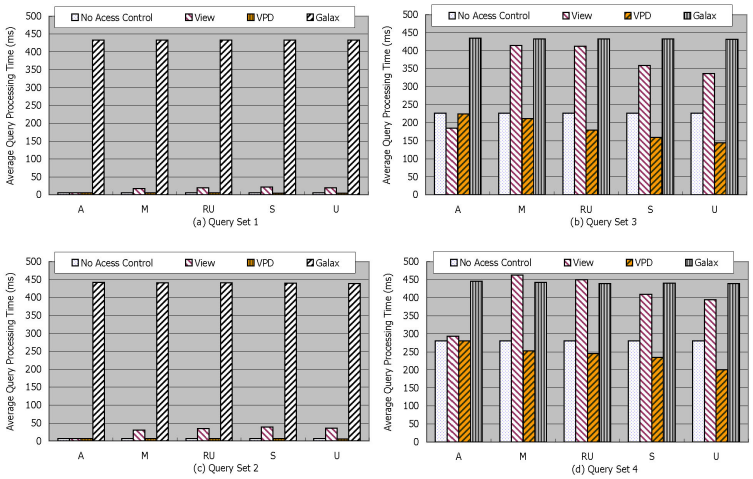


Fig. 5. Query processing time for four sets of queries.

$$SA_X = ACR\langle A_X \rangle = ACR\langle Q_X \langle D_X \rangle \rangle$$

In XRDB, this approach could be conducted through: (1) XML answer filtering (③ in Fig. 3); or (2) relational answer filtering (⑥ in Fig. 3). (1) is similar to the postprocessing approach in [33], while (2) evaluates relational query Q_R to obtain unsafe relational answer, and process ACR_R against the answers:

$$SA_X = C^{-1}(SA_R) = C^{-1}(ACR_R\langle A_R \rangle) = C^{-1}(ACR_R\langle Q_R \langle D_R \rangle \rangle)$$

However, the post-processing filters require the intermediate answers ($\langle A_R \rangle$ or $\langle A_X \rangle$) to retain information of the original paths for ACR to operate on. As an example of this approach, [8] check streaming XML data against both query and ACR at the same time. Since it works in the streaming data environment, full paths are retained. However, in most X2R conversion algorithms, the intermediate answer A_R or A_X does not contain full path information. Therefore, postprocessing approaches are not suitable for all applications.

6 Experimental Validation

To show that the proposed theory and implementations are practical yet efficient, we show our preliminary experimental results.

An XML document with 8517 nodes are generated by XMark [49]. We use XRDB(XRel) [56], with Oracle 10g as underlying RDBMS. We design five roles, abbreviated as A (administrator), M, RU, S and U, respectively. We do not have any descendant elimination negative rules since XRDB(XRel) cannot directly handle it (Lemma 3. We generate four groups of synthetic XPath queries, each has a different setting of wildcards and predicates.

In the XRDB(XRel), we convert all rules into relational, and enforce them through views and VPD. For a comparison, we also enforce same rule sets on the same XML document in native XML environment. We enforce XML access control rules using QFilter [34], and answer XML queries using Galax. In all the

experiments, we use the *query processing time* as an evaluation metric. Figure 5 shows the results. Comparing both view-based and VPD-based approaches with the reference (no security enforcement), our approaches do not add much overhead for fine-grained access control. Meanwhile, the size of accessible data gets smaller with security enforcement. Therefore, querying on smaller set of records is even faster than that on no-security case.

7 Conclusion

In this paper, we propose a generic analysis to the access control problem in XRDB. We first analyze XML control models to propose a formal description of XML access control using deep set operators. Then we articulate the problem of XML access control in XRDB as essentially the problem of XML/Relational object and operation equivalency and conversion. We show that, equivalent counterparts of deep set operators in relational model are needed to fully implement XML access control in XRDB. We analyze the definition and semantics of each operator, and show how they can be converted to XRDB through two lemmas. Although detailed conversion implementation is connected with the specific X2R conversion algorithm used in XRDB, we propose an algebraic description of these operators. Moreover, we study possible implementations of XML access control in XRDB. We categorize them into three approaches, and formally describe the semantics of each approach using deep set operators. Finally, we show the validity of our approaches using experiment results.

References

1. D. Barbosa, J. Freire, and A. O. Mendelzon. "Designing Information-preserving Mapping Schemes for XML". In *VLDB*, pages 109–120, Trondheim, Norway, 2005.
2. A. Berglund, S. Boag, D. Chamberlin, M. F. Fernndez, M. Kay, J. Robie, and J. Simeon. "XML Path Language (XPath) 2.0". W3C Working Draft, Nov. 2003.
3. E. Bertino and E. Ferrari. "Secure and Selective Dissemination of XML Documents". *ACM TISSEC*, 5(3):290–331, Aug. 2002.
4. E. Bertino, S. Castano, and E. Ferrari. Securing xml documents with author-x. *IEEE Internet Computing*, 5(3):21–31, 2001.
5. E. Bertino, E. Ferrari, and L. P. Provenza. Signature and access control policies for xml documents. *ESORICS*, pages 1–22. Springer, 2003.
6. K. Beyer, F. Ozcan, S. Saiprasad, and B. V. der Linden. DB2/XML: designing for evolution. In *ACM SIGMOD*, pages 948–952, 2005.
7. S. Boag, D. Chamberlin, M. F. Fernandez, D. Florescu, J. Robie, and J. Simeon. "XQuery 1.0: An XML Query Language". W3C Working Draft, Nov. 2003.
8. L. Bouganim, F. D. Ngoc, and P. Pucheral. "Client-Based Access Control Management for XML Documents". In *VLDB*, Toronto, Canada, 2004.
9. T. Bray, J. Paoli, and C. M. Sperberg-McQueen (Eds). "Extensible Markup Language (XML) 1.0 (2nd Ed.)". W3C Recommendation, Oct. 2000.
10. Barbara Carminati, Elena Ferrari, and Elisa Bertino. Securing xml data in third-party distribution systems. In *ACM CIKM*, pages 99–106, 2005.
11. S. Cho, S. Amer-Yahia, L. V.S. Lakshmanan, and D. Srivastava. "Optimizing the Secure Evaluation of Twig Queries". In *VLDB*, Hong Kong, China, Aug. 2002.

12. E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. "A Fine-Grained Access Control System for XML Documents". *ACM TISSEC*, 5(2):169–202, May 2002.
13. 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.
14. E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Securing xml documents. In *EDBT*, pages 121–135. Springer, 2000.
15. A. Deutsch, M. F. Fernandez, and D. Suciu. "Storing Semistructured Data with STORED". In *ACM SIGMOD*, Philadephia, PA, Jun. 1998.
16. Wenfei Fan, Chee-Yong Chan, and Minos Garofalakis. Secure xml querying with security views. In *ACM SIGMOD*,Paris, France, 2004.
17. E. Fernandez, E. Gudes, and H. Song. "A Model of Evaluation and Administration of Security in Object-Oriented Databases". *IEEE TKDE*, 6(2):275–292, 1994.
18. B Finance, S Medjdoub, and P Pucheral. The case for access control on xml relationships. In *ACM CIKM*, pages 107–114, Bremen, Germany, 2005.
19. D. Florescu and D. Kossmann. "Storing and Querying XML Data Using an RDBMS". *IEEE Data Eng. Bulletin*, 22(3):27–34, Sep. 1999.
20. I. Fundulaki and M. Marx. Specifying access control policies for xml documents with xpath. In *ACM SACMAT*, pages 61–69, Yorktown Heights, USA, 2004.
21. A. Gabillon and E. Bruno. Regulating access to xml documents. In *DBSec*, pages 299–314, Ontario, Canada, 2001.
22. S. Godik and T. Moses (Eds). "eXtensible Access Control Markup Language (XACML) Version 1.0". OASIS Specification Set, Feb. 2003.
23. P. P. Griffiths and B. W. Wade. "An Authorization Mechanism for a Relational Database System". *ACM TODS*, 1(3):242–255, Sep. 1976.
24. S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. "Flexible Support for Multiple Access Control Policies". *ACM TODS*, 26(2):214–260, 2001.
25. S. Jajodia, P. Samarati, V.Subrahmanian, and E. Bertino. "A Unified Framework for Enforcing Multiple Access Control Policies". In *ACM SIGMOD*, 1997.
26. S. Jajodia and R. Sandhu. "Toward a Multilevel Secure Relational Data Model". In *ACM SIGMOD*, May 1990.
27. Mingfei Jiang and Ada Wai-Chee Fu. Integration and efficient lookup of compressed xml accessibility maps. *IEEE TKDE*, 17(7):939–953, 2005.
28. M. Kudo and S. Hada. "XML Document Security Based on Provisional Authorization". In *ACM Conf. on Computer and Communications Security (CCS)*, 2000.
29. G. Kuper, F. Massacci, and N. Rassadko. Generalized xml security views. In *SACMAT*, Stockholm, Sweden, 2005.
30. D. Lee and W. W. Chu. "Constraints-preserving Transformation from XML Document Type Definition to Relational Schema". In *ER*, Salt Lake City, 2000.
31. D. Lee, W.-C. Lee, and P. Liu. "Supporting XML Security Models using Relational Databases: A Vision". In *XML Symp. (XSym)*, Berlin, Germany, 2003.
32. H. Lu et al. What makes the differences: benchmarking xml database implementations. *ACM Trans. on Internet Technology (TOIT)*, 5(1):154–194, 2005.
33. B. Luo, D. Lee, W. Lee, and P. Liu. "A Flexible Framework for Architecting XML Access Control Enforcement Mechanisms". In *VLDB Workshop on SDM*, 2004.
34. B. Luo, D. Lee, W. Lee, and P. Liu. "QFilter: Fine-Grained Run-Time XML Access Control via NFA-based Query Rewriting". In *ACM CIKM*, 2004.
35. B. Luo, D. Lee, W. Lee, and P. Liu. Deep set operators for xquery. In *ACM SIGMOD Workshop on XIME-P*, Baltimore, MD, USA., 2005.

36. Bo Luo, Dongwon Lee, and Peng Liu. Pragmatic xml access control using off-the-shelf rdbms. Technical report, Penn State University, 2007.
37. S. Mohan, A. Sengupta, and Y. Wu. Access control for xml: a dynamic query rewriting approach. In *ACM CIKM*, pages 251–252, Bremen, Germany 2005.
38. S. Mohan and Y. Wu. Ipac: an interactive approach to access control for semi-structured data. In *VLDB*, pages 1147–1150. 2006.
39. M. Murata, A. Tozawa, and M. Kudo. "XML Access Control using Static Analysis". In *ACM CCS*, Washington D.C., 2003.
40. Makoto Murata, Akihiko Tozawa, Michiharu Kudo, and Satoshi Hada. Xml access control using static analysis. *ACM Trans. Inf. Syst. Secur.*, 9(3):292–324, 2006.
41. R. Murthy, Z. Liu, M. Krishnaprasad, S. Chandrasekar, A. Tran, E. Sedlar, D. Florescu, S. Kotsovolos, N. Agarwal, V. Arora, and V. Krishnamurthy. Towards an enterprise XML architecture. In *ACM SIGMOD*, Baltimore, 2005.
42. Matthias Nicola and Bert van der Linden. Native XML support in DB2 universal database. In *VLDB*, 2005.
43. N. Qi and M. Kudo. Access-condition-table-driven access control for xml databases. In *ESORICS*, 2004.
44. N. Qi and M. Kudo. Xml access control with policy matching tree. In *ESORICS*, 2005.
45. N. Qi, M. Kudo, J. Myllymaki, and H. Pirahesh. A function-based access control model for xml databases. In *ACM CIKM*, Bremen, Germany, 2005.
46. Michael Rys. XML and relational database management systems: inside Microsoft SQL Server 2005. In *ACM SIGMOD*, Baltimore, MD, 2005.
47. P. Samarati, E. Bertino, and S. Jajodia. "An Authorization Model for a Distributed Hypertext System". *IEEE TKDE*, 8(4):555–562, 1996.
48. R. Sandhu and F. Chen. "The Multilevel Relational (MLR) Data Model". *ACM TISSEC*, 1(1), 1998.
49. A. Schmidt, F. Waas, M. Kersten, D. Florescu, I. Manolescu, M. Carey, and R. Busse. "The XML Benchmark Project". Technical Report, CWI, 2001.
50. 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.
51. Andrei Stoica and Csilla Farkas. Secure xml views. In *DBSec*, pages 133–146, 2002.
52. K.-L. Tan, M. L. Lee, and Y. Wang. "Access Control of XML Documents in Relational Database Systems". In *IC*, Las Vegas, NV, Jun. 2001.
53. Jingzhu Wang and Sylvia L. Osborn. A role-based approach to access control for xml databases. In *ACM SACMAT*, pages 70–77, Yorktown Heights, USA 2004.
54. M. Winslett, K. Smith, and X. Qian. "Formal Query Languages for Secure Relational Databases". *ACM TODS*, 19(4):626–662, 1994.
55. Yan Xiao, Bo Luo, and Dongwond Lee. "Security-Conscious XML Indexing". In *DASFAA*, Bangkok, Thailand, 2007.
56. 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 TOIT*, 1(2):110–141, Nov. 2001.
57. T. Yu, D. Srivastava, L. Lakshmanan, and H. Jagadish. "Compressed Accessibility Map: Efficient Access Control for XML". In *VLDB*, Hong Kong, 2002.