Storing Multidimensional XML documents in Relational Databases*

Nikolaos Fousteris, Manolis Gergatsoulis, and Yannis Stavrakas

Department of Archive and Library Sciences, Ionian University,
Palea Anaktora, Plateia Eleftherias, 49100 Corfu, Greece.
{nfouster,manolis}@ionio.gr, ys@dblab.ntua.gr

Abstract. The problem of storing and querying XML data using relational databases has been considered a lot and many techniques have been developed. MXML is an extension of XML suitable for representing data that assume different facets, having different value and structure under different contexts, which are determined by assigning values to a number of dimensions. In this paper, we explore techniques for storing MXML documents in relational databases, based on techniques previously proposed for conventional XML documents. Essential characteristics of the proposed techniques are the capabilities a) to reconstruct the original MXML document from its relational representation and b) to express MXML context-aware queries in SQL.

1 Introduction

The problem of storing XML data in relational databases has been intensively investigated [4, 10, 11, 13] during the past 10 years. The objective is to use an RDBMS in order to store and query XML data. First, a relational schema is chosen for storing the XML data, and then XML queries, produced by applications, are translated to SQL for evaluation. After the execution of SQL queries, the results are translated back to XML and returned to the application.

Multidimensional XML (MXML) is an extension of XML which allows context specifiers to qualify element and attribute values, and specify the contexts under which the document components have meaning. MXML is therefore suitable for representing data that assume different facets, having different value or structure, under different contexts. Contexts are specified by giving values to one or more user defined dimensions. In MXML, dimensions may be applied to elements and attributes their values depend on the dimensions). An alternative solution would be to create a different XML document for every possible combination, but such an approach involves excessive duplication of information.

In this paper, we present two approaches for storing MXML in relational databases, based on XML storage approaches. We use MXML-graphs, which

* This research was partially co-funded by the European Social Fund (75%) and National Resources (25%) - Operational Program for Educational and Vocational Training (EPEAEK II) and particularly by the Research Program "PYTHAGORAS II".

are graphs using appropriate types of nodes and edges, to represent MXML documents. In the first (naive) approach, a single relational table is used to store all information about the nodes and edges of the MXML-graph. Although simple, this approach presents some drawbacks, like the large number of expensive self-joins when evaluating queries. In the second approach we use several tables, each of them storing a different type of nodes of the MXML-graph. In this way the size of the tables involved in joins is reduced and consequently the efficiency of query evaluation is enhanced. Both approaches use additional tables to represent context in a way that it can be used and manipulated by SQL queries.

2 Preliminaries

2.1 Mutidimensional XML

In MXML, data assume different facets, having different value or structure, under different contexts according to a number of *dimensions* which may be applied to elements and attributes [7, 8]. The notion of "world" is fundamental in MXML. A world represents an environment under which data obtain a meaning. A *world* is determined by assigning to every dimension a single value, taken from the domain of the dimension. In MXML we use syntactic constructs called *context specifiers* that specify sets of worlds by imposing constraints on the values that dimensions can take. The elements/attributes that have different facets under different contexts are called *multidimensional elements/attributes*). Each multidimensional element/attribute contains one or more facets, called *context elements/attributes*), accompanied with the corresponding context specifier which denotes the set of worlds under which this facet is the holding facet of the element/attribute. The syntax of MXML is shown in Example 1, where a MXML document containing information about a book is presented.

Example 1. The MXML document shown below represents a book in a book store. Two dimensions are used namely `edition` whose domain is {`greek, english`}, and `customer_type` whose domain is {`student, library`}.

```
<book isbn=[edition=english]"0-13-110362-8"[/]
           [edition=greek]"0-13-110370-9"[/]>
  <title>The C programming language</title>
  <authors>
     <author>Brian W. Kernighan</author>
     <author>Dennis M. Ritchie</author>
  </authors>
  <@publisher>
     [edition = english] <publisher>Prentice Hall</publisher>[/]
     [edition = greek] <publisher>Klidarithmos</publisher>[/]
  </@publisher>
  <@translator>
     [edition = greek] <translator>Thomas Moraitis</translator>[/]
  </@translator>
  <@price>
```

```
    [edition=english]<price>15</price>[/]
    [edition=greek,customer_type=student]<price>9</price>[/]
    [edition=greek,customer_type=library]<price>12</price>[/]
  </@price>
  <@cover>
    [edition=english]<cover><material>leather</material></cover>[/]
    [edition=greek]
       <cover>
          <material>paper</material >
          <@picture>
            [customer_type=student]<picture>student.bmp</picture>[/]
            [customer_type=library]<picture>library.bmp</picture>[/]
          </@picture>
        </cover>
     [/]
  </@cover>
</book>
```

Notice that multidimensional elements (see for example the element `price`) are the elements whose name is preceded by the symbol `@` while the corresponding context elements have the same element name but without the symbol `@`.

A MXML document can be considered as a compact representation of a set of (conventional) XML documents, each of them holding under a specific world. In Subsection 3.3 we will present a process called *reduction* which extracts XML documents from a MXML document.

2.2 Storing XML data in relational databases

Many researchers have investigated how an RDBMS can be used to store and query XML data. Work has also been directed towards the storage of temporal extensions of XML [15, 1, 2]. The techniques proposed for XML storage can be divided in two categories, depending on the presence or absence of a schema:

1. *Schema-Based XML Storage techniques*: the objective here is to find a relational schema for storing an XML document, guided by the structure of a schema for that document [9, 13, 5, 14, 10, 3, 11].
2. *Schema-Oblivious XML Storage techniques*: the objective is to find a relational schema for storing XML documents independent of the presence or absence of a schema [13, 5, 14, 16, 10, 6, 4].

The approaches that we propose in this paper do not take schema information into account, and therefore belong to the Schema-Oblivious category.

3 Properties of MXML documents

3.1 A graphical model for MXML

In this section we present a graphical model for MXML called *MXML-graph*. The proposed model is node-based and each node is characterized by a unique "id".

In MXML-graph, except from a special node called *root node*, there are the following node types: *multidimensional element nodes*, *context element nodes*, *multidimensional attribute nodes*, *context attribute nodes*, and *value nodes*. The *context element nodes*, *context attribute nodes*, and *value nodes* correspond to the element nodes, attribute nodes and value nodes in a conventional XML graph. Each multidimensional/context element node is labelled with the corresponding element name, while each multidimensional/context attribute node is labelled with the corresponding attribute name. As in conventional XML, value nodes are leaf nodes and carry the corresponding value. The facets (context element/attribute nodes) of a multidimensional node are connected to that node by edges labelled with context specifiers denoting the conditions under which each facet holds. These edges are called *element / attribute context edges* respectively. Context elements/attributes are connected to their child elements/attribute or value nodes by edges called *element/attribute/value edges* respectively. Finally, the context attributes of type IDREF(S) are connected to the element nodes that they point to by edges called *attribute reference edges*.

Example 2. In Fig. 1, we see the representation of the MXML document of Ex-

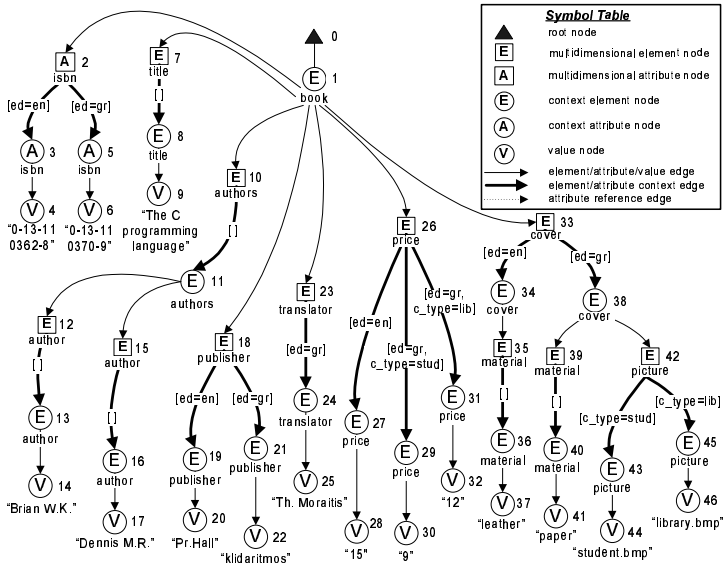


Fig. 1. Graphical representation of MXML (MXML tree)

ample 1 as a MXML-graph. Note that some additional multidimensional nodes (e.g. nodes 7 and 10) have been added to ensure that the types of the edges alternate consistently in every path of the graph. This does not affect the information contained in the document, but facilitates the navigation in the graph and the formulation of queries. For saving space, in Fig. 1 we use obvious abbreviations for dimension names and values that appear in the MXML document.

3.2 Properties of contexts

Context specifiers qualifying element/attribute context edges give the *explicit contexts* of the nodes to which these edges lead. The explicit context of all the other nodes of the MXML-graph is considered to be the *universal context* [ ], denoting the set of all possible worlds. The explicit context can be considered as the true context only within the boundaries of a single multidimensional element/attribute. When elements and attributes are combined to form a MXML document, the explicit context of each element/attribute does not alone determine the worlds under which that element/attribute holds, since when an element/attribute e_2 is part of another element e_1, then e_2 have substance only under the worlds that e_1 has substance. This can be conceived as if the context under which e_1 holds is inherited to e_2. The context propagated in that way is combined with (constraint by) the explicit context of a node to give the *inherited context* for that node. Formally, the inherited context $ic(q)$ of a node q is defined as $ic(q) = ic(p) \cap^c ec(q)$, where $ic(p)$ is the inherited context of its parent node p. $\cap^c$ is an operator called *context intersection* defined in [12] which combines two context specifiers and computes a new context specifier which represents the intersection of the worlds specified by the original context specifiers. The evaluation of the inherited context starts from the root of the MXML-graph. By definition, the inherited context of the root of the graph is the universal context [ ]. Note that contexts are not inherited through attribute reference edges.

As in conventional XML, the leaf nodes of MXML-graphs must be value nodes. The *inherited context coverage* of a node further constraints its inherited context, so as to contain only the worlds under which the node has access to some value node. This property is important for navigation and querying, but also for the reduction process presented in the next section. The inherited context coverage $icc(n)$ of a node n is defined as follows: if n is a leaf node then $icc(n) = ic(n)$; otherwise $icc(n) = icc(n_1) \cup^c icc(n_2) \cup^c ... \cup^c icc(n_k)$, where $n_1, \ldots, n_k$ are the child element nodes of n. $\cup^c$ is an operator called *context union* defined in [12] which combines two context specifiers and computes a new one which represents the union of the worlds specified by the original context specifiers. The inherited context coverage gives the true context of a node in a MXML-graph.

3.3 Reduction of MXML to XML

Reduction is a process that given a world w, and a MXML document (MXML-graph) G, we can obtain a conventional XML document (XML-Graph) G' which is the facet of G under w. Reduction is based on the idea that we should eliminate all subtrees of G for which the world w does not belong to the worlds specified by the inherited context coverage of their roots. Then, we eliminate each element context edge (resp. attribute context edge) (p, C, q) of the graph G_1 obtained from G in this way, as follows: Let (s, p) be the element edge (resp. attribute edge) leading to the node p. Then a) add a new element edge (resp. attribute edge) (s, q), and b) remove the edges (p, C, q) and (s, p) and the node p.

The XML document (XML-graph) G' obtained in this way is the holding facet of the MXML document G under the world w.

4 Storing MXML in relational databases

In this section we present two approaches for storing MXML documents using relational databases.

4.1 Naive Approach

The first approach, called *naive approach*, uses a single table (*Node Table*), to store all information contained in a MXML document. Node Table contains all the information which is necessary to reconstruct the MXML document(graph). Each row of the table represents a MXML node. The attributes of Node Table are: `node_id` stores the id of the node, `parent_id` stores the id of the parent node, `ordinal` stores a number denoting the order of the node among its siblings, `tag` stores the label (tag) of the node or NULL (denoted by "-") if it is a value node, `value` stores the value of the node if it is a value node or NULL otherwise, `type` stores a code denoting the node type (CE for context element, CA for context attribute, ME for multidimensional element, MA for multidimensional attribute, and VN for value node), and `explicit_context` stores the explicit context of the node (as a string). Noted that the explicit context is kept here for completeness, and does not serve any retrieval purposes. In the following we will see how the correspondence of nodes to the worlds under which they hold is encoded.

Example 3. Fig. 2 shows how the MXML Graph of Fig. 1 is stored in the Node Table. Some of the nodes have been omitted, denoted by (...), for brevity.

Node Table						
node_id	parent_id	ordinal	tag	value	type	explicit_context
1	0	1	book	-	CE	-
2	1	1	isbn	-	MA	-
3	2	1	isbn	-	CA	[ed=en]
4	3	1	-	0-13-110362-8	VN	-
5	2	2	isbn	-	CA	[ed=gr]
6	5	1	-	0-13-110370-9	VN	-
7	1	2	title	-	ME	-
8	7	1	title	-	CE	[ ]
9	8	1	-	The C progr. lang.	VN	-
....	....	....	....	....	....	....
43	42	1	picture	-	CE	[c_type=stud]
....	....	....	....	....	....	....

Fig. 2. Storing the MXML-graph of Fig. 1 in a Node Table.

We now explain how context is stored in such a way so as to facilitate the formulation of context-aware queries. We introduce three additional tables, as shown in Fig. 3. The *Possible Worlds Table* which assigns a unique ID (attribute `word_id`)

to each possible combination of dimension values. Each dimension in the MXML document has a corresponding attribute in this table. The *Explicit Context Table* keeps the correspondence of each node with the worlds represented by its explicit context. Finally, the *Inherited Coverage Table* keeps the correspondence of each node with the worlds represented by its inherited context coverage.

Example 4. Fig. 3, depicts (parts of) the Possible Worlds Table, the Explicit Context Table, and the Inherited Coverage Table obtained by encoding the context information appearing in the MXML-graph of Fig. 1. For example,

Possible Worlds Table		
world_id	edition	customer_type
1	gr	stud
2	gr	lib
3	en	stud
4	en	lib

Explicit Context Table	
node_id	world_id
1	1
1	2
1	3
1	4
....	....
5	1
5	2
6	1
6	2
6	3
6	4
....	....

Inherited Coverage Table	
node_id	world_id
1	1
1	2
1	3
1	4
....	....
5	1
5	2
6	1
6	2
....	....

Fig. 3. Mapping MXML nodes to worlds.

the inherited context coverage of the node with `node_id=6` includes the worlds { `(edition, greek)`, `(customer_type, student)`} and {`(edition, greek)`, `(customer_type, library)`}. This is encoded in the Inherited Coverage Table as two rows with `node_id=6` and the world ids 1 and 2. In the Explicit Context Table the same node corresponds to all possible worlds (ids 1, 2, 3 and 4).

Representing in this way the context information of MXML-graphs facilitates the construction of SQL queries referring to context. Moreover, it makes possible the translation of queries expressed in a language called MXPath, which is a multidimensional extension of XPath, into equivalent SQL queries. Encoding both the explicit context and the inherited context coverage as above allows us to construct queries which use both the explicit context and the inherited context coverage of nodes. As an example consider the following query given in natural language: *Find the ISBN of the greek edition of the book with title ''The C Programming Language''*. This query is encoded in SQL as:

```
select N4.value
from Node as N1, Node as N2, Node as N3,..., Node as N7
where N7.type="VN" and N7.value="The C Programming language" and
         N7.parent_id=N6.id and
    N6.type="CE" and N6.tag="title" and N6.parent_id=N5.id and
```

```
N5.type="ME" and N5.tag="title" and N5.parent_id=N1.id and
N1.type="CE" and N1.tag="book" and N1.id=N2.parent_id and
N2.type="MA" and N2.tag="isbn" and N2.id=N3.parent_id and
N3.type="CA" and N3.tag="isbn" and N3.id=N4.parent_id and
N4.type="VN" and N4.id in (select IC1.node_id
  from Inherited_Coverage as IC1, Inherited_Coverage as IC2
  where IC1.world_id=1 and IC2.world_id=2 and IC1.node_id=IC2.node_id)
```

The "where" clause implements the navigation on the tree of Fig. 1, while the nested query implements the constraints related to context, in order to finally return node 6 but not node 4. Note that to make the query more readable we have named the table variables after corresponding node ids, and we have included in the query some conditions, which are redundant as they are deduced from the properties of the MXML graph. Observe that, the "greek edition" context contains both the worlds with ids 1 and 2 according to the Possible Worlds table, which has not been used in the SQL query for brevity. Finally, notice the large number of self-joins which is proportional to the depth of the navigation path.

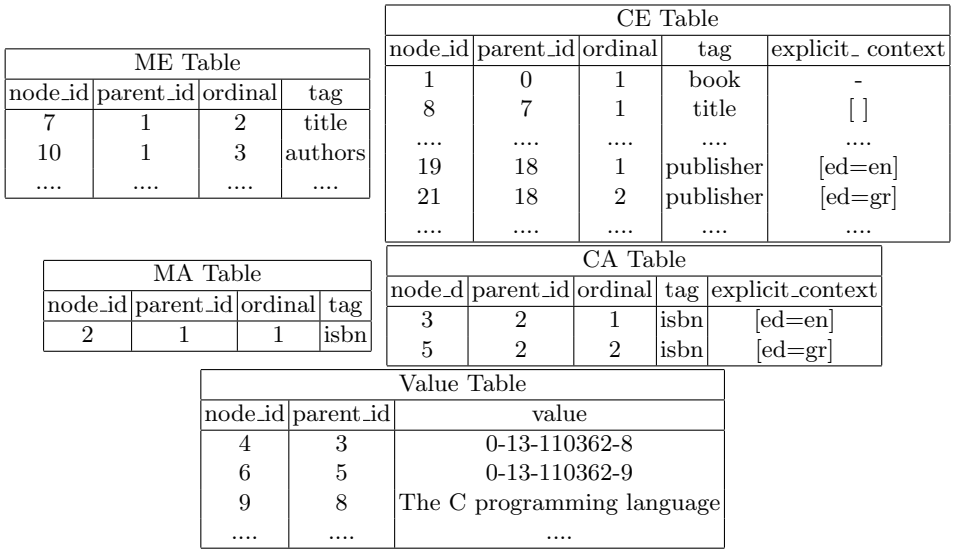
4.2 Limitations of the Naive Approach

The naive approach is straightforward, but it has some drawbacks mainly because of the use of a single table. As the different types of nodes are stored in the table, many NULL values appear in the fields `explicit_context`, `tag`, and `value`. Those NULL values could be avoided if we used different tables for different node types. Moreover, as we showed in Subsection 4.1, queries on MXML documents involve a large number of self-joins of the Node Table, which is anticipated to be a very long table since it contains the whole tree. Splitting the Node Table would reduce the size of the tables involved in joins, and enhance the overall performance of queries. Finally, notice that the context representation scheme we introduced leads to a number of joins in the nested query. Probably a better scheme could be introduced that reduces the number of joins.

4.3 A Better Approach

In the *Type Approach* presented here, MXML nodes are divided into groups according to their type. Each group is stored in a separate table named after the type of the nodes. In particular *ME Table* stores multidimensional element nodes, *CE Table* stores context element nodes, *MA Table* stores multidimensional attribute nodes, *CA Table* stores context attribute nodes, and *Value Table* stores value nodes. The schema of these tables is shown in Fig. 4. Each row in these tables represents a MXML node. The attributes in the tables have the same meaning as the respective attributes of the Node Table. Using this approach we tackle some of the problems identified in the previous section. Namely, we eliminate NULL values and irrelevant attributes, while at the same time we reduce the size of the tables involved in joins when navigating the MXML-Graph.

To represent context, we propose a scheme that reduces the size of tables and the number of joins in context-driven queries. First, we use one table for each



ME Table

node_id	parent_id	ordinal	tag
7	1	2	title
10	1	3	authors
....	....	....	....

CE Table

node_id	parent_id	ordinal	tag	explicit_ context
1	0	1	book	-
8	7	1	title	[ ]
....	....	....	....	....
19	18	1	publisher	[ed=en]
21	18	2	publisher	[ed=gr]
....	....	....	....	....

MA Table

node_id	parent_id	ordinal	tag
2	1	1	isbn

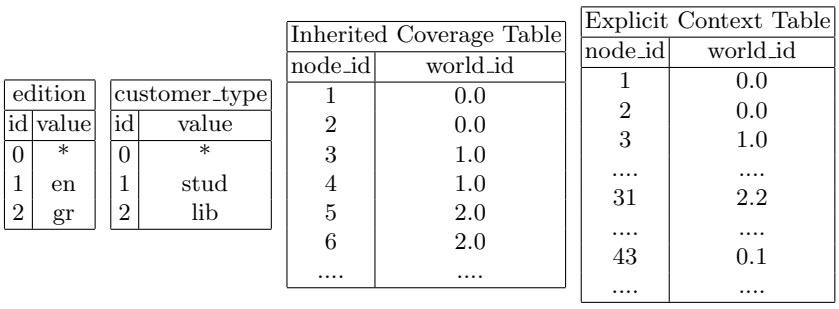
CA Table

node_d	parent_id	ordinal	tag	explicit_context
3	2	1	isbn	[ed=en]
5	2	2	isbn	[ed=gr]

Value Table

node_id	parent_id	value
4	3	0-13-110362-8
6	5	0-13-110362-9
9	8	The C programming language
....	....	....

Fig. 4. The Type tables.



edition

id	value
0	*
1	en
2	gr

customer_type

id	value
0	*
1	stud
2	lib

Inherited Coverage Table

node_id	world_id
1	0.0
2	0.0
3	1.0
4	1.0
5	2.0
6	2.0
....	....

Explicit Context Table

node_id	world_id
1	0.0
2	0.0
3	1.0
....	....
31	2.2
....	....
43	0.1
....	....

Fig. 5. Dimension Tables.

dimension (in our example `edition` and `customer_type`) to assign an id (`id` column) to each possible value (`value` column). Additionally, id "0" represents all possible values of the dimension (for id = 0 we use the value "*"). Then, we assume a fixed order of the dimension names, which will eventually be taken into account in the formulation of queries. Finally, in the Inherited Coverage and Explicit Context tables we use world ids of the form "$a_1.a_2 \ldots a_n$", where a_1, a_2, ..., a_n are ids of dimension values (Fig. 5). For example, the inherited context coverage of the node with id 6 in Fig. 1 is encoded as "2.0" in Fig. 5.

5 Discussion and motivation for future work

Two techniques to store MXML documents in relational databases are presented in this paper. The first one is straightforward and uses a single table to store

MXML. The second divides MXML information according to node types in the MXML-graph and, although it is more complex than the first one, it performs better during querying. We are currently working on an extension of XPath for MXML and its translation to SQL. Our plans for future work include the investigation of techniques to update MXML data stored in relational databases.

References

1. T. Amagasa, M. Yoshikawa, and S. Uemura. A Data Model for Temporal XML Documents. In *Proc. of DEXA 2000*, pages 334–344. Springer, 2000.
2. T. Amagasa, M. Yoshikawa, and S. Uemura. Realizing Temporal XML Repositories using Temporal Relational Databases. In *Proc. of the 3rd Int. Symp. on Cooperative Database Systems and Applications, Beijing, China*, pages 63–68, 2001.
3. P. Bohannon, J. Freire, P. Roy, and J. Simon. From XML Schema to Relations: A Cost-Based Approach to XML Storage. In *Proc. of ICDE 2002*.
4. A. Deutsch, M. F. Fernandez, and D. Suciu. Storing Semistructured Data with STORED. In *Proc. of ACM SIGMOD Int. Conf. on Management of Data*, pages 431–442. ACM Press, 1999.
5. F. Du, S. Amer-Yahia, and J. Freire. ShreX: Managing XML Documents in Relational Databases. In *Proc. of VLDB' 04*, pages 1297–1300. Morgan Kaufmann.
6. D. Florescu and D. Kossmann. Storing and Querying XML Data using an RDBMS. *Bulletin of the IEEE Comp. Soc. Tech. Com. on Data Eng.*, 22(3):27–34, 1999.
7. M. Gergatsoulis, Y. Stavrakas, and D. Karteris. Incorporating Dimensions to XML and DTD. In Proc. of *DEXA' 01*, LNCS Vol. 2113, pages 646–656. Springer, 2001.
8. M. Gergatsoulis, Y. Stavrakas, D. Karteris, A. Mouzaki, and D. Sterpis. A Web-based System for Handling Multidimensional Information through MXML. In Proc. of *ADBIS' 01*, LNCS, Vol. 2151, pages 352–365. Springer-Verlag, 2001.
9. M. Ramanath, J. Freire, J. R. Haritsa, and P. Roy. Searching for Efficient XML-to-Relational Mappings. In Proc. of *XSym 2003*, pages 19–36. Springer, 2003.
10. J. Shanmugasundaram, E. J. Shekita, J. Kiernan, R. Krishnamurthy, S. Viglas, J. F. Naughton, and I. Tatarinov. A General Technique for Querying XML Documents using a Relational Database System. *SIGMOD Record*, 30(3):20–26, 2001.
11. J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In Proc. of *VLDB'99*, pages 302–314. Morgan Kaufmann, 1999.
12. Y. Stavrakas and M. Gergatsoulis. Multidimensional Semistructured Data: Representing Context-Dependent Information on the Web. In Proc. of *CAiSE 2002*, LNCS Vol. 2348, pages 183–199. Springer, 2002.
13. I. Tatarinov, S. Viglas, K. S. Beyer, J. Shanmugasundaram, E. J. Shekita, and C. Zhang. Storing and querying ordered XML using a relational database system. In *Proc. of the 2002 ACM SIGMOD Int. Conf. on Management of Data*, pages 204–215. ACM, 2002.
14. F. Tian, D. J. DeWitt, J. Chen, and C. Zhang. The Design and Performance Evaluation of Alternative XML Storage Strategies. *SIGMOD Record*, 31(1):5–10, 2002.
15. F. Wang, X. Zhou, and C. Zaniolo. Using XML to Build Efficient Transaction-Time Temporal Database Systems on Relational Databases. In *Proc. of ICDE 2006*.
16. 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 Transactions on Internet Technology*, 1(1):110–141, 2001.