ISBN: 978-972-8924-44-7 © 2007 IADIS

STORING-UPDATING AND QUERYING MULTIDIMENSIONAL XML DOCUMENTS USING RELATIONAL DATABASES[1]

Nikolaos Fousteris, Yannis Stavrakas, Manolis Gergatsoulis
Department of Archive and Library Sciences, Ionian University
Palea Anaktora, Plateia Eleftherias, 49100 Corfu, Greece
nfouster@ionio.gr, ys@dblab.ntua.gr, manolis@ionio.gr

ABSTRACT

In Web applications it is often required to manipulate information of semistructured nature, which may present variations according to different circumstances. Multidimensional XML (MXML) is an extension of XML suitable for representing data that assume different facets, having different value and structure, under different *contexts*. Following previous work on storing XML in relational databases, we present results of our ongoing research on storing, querying and updating MXML documents using relational databases. First we consider alternative schemas for representing MXML in RDBs. Then we introduce MXPath, an extension of XPath for MXML, and give an intuition of how MXPath queries can be translated into SQL queries. Finally we consider the problem of updating MXML by introducing a number of basic update operations.

KEYWORDS

Semistructured databases, XML, Multidimensional data.

1. INTRODUCTION

In recent WWW applications, it is often necessary to manipulate a huge amount of semistructured data, often represented in XML format. Moreover, in many occasions, there is need to handle different variants of the same information entity, depending on parameters such as the background and situation of the user, or the capabilities of the device presenting the information. According to Gergatsoulis et al. (2001a, b), Multidimensional XML is an extension of XML suitable for representing and exchanging information presenting different variants (or facets), having different value or structure, under different contexts. Contexts are specified by giving values to one or more user defined *dimensions*.

In this paper we investigate the problem of using relational databases to store, update and query MXML documents. 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. Then the results obtained by evaluating SQL queries, are translated back to XML and returned to the application.

In particular in this paper:

a) We present two approaches for storing MXML in relational databases. To develop our results, we are based on and extend techniques previously proposed for storing and manipulating (conventional) XML documents. Such techniques have been intensively investigated during the past 10 years (Deutsch et al. 1999; Shanmugasundaram et al. 2001; Shanmugasundaram et al. 1999; Tatarinov et al. 2002). The difference between our techniques and the techniques proposed for storing (conventional) XML stems from the need to represent and manipulate contexts, plus the need for representing additional structures that exist in MXML but not in XML.

[1] 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".

b) We outline MXpath, an extension of Xpath suitable for navigating and querying MXML documents. MXPath takes context into account in order to specify navigation patterns in MXML.

c) We briefly present a set of basic change operations, suitable for updating MXML documents. The problem of updating XML documents stored in relational databases has also be given considerable attention (Braganholo et al. 2004, 2003a, b).

The structure the rest of this paper is as follows: Section 2 presents the syntax and the main properties of MXML. Section 3 presents two storage techniques for MXML documents. Section 4 outlines the MXpath. Section 5 presents the proposed update operations. Finally, Section 6 concludes the paper.

2. MULTIDIMENSIONAL XML

In this section we present the syntax of MXML and discuss some of its basic properties.

2.1 The Syntax of MXML

The notion of *world* is fundamental in MXML and represents an environment under which data obtain a meaning. A world is determined by assigning values to a set of user defined *dimensions* such that to every dimension, a single value, taken from its domain, is assigned. 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* and are preceded by the symbol @. The facets are called *context elements/attributes*, and are preceded by context specifiers denoting the sets of worlds under which each facet holds. The MXML document shown in Fig. 1 represents a book in a bookstore. 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>
```

Figure 1. An MXML document representing a book in a bookstore.

2.2 A Graphical Model for MXML

We have developed a graphical, node-based model for MXML, called *MXML-graph*. The MXML-graph representing the MXML document in Fig. 1 is shown in Fig. 2. The different types of nodes and edges used in the graph are shown in the symbol table of Fig. 2.

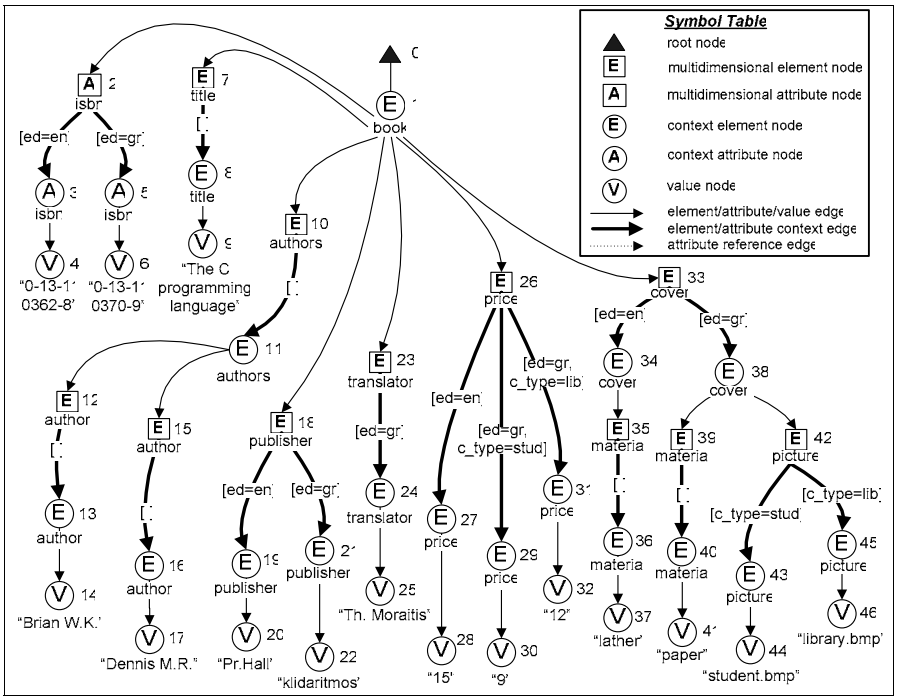

Figure 2. Graphical representation of MXML (MXML tree).

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. 2 we use obvious abbreviations for dimension names and values.

2.3 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. When elements and attributes are combined in a MXML document, the explicit context of each element/attribute e_1 is inherited to the elements/attributes contained in e_1. 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(q)* is the inherited context of its parent node *p*. $\cap^c$ is an operator called *context intersection* defined by Stavrakas and Gergatsoulis (2002), which combines two context specifiers and computes a new one, which represents the intersection of the worlds specified by the original 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 icc(n)* of a node *n* further constraints its inherited context *ic(n)*, so as to contain only the worlds under which the node *n* has access to some value node. *icc(n)* is defined as follows: if *n* is a value node then $icc(n) = ic(n)$; else if *n* is a leaf node but not a value node then $icc(n) = [-]$; otherwise $icc(n) = icc(n_1) \cup^c ... \cup^c icc(n_k)$, where $n_1, \ldots, n_k$ are the child element nodes of *n*. $\cup^c$ is an operator called *context union* defined by

Stavrakas and Gergatsoulis (2002), 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.

Based on the properties of contexts we have proposed a process called *reduction*, which given a world *w*, and a MXML document (graph) *G*, produces a conventional XML document (graph) *G'*, which is the facet of *G* under *w*. The detailed presentation of reduction is out of the scope of this paper.

3. STORING MXML IN RELATIONAL DATABASES

Many researchers have investigated how an RDBMS can be used to store and query XML data. The techniques proposed for XML storage can be divided in two categories: (a) *Schema-Based XML Storage techniques*: the task of finding a relational schema is guided by the structure of a schema for that document (Ramanath et al. 2003; Tatarinov et al. 2002; Du et al. 2004; Tian et al. 2002; Shanmugasundaram et al. 1999, 2001; Bohannon et al. 2002). (b) *Schema-Oblivious XML Storage techniques*: the relational schema is designed independently of the presence or absence of a schema for the XML data (Tatarinov et al. 2002; Du et al. 2004; Tian et al. 2002; Yoshikawa et al. 2001; Shanmugasundaram et al. 2001; Florescu and Kossmann 1999; Deutsch et al. 1999).

In this section we present techniques for storing MXML documents using relational databases. Our approaches belong to the Schema-Oblivious category. Note that Schema-Oblivious techniques are more general than Schema-Based as they do not require the existence of a DTD or XML Schema.

3.1 Naive Approach

In the *naive approach*, a single table (*Node Table*) contains all the information which is necessary to reconstruct the MXML document (MXML-graph).

Node Table

node_id	parent_id	ordinal	tag	value	type	explicit_context
1	0	1	book	-	CE	-
2	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	-
....	....	....	....	....	....	....
26	1	5	price	-	ME	-
27	26	1	price	-	CE	[ed=en]
28	27	1	-	15	VN	-
....	....	....	....	....	....	....

Figure 3. Storing the MXML-graph of Fig.2 in a Node Table

Each row in *Node Table* represents a MXML node through the attributes: node_id which stores the id of the node, parent_id which stores the id of the parent node, ordinal which stores a number denoting the order of the node among its siblings (for element and value nodes), tag which stores the label of the node or NULL (denoted by "-") if the node is a value node, value stores the value of the node if it is a value node or NULL otherwise, type which 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 which stores the explicit context of the node *as a string*. Fig. 3 is a fragment of the Node Table for the graph of Fig. 2.

In order to store context in such a way so as to facilitate the formulation of context-aware queries (see Section 4) three additional tables are introduced (see Fig. 4). The *Possible Worlds Table* 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.

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

Explicit Context Table	
node_id	world_id
1	1
1	2
1	3
1	4
….	….
28	1
28	2
28	3
28	4
….	….

Inherited Coverage Table	
node_id	world_id
1	1
1	2
1	3
1	4
….	….
28	2
28	4
….	….

Figure 4. Mapping MXML nodes of the graph of Fig.2 to worlds.

3.2 A Better 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 to the table, we observe many NULL values in the fields explicit_context, tag, and value. Those NULL values can be avoided by using different tables for different node types. Moreover, 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 MXML tree. Splitting the *Node Table* would reduce the size of the tables involved in joins, and enhance the overall performance of queries. Finally, as it will become evident in Section 4, the context representation scheme that we introduced leads to a number of joins in case we query for nodes holding under specific worlds. Introducing a better scheme might reduce the number of joins. In the *Type Approach* presented in this section, 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 (see Fig. 5, Fig. 6, and Fig.7). 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.

ME Table			
node_id	parent_id	ordinal	tag
7	1	1	title
10	1	2	authors
12	11	1	author
….	….	….	….

CE Table				
node_id	parent_id	ordinal	tag	explicit_context
1	0	1	book	-
8	7	1	title	[ ]
….	….	….	….	….
27	26	1	price	[ed=en]
29	26	2	price	[ed=gr, c_type=stud]
….	….	….	….	….

Figure 5. ME Table and CE Table.

MA Table		
node_id	parent_id	tag
2	1	isbn

CA Table				
node_id	parent_id	ordinal	tag	explicit_context
3	2	1	isbn	[ed=en]
5	2	2	isbn	[ed=gr]

Figure 6. MA Table and CA Table.

Value Table		
node_id	parent_id	value
4	3	0-13-110362-8
....	....	....
28	27	15
....	....	....

Figure 7. Value Table.

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 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. In the *Inherited Coverage* and *Explicit Context* tables we use world ids of the form ``$a_1.a_2. \; . \; .a_n$", where a_1, a_2, ..., a_n are ids of dimension values (Fig. 8).

edition	
id	value
0	*
1	en
2	gr

customer_type	
id	value
0	*
1	stud
2	lib

Explicit Context Table	
node_id	world_id
1	0.0
2	0.0
3	1.0
....	....
45	0.2
....	....

Inherited Coverage Table	
node_id	world_id
1	0.0
2	0.0
3	1.0
....	....
30	2.1
....	....

Figure 8. Dimension Tables

4. QUERYING MXML DOCUMENTS

In this section we give an overview of a multidimensional extension of XPath, called MXPath and present some examples showing how MXPath queries can be translated to equivalent SQL queries.

4.1 MXPath Overview

Multidimensional XPath (*MXPath*) is an extension of XPath able to easily express context-aware queries on MXML data. MXPath retains the characteristics of XPath and uses the inherited context coverage and the explicit context of MXML to specify navigation patterns over the additional MXML features (eg. multidimensional elements and attributes). In this section we provide a brief overview of MXPath. 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''*. The corresponding MXPath expression is:

[icc( )>="ed=gr"], /child::book[child::title="The C Programming Language"]/attribute::isbn

The [icc() >= "ed=gr"] is the *inherited coverage qualifier*, which is the first part of every MXPath, and denotes a context condition for the inherited context coverage of the result nodes. In this example, we demand that the inherited context coverage of the results (which is returned by the function icc( )) is context superset (>=) of the context [ed=gr]. The rest of the syntax in this example is similar to conventional XPath. However, each element of the form /axis::label in MXPath describes paths that include two MXML nodes, one multidimensional node and one context node. For example, the /attribute::isbn in the above MXPath starts from node with ID 1, crosses the multidimensional node labeled "isbn" with ID 2 and returns the context nodes labeled "isbn" with IDs 3 and 5.

There are two main additional features in MXPath: (a) it is possible to express conditions on the explicit context at any point of the path, by using the *explicit context qualifier* predicate, and (b) it is possible to use the characters (->) instead of (::) in order to return multidimensional nodes instead of context nodes. For example, consider the following MXPath expression:

[icc() >= “-”], /child::book/child::cover[ec() >= “ed=gr”]/child->picture

Evaluation of this expression returns the “picture” multidimensional node(s) of covers of the greek edition. On the MXML tree of our example, it will evaluate to node with ID 42. As already stated, [icc() >= “-”] is the inherited coverage qualifier denoting a context condition for the inherited context coverage of the result nodes. In this case the inherited context coverage of the results must be context superset of the empty context [-], which is always true because any context is superset of the empty context. Thus, in this case, the inherited coverage qualifier may be omitted and is implied. The [ec() >= “ed=gr”] is a predicate stating that the explicit context of “cover” context nodes must be superset of [ed=gr], therefore leading the navigation to node with ID 38. Finally, child->picture evaluates to the multidimensional node labeled “picture” which is child of node with ID 38. As in XPath, there are shorthands in MXPath. The above MXPath can take the following form:

/book/cover[ec() >= “ed=gr”]/->picture

4.2 SQL Translation

Having outlined MXPath, our aim is to translate MXPath context-aware queries to SQL queries over the aforementioned storage approaches. The key is representing context in a way that SQL can understand and handle. Encoding the explicit context and the inherited context coverage as previously described in relational tables allows us to construct SQL queries which use both the explicit context and the inherited context coverage of nodes. As an example, consider the first MXPath query given in Subsection 4.1:

[icc( )>=“ed=gr”], /child::book[child::title=“The C Programming Language”]/attribute::isbn

which returns the ISBN of the Greek edition of the book. This query would be translated into the following SQL query for the “naive” storage approach:

```
select N46.node_id, N46.value
from Node as N9, Node as N8, Node as N7, Node as N1,
      Node as N2, Node as N35, Node as N46
where N9.type="VN" and N9.value="The C Programming language" and
    N9.parent_id=N8.id and N8.type="CE" and N8.tag="title" and N8.parent_id=N7.id and
    N7.type="ME" and N7.tag="title" and N7.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=N35.parent_id and N35.type="CA" and N35.tag="isbn" and N35.id=N46.parent_id and
    N46.type="VN" and N46.id in
        (select IC1.node_id
         from Inherited_Coverage as IC1, Inherited_Coverage as IC2
         where IC1.world_id=1 and IC2.world_id=3 and IC1.node_id=IC2.node_id)
```

The “where” clause implements the navigation on the tree of Fig. 2, 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. Besides, we have included in the query some conditions, which are redundant as they are deduced from the properties of the MXML graph. For example, as we know that the child elements of a multidimensional attribute are necessarily context attributes with the same tag, the conditions N35.type="CA" and N35.tag="isbn" can be eliminated. Observe that, the “greek edition” context contains both the worlds with ids 1 and 3 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 number of nodes participated in the MXPath query expression. To see how this last point can be improved, we now give the SQL translation of same MXPath query for the “type” storage approach.

```
select N46.node_id, N46.value
from Value as N9, CE as N8, ME as N7, CE as N1, MA as N2, CA as N35, VN as N46
where N9.value="The C Programming language" and N9.parent_id=N8.id and
    N8.tag="title" and N8.parent_id=N7.id and N7.tag="title" and N7.parent_id=N1.id and
    N1.tag="book" and N1.id=N2.parent_id and N2.tag="isbn" and N2.id=N35.parent_id and
    N35.tag="isbn" and N35.id=N46.parent_id and N46.id in
            (select IC.node_id
             from Inherited_Coverage as IC
             where IC.world_id=“2.0”)
```

Note that although in this SQL query we have the same number of joins as in the previous SQL query, the joined tables are significantly smaller in this approach. Moreover, in this SQL query we do not need conditions that check the type of the node (such as the conditions N7.type="ME" and N9.type="VN" used in the previous SQL query), as in this approach the type information is derived from the table in which the node is stored. Finally, notice that the encoding of the context in the "type" approach allowed us to avoid the join in the nested query. The result returned by both SQL queries is the tuple (node_id: 6, value: "0-13-110362-9"). In this case, the result node is a leaf node holding a value. In the general case however, the result node(s) may be internal nodes. Starting from the node IDs of the result nodes, we can reconstruct from the relational tables the subtrees of which those nodes are roots, and return those subtrees as the result of the initial MXPath.

The case of an MXPath that evaluates to multidimensional nodes, as in the second example of Subsection 4.1, is not essentially different from the example we have examined in this Subsection.

5. UPDATING MXML DATA

The ability to update MXML documents is crucial in many applications as MXML data may change over time. When MXML is stored in an RDBMS then the user may define updates on the MXML graph and the system must map these updates to equivalent updates on the relational storage level and execute them so as to keep the stored data current. Several papers in the literature refer to updating XML documents stored in relational databases (Braganholo et al. 2004, 2003a, b). In this section we outline our approach for updating MXML documents which are stored in relational databases. Note that the main difference between updating MXML and updating XML documents is the existence of context information. In our approach we consider six primitive change operations, on MXML documents defined as follows:

- delete(P): All subtrees rooted at the tree nodes specified by the MXPath expression P are deleted.
- insert(P, T): The MXML tree T is inserted by unifying its root with each node specified by the MXPath expression P.
- update label(P, L): The labels of the multidimensional element/attribute nodes specified by the MXPath expression P as well as the labels of the corresponding context elements/attributes are replaced by the new label L.
- update context(P, E): The explicit context of the nodes returned by evaluating the MXPath expression P, is updated according to the context expression E.
- update value(P, V): The values of the (atomic) element/attribute nodes specified by the MXPath expression P are replaced by the new value V.
- replace(P, T): The subtrees rooted at the nodes returned by evaluating the MXPath expression P are replaced by the new subtree T.

As most of the above update operations affect the inherited context coverage of (part of) the MXML-graph, the *icc* of the affected nodes should be recalculated at the end of the update operation.

Notice that using the above operations we can change not only the data values of the MXML documents but we can also change its schema. The update operations have been defined on MXML-graph, and our current work is focused on translating them, in an automatic way, to equivalent operations applied on the stored data. A more detailed discussion of the basic update operations is omitted for space reasons.

6. CONCLUSIONS & FUTURE WORK

MXML is our suggestion for representing and handling XML data that assume different facets and have alternative variations depending on the context (Gergatsoulis et al. 2001a). Several applications of MXML have been investigated (Gergatsoulis et al. 2001b; Gergatsoulis and Stavrakas, 2003). MXML generalizes other approaches that add (mainly) temporal information in XML (Amagasa, T. et al, 2000; Wang, 2004). In this paper we briefly presented our approach for storing, querying and updating MXML documents using relational databases. We presented two approaches for storing MXML data. Then we introduced MXPath, an extension of XPath that incorporates context into navigation patterns, and showed how MXPath expressions

can be translated to SQL queries on each of the aforementioned storage approaches. Based on those translations, we gave an intuition of how these approaches can have different performance when answering queries. Finally we presented a set of basic update operations. Future work will focus on (a) algorithms for SQL translation of MXPath queries, (b) experimental evaluation of the performance of the two storage approaches, and (c) SQL translation of the basic update operations.

REFERENCES

Amagasa, T. et al, 2000. A Data Model for Temporal XML Documents. *Database and Expert Systems Applications, 11th International Conference (DEXA 2000)*, pp. 334–344.

Bohannon, P. et al, 2002. From XML Schema to Relations: A Cost-Based Approach to XML Storage. *Proceedings of the 18th International Conference on Data Engineering*. San Jose, CA, pp. 64-75.

Braganholo, V.P. et al, 2003a. UXQuery: Building Updatable XML Views over Relational Databases. *XVIII Simpósio Brasileiro de Bancos de Dados, Anais/Proceedings.* Manaus, Amazonas, Brasil, pp. 26-40.

Braganholo V.P. et al, 2003b. On the updatability of XML views over relational databases. *International Workshop on Web and Databases*. San Diego, California, pp. 31-36.

Braganholo, V.P. et al, 2004. From XML View Updates to Relational View Updates: old solutions to a new problem. *Proceedings of the Thirtieth International Conference on Very Large Data Bases*. Toronto, Canada, pp. 276-287.

Deutsch, A. et al, 1999. Storing Semistructured Data with STORED. *Proceedings ACM SIGMOD International Conference on Management of Data*. Philadelphia, Pennsylvania, USA, pp. 431-442.

Du, F. et al, 2004. ShreX: Managing XML Documents in Relational Databases. *Proceedings of the Thirtieth International Conference on Very Large Data Bases*. Toronto, Canada, pp. 1297-1300.

Florescu, D. and Kossmann, D., 1999. Storing and Querying XML Data using an RDBMS. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, **22**(3), pp. 27-34.

Gergatsoulis, M. and Stavrakas Y., 2003. Representing Changes in XML Documents using Dimensions. *Database and XML Technologies, Fifth International XML Database Symposium, XSym 2003*, pp. 208-222.

Gergatsoulis, M. et al, 2001a. Incorporating Dimensions to XML and DTD. *Database and Expert Systems Applications, 12th International Conference.* Munich, Germany, pp. 646-656.

Gergatsoulis, M. et al, 2001b. A Web-based System for Handling Multidimensional Information through MXML. *Advances in Databases and Information Systems 5th East European Conference*. Vilnius, Lithuania, pp. 352-365.

Ramanath, M. et al, 2003. Searching for Efficient XML-to-Relational Mappings. *First International XML Database Symposium*. Berlin, Germany, pp. 19-36.

Shanmugasundaram, J. et al, 2001. A General Technique for Querying XML Documents using a Relational Database System. *SIGMOD Record*, **30**(3), pp. 20-26.

Shanmugasundaram, J. et al, 1999. Relational Databases for Querying XML Documents: Limitations and Opportunities. *Proceedings of 25th International Conference on Very Large Data Bases.* Edinburgh, Scotland, pp. 302-314.

Stavrakas, Y. and Gergatsoulis M., 2002. Multidimensional Semistructured Data: Representing Context-Dependent Information on the Web. *14th International Conference in Advanced Information Systems Engineering.* Toronto, Canada, pp. 183-199.

Tatarinov, I. et al, 2002. Storing and querying ordered XML using a relational database system. *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Madison, Wisconsin, pp. 204-215.

Tian, F. et al, 2002. The Design and Performance Evaluation of Alternative XML Storage Strategies. *SIGMOD Record*, **31**(1), pp. 5-10.

Wang, F. 2004. XML-Based Support for Database Histories and Document Versions. *Ph.D. Dissertation*. Computer Science Department, University of California, Los Angeles.

Yoshikawa, M. et al, 2001. XRel: a path-based approach to storage and retrieval of XML documents using relational databases. *ACM Transactions on Internet Technology*, **1**(1), pp. 110-141.