UPDATING MULTIDIMENSIONAL XML DOCUMENTS [1)]

Nikolaos Fousteris, Manolis Gergatsoulis, Yannis Stavrakas
Department of Archive and Library Science, Ionian University,
Ioannou Theotoki 72, 49100 Corfu, Greece.
{nfouster, manolis}@ionio.gr, ys@dblab.ntua.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/or structure, under different contexts. In this paper, we consider the problem of updating MXML. Updating must take into account the additional features of MXML compared to XML. Those features stem from the incorporation of context into MXML. We introduce six basic update operations, which are suitable for any possible change in MXML.

1 Introduction

Multidimensional XML (MXML) [5] 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 previous work [4], the problem of storing MXML in a Relational Database (RDB) has been studied. This problem has been extensively studied in the past [3, 7, 10] for XML. However, the problem of updating MXML documents has not been studied yet although the problem of updating conventional XML stored in relational databases has also been studied in the past [1, 2]. The goal of our approach is to develop a similar framework for MXML. The main contribution of the present paper is that we define a set of six basic change operations for updating MXML documents. We give algorithmic definitions of the operations in a way independent of any specific storage approach for MXML. Moreover, we discuss in detail the effect of those operations on MXML

[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".

documents and give relevant examples. We also give an overview of an extension of XPath, called *Multidimensional XPath* (MXPath), which is used here to specify the nodes of the MXML tree on which the update operations are applied.

2 Preliminaries

2.1 Mutidimensional XML (MXML)

In Mutidimensional XML (MXML), data assume different facets, having different value or structure, under different contexts[5, 6]. Contexts are described through syntactic constructs called *context specifiers* and are used to specify sets of *worlds* by imposing constraints on the values that a set of user defined *dimensions* can take. A *world*, which represents an environment under which data obtain meaning, is determined by assigning a single value to every dimension, taken from the domain of the dimension. The elements/attributes that have different facets under different contexts are called *multidimensional elements/attributes* while their facets are called *context elements/attributes*, and are accompanied with a corresponding context specifier denoting the set of worlds under which each facet is the holding one. MXML documents can be represented in a node-based graphical model called *MXML-graph* [4]. The syntax of MXML is shown in Example 2.1.

Example 2.1. The MXML document shown below represents a book in a book store. Two dimensions are used in the document. The dimension `edition` whose domain is {`greek, english`}, and the dimension `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 the name of a multidimensional elements is preceded by the symbol `@` while the corresponding context elements have the same element name but without the symbol `@`. The MXML-graph representing the MXML document presented above is shown in Fig. 1. For saving space, obvious abbreviations for dimension names and values are used. IDs assigned to nodes are used in this paper to facilitate the reference to the nodes. However, these IDs are also used when the MXML tree is stored in relational tables as explained in [4].

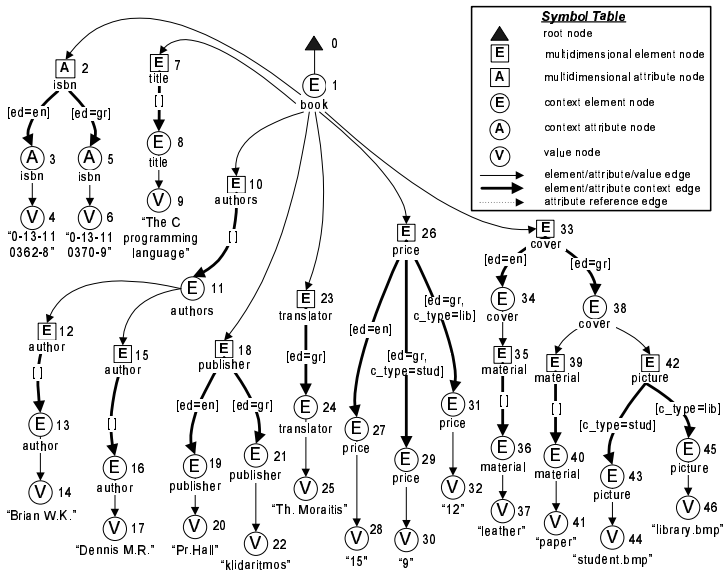

Figure 1: Graphical representation of MXML (MXML graph)

2.2 Properties of contexts

Context specifiers qualifying context edges give the *explicit contexts* of the nodes to which the edges lead. The explicit context of all the other nodes is by definition the *universal context* represented by [], denoting the set of all possible worlds. When elements and attributes are combined to form a MXML document, their explicit contexts do not alone determine the

worlds under which each 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 is conceived as if the context of e_1 is inherited to e_2. The context propagated in that way is combined with the explicit context of a node to give its *inherited context*. Formally, the inherited context $ic(q)$ of a node q is $ic(q) = ic(p) \cap^c ec(q)$, where $ic(p)$ is the inherited context of its parent node p and $\cap^c$ is the *context intersection* defined in [9]. $\cap^c$ combines two context specifiers and computes a new one representing 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 is the universal context [ ]. 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) 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* of MXML to XML [6, 4]. The $icc(n)$ of a node 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 $icc(n) = icc(n_1) \cup^c icc(n_2) \cup^c ... \cup^c icc(n_k)$, where $n_1, \ldots, n_k$ are the child element nodes of n. $[-]$ stands for the *empty context specifier* which represents the empty set of worlds. $\cup^c$ is the *context union* operator defined in [9] which combines two context specifiers and computes a new one representing the union of the worlds specified by the original context specifiers.

Example 2.2. In Fig. 2 we see a fragment of the MXML graph of Fig. 1 in which it is shown

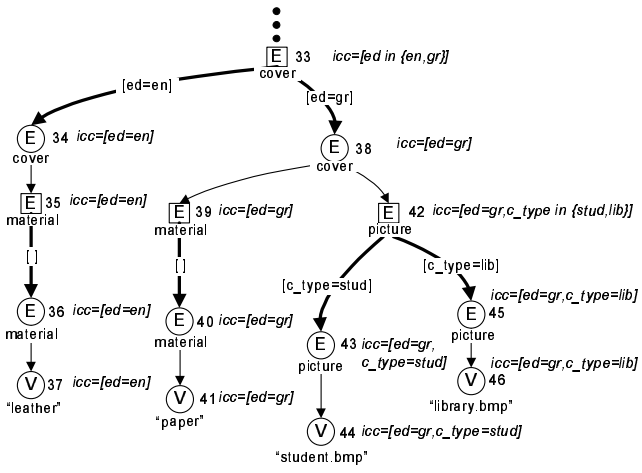

Figure 2: Representing the Inherited Context Coverage

the value of the inherited context coverage (icc) for every node of the subtree.

3 Multidimensional XPath

Multidimensional XPath (MXPath) is an extension of XPath that uses the inherited context coverage and the explicit context of MXML to specify navigation patterns over the additional MXML features. A detailed discussion of MXPath is out of the scope of this paper, however in this section we provide a brief overview. 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[title=‘‘The C Programming Language’’]/attribute::isbn
```

The `[icc()>=‘‘ed=gr’’]` is the *icc qualifier*, which is the first part of every MXPath, and denotes a condition for the *icc* of the result nodes. In this example, we demand that the *icc* of the results (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` describes paths that include two MXML nodes, one multidimensional node and one context node. For example, the `attribute::isbn` looks for attributes of node 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 of MXPath: (a) it allows conditions on the explicit context of a node, by using the *ec 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:

```
[icc()>=‘‘-’’], /child::book/child::cover[ec()>=‘‘ed=en’’]/child− >material
```

It returns the "`material`" multidimensional nodes of covers of the english edition. On the tree of Fig. 1, it will evaluate to node with ID 35. Note that `[icc()>=‘‘-’’]` denotes that the inherited context coverage of the results must be context superset of the empty context `[-]`. As this is always true, in this case the inherited coverage qualifier may be omitted and is implied. The predicate `[ec()>=‘‘ed=en’’]` states that the explicit context of "`cover`" context nodes must be superset of `[ed=en]`, thus leading the navigation to node 34. Then, `child->material` evaluates to the multidimensional nodes labeled "`material`" which are children of node 34. As in XPath, there are shorthands in MXPath. The above MXPath can take the form:

```
/book/cover[ec()>=‘‘ed=en’’]/− >material
```

4 MXML Update Operations

In this section we define a set of basic update operations namely *delete*, *insert*, *update_label* *update_context*, *update_value* and *replace* that are used to modify MXML documents (MXML graphs). These operations can be combined to perform any possible change on a document, and when the apply on a well-formed MXML document, the document obtained is well-formed.

a) Deleting subtrees: The operation *delete(P)* is used to delete subtrees rooted at specific (context or multidimensional) nodes specified by the MXPath P. The operation is defined by:

***delete(P)**:*
Input: ***P**: MXPath expression.*
1. Let N_P be the set of nodes returned by evaluating P.
2. ***For each** $n \in N_P$ **do***
- Delete the subtree rooted at n and the edge leading to n.
- Recalculate the iccs of n and its ancestors in the tree.
- Delete the subtree rooted at each node m for which $icc(m) = [-]$ as well as the edges leading to m.

Example 4.1. The tree in Fig. 3(b) is obtained by deleting the subtree rooted at the node

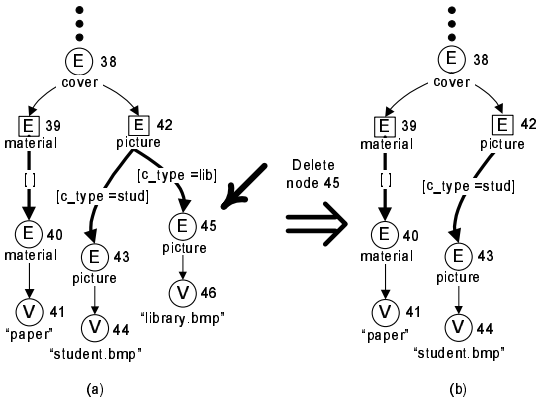

Figure 3: Deleting a node

45 from the tree in Fig. 3(a) (which is a fragment of the tree in Fig. 1). By recalculating the icc's of the nodes from the node 42 to the root we get `icc(42)=[ed=gr,c_type=stud]`, `icc(38)=[ed=gr]`, `icc(33)=[ed in {gr,en}]` and `icc(1)=[]`. Notice that the MXPath P returning node 45 may be: `/book/cover[ec()=''ed=gr'']/picture[ec()=''c_type=lib'']`

b) Inserting Subtrees: Inserting a (well-formed) tree T at specific points of the MXML tree, specified by an MXPath expression P, is done through the operation *insert(P, T)*:

***insert(P, T)**:*
Input: ***P**: MXPath expression.*
***T**: A well-formed MXML tree.*
1. Let N_P be the set of nodes returned by evaluating P.
2. ***For each** $n \in N_P$ **do***
If** n and $root(T)$ are multidimensional nodes **and** $label(n) = label(root(T))$ **then
- Let C_n be the set of context specifiers of all context edges departing from n.
- Let C_T be the set of context specifiers of all context edges departing from the root of T.
*- **If** $C_n \cup C_T$ is a context deterministic set of context specifiers **then***
- Hang T on the node n (unify n with the root of T).
- Recalculate the icc's of all ancestors and descendants of n in the resulted tree.
else If** n and $root(T)$ are context nodes **and** $label(n) = label(root(T))$ **then
- Hang T on the node n (unify n with the root of T).
- Recalculate the iccs of n and all its ancestors and descendants in the resulted tree.
3. Delete the subtree rooted at each node m for which $icc(m) = [-]$ as well as the edges leading to m.

Notice that: (a) Insertion applies only to nodes that are of the same type and have the same label with the root node of T. (b) If the root of T is multidimensional, the algorithm ensures that the tree obtained is context deterministic. (c) If P returns multiple nodes, the insertion of T is applied for each node separately. The sequence of the nodes does not play any role.

Example 4.2. The tree in Fig. 4(c) is obtained by inserting the tree T shown in Fig. 4(b) at the node 34 of the tree in Fig. 4(a) (which is a fragment of the tree in Fig. 1). By

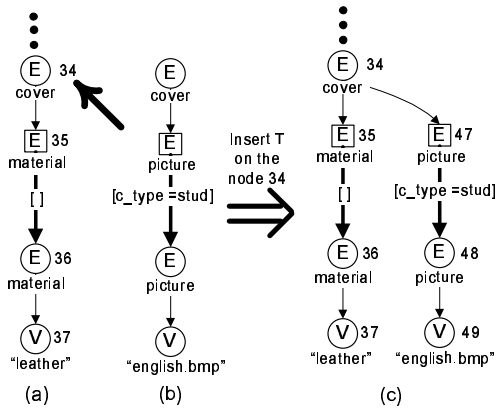

Figure 4: Inserting a subtree

recalculating the icc's of the nodes we get `icc(49)=icc(48)=icc(47) =[ed=en,c_type=stud]` and `icc(34)=[ed=en]`. Note that P is the MXPath: `/book/cover[ec()=‘‘ed=en’’]`.

c) Updating Label: The following operation is used to update the label of the nodes:

update_label(P, L):
Input: ***P****: MXPath expression.*
L*: A node label.*
1. Let N_P be the set of nodes returned by evaluating P.
2. ***For each*** *multidimensional node $n \in N_P$* ***do***
- Replace the label of n by L.
- Replace the label of each child context node of n by L.

Note that *update_label* applies only to multidimensional nodes. This is not a restriction when the MXML tree is in *canonical form* [8]. Besides, it prevents inconsistent situations in which a child context element has different label from its parent multidimensional element.

Example 4.3. In Fig. 5 we see how *update_label* operation applies to node 42 and changes the label `picture` of node 42 (and of its child context nodes 43 and 45), to the new label `image`. The argument P is: `/child::book/child::cover[ec()=‘‘ed=gr’’]/child->picture`

d) Updating Context: The operation *update_context(P, E)* is used to update the explicit context of the nodes returned by evaluating the MXPath expression P. E is a *context expression* which specifies, through the use of operations on contexts (e.g. context union, context

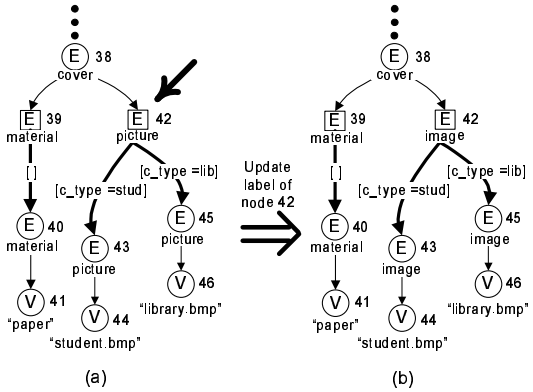

Figure 5: Updating label

intersection etc.), how the new explicit contexts will be constructed. For space saving we will not refer in detail to the syntax of context expressions. This operation is defined as follows:

***update_context(P, E)**:*
Input: ***P**: MXPath expression.*
***E**: A context expression.*
1. Let N_P be the set of nodes returned by evaluating P.
2. Discard all non-context nodes from N_P. Let M *be a partition of N_P such that each member of the partition consists of all nodes in N_P which are siblings.*
3. ***For each** $m \in M$ **do***
- *Collect in a set S_m all explicit contexts of the nodes in m and in S the explicit contexts of all siblings of the nodes in m which are not in m.*
- *Let S_m^E be the set of context specifiers obtained by applying E to the elements of S_m.*
- ***If** $S_m^E \cup S$ is a context deterministic set of context specifiers **then***
  - *Replace the explicit context of each node in m by the corresponding element of S_m^E.*
  - *Recalculate the iccs of all nodes in m and all their ancestors and descendants in the resulted tree.*

4. Delete the subtree rooted at each node n for which $icc(n) = [-]$ as well as the edges leading to n.

The *update_context* operation applies only to context element/attribute nodes as these nodes possess a (user defined) explicit context. Note that when the evaluation of P returns two or more context nodes which are all children of the same multidimensional node, these nodes are updated simultaneously because the final tree must be context deterministic.

Example 4.4. The tree in Fig. 6(b) is obtained by updating the context of the node 45 in the tree of Fig. 6(a) (which is a fragment of the tree in Fig. 1). The MXPath expression P is:

`/child::book/child::cover[ec()=''ed=gr'']/child::picture[ec()=''c_type=lib'']`

The context expression E = `[+(c_type=museum)]` adds "`museum`" to the values of the dimension "`c_type`" to form the new explicit context of node 45 which becomes `[c_type in {lib,museum}]`. Recalculating the iccs we get `icc(46)=icc(45)=[c_type in {lib,museum}, ed=gr]` and `icc(42)=[ed=gr,c_type in {lib,stud,museum}]`. Note that, if the expression `E` is `E=[+(c_type=stud)]`, we can not apply the context update because the resulted tree is not context deterministic (as `ec(43)=[c_type=stud]` and `ec(45)=[c_type in {lib,stud}]`).

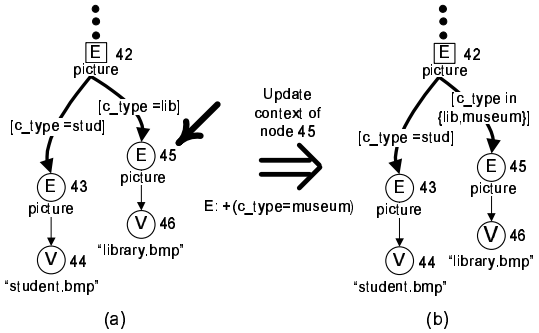

Figure 6: Updating the context of a node

e) Updating Values: The operation *update_value(P, C)* replaces the value of the leaf nodes specified by the MXPath expression P, by a new value obtained by evaluating the v*alue expression* C. This operation is simple and its formal definition is omitted for space reasons.

f) Replacing Subtrees: The operation *replace(P, T)*, replaces the subtrees rooted at the (context or multidimensional) nodes returned by evaluating P, by the MXML tree T. The root nodes of the replaced subtrees must match in type and in label with the root node of T.

replace(P, T):
Input: ***P***: *MXPath expression.*
T: *A well-formed MXML tree.*
1. *Let N_P be the set of nodes returned by evaluating P.*
2. ***For each*** *$n \in N_P$* ***do***
   If *n and $root(T)$ are nodes of the same type (context or multidimensional)* ***and***
   $label(n) = label(root(T))$ ***then***
   - *Replace the subtree with root node n with the subtree T.*
   - *Recalculate the iccs of n and all its ancestors and descendants in the resulted tree.*
3. *Delete the subtree rooted at each node m for which $icc(m) = [-]$ as well as the edges leading to m.*

Example 4.5. In Fig. 7 we see the tree obtained (see Fig. 7(c)) by replacing the tree rooted at node 42 (see Fig. 7(a) which is a fragment of the MXML graph of Fig. 1) by the tree shown in Fig. 7(b). By recalculating the icc's we get `icc(48)=icc(47)=icc(42)=icc(38)=[ed=gr]`.

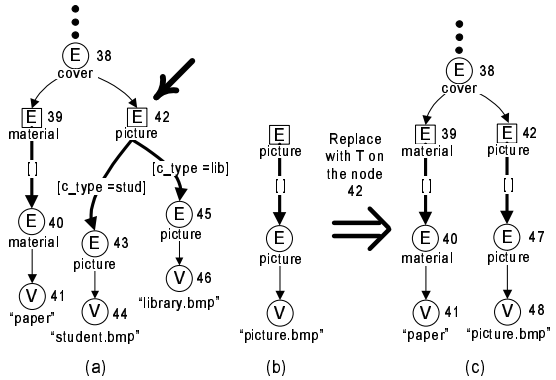

Figure 7: Replacing a subtree

5 Conclusions and Future Work

In this paper we investigated the problem of updating MXML documents and presented a set of basic change operations which can be used for any possible update of the document. The present work is part of a framework aiming at storing, querying and updating MXML documents using relational databases. Approaches for storing MXML in relational databases, where the nodes and edges of MXML-graphs are mapped to appropriately chosen relational table, are presented in [4]. In both storing approaches presented in [4], extra tables are employed to store the explicit context and the inherited context coverage of each node. The target of our future work is twofold: (a) to see how the proposed update operations defined at the level of MXML-graph can be mapped to update operations on the relational tables which store the MXML data, and (b) to formally define MXPath, and translate MXPath expressions to equivalent SQL queries.

References

[1] V. P. Braganholo, S. B. Davidson, and C. A. Heuser. From XML View Updates to Relational View Updates: old solutions to a new problem. In *Proc. of VLDB'04*, pp. 276–287. Morgan Kaufmann, 2004.

[2] Vanessa P. Braganholo, Susan B. Davidson, and Carlos A. Heuser. On the updatability of XML views over relational databases. In *Int. Workshop on Web and Databases*, pp. 31–36, 2003.

[3] A. Deutsch, M. F. Fernandez, and D. Suciu. Storing Semistructured Data with STORED. In *SIGMOD 1999, Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 431–442. ACM Press, 1999.

[4] N. Fousteris, M. Gergatsoulis, and Y. Stavrakas. Storing Multidimensional XML documents in Relational Databases, In *Proc. of DEXA'07*, pp. 23–33. Springer, 2007.

[5] M. Gergatsoulis, Y. Stavrakas, and D. Karteris. Incorporating Dimensions in XML and DTD. In *Proc. of DEXA'01*, pp. 646–656. Springer, 2001.

[6] 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*, pp. 352–365. Springer, 2001.

[7] 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.

[8] Y. Stavrakas. *Multidimensional semistructured data: representing and querying context-dependent multi-facet information on the web*. PhD thesis, Department of Electrical and Computer Engineering, National Technical University of Athens, Greece, June 2003.

[9] Y. Stavrakas, and M. Gergatsoulis. Multidimensional Semistructured Data: Representing Context-Dependent Information on the Web. In *Proc. of CAiSE'02*, pp. 183–199, Springer 2002.

[10] 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*, pp. 204–215. ACM, 2002.