Path-based MXML Storage and Querying

Nikolaos Fousteris ¹, Manolis Gergatsoulis ¹, Yannis Stavrakas ²

¹ Department of Archives and Library Science, Ionian University
Ioannou Theotoki 72,49100 Corfu, Greece.
{nfouster,manolis}@ionio.gr

² Institute for the Management of Information Systems (IMIS),
R. C. Athena,
G. Mpakou 17, 11524, Athens, Greece.
yannis@inmis.gr
Introduction & Motivation

- The problem of storing and querying XML data using relational databases has been considered a lot.

- Multidimensional XML is an extension of XML and it is used for representing data that assume different facets, having different values or structure, under different contexts.

- We expand the problem of storing and querying XML to multidimensional XML data.
Outline

- XML Storage
- Multidimensional XML (MXML)
  - Fundamental concepts
  - MXML example and graphical representation
- MXML Storage
  - A path-based approaches is presented
- Context Representation
- Multidimensional XPath (MXPath)
- MXPath to SQL conversion algorithm
- Summary & Future work
XML Storage (1/2)

- Includes techniques to store XML data in Relational Databases

- XML applications (internet applications) are able to exploit the advantages of the RDBMS technology

- Operations over XML data, are transformed to operations over the Relational Schema
XML Storage (2/2)

- **Methodology**
  - A Relational Schema is chosen for storing XML data
  - XML queries are produced by users/applications
  - XML queries are translated to SQL queries
  - SQL queries are executed
  - Results are translated back to XML and returned to the user/application

- **Techniques**
  - Schema Based
  - Schema Oblivious
Multidimensional XML (MXML)  
Fundamental Concepts (1/3)

- MXML is an extension of 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
**MXML – Fundamental Concepts (2/3)**

- **Dimension**: is a variable. Assigning different values for each dimension it is possible to construct different environments for MXML data.

- **World**: represents an environment under which data obtain a meaning and is determined by assigning to every dimension a single value.

- **Context Specifier**: an expression which specifies a set of worlds (context) under which a facet of an MXML element or attribute, is the holding facet of this element or attribute.
<book isbn="0-13-110362-8" [edition=english]>
  [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>
    ..........
  </@ price>
</book>

**Multidimensional elements/attributes** are elements/attributes that have different facets under different contexts.

Each multidimensional element/attribute contains one or more facets, called **Context element/attributes**.
**Explicit Context**: Is the true context (defined by a context specifier) only within the boundaries of a single multidimensional element/attribute.

**Inherited Context**: Is the context, which is inherited from the ancestor nodes to a descendant node in the MXML graph.

**Inherited Context Coverage**: It constraints the inherited context of a node, so as to contain only the worlds under which the node has access to some value node.
MXML Storage (1/5)

- MXML storage includes techniques that store MXML data in Relational Databases.

- Applications using MXML storage are able to exploit the advantages of the RDBMS technology.

- MXML additional features (context, different types of MXML nodes/edges etc.) should be considered.
Path-based Approach

- MXML nodes are divided into groups, according to their types. Each group is stored in a separate table named after the type of the nodes (elements, attributes and value nodes).

- There is a *Path Table*, which stores all possible paths of the MXML tree.

- For node indexing, it is used a dotted format of Dewey-labeling schema.
MXXML Storage (3/5)

Dewey-labeling schema

- Used for indexing the nodes of MXML tree.
- A label is a dotted string \( a_1.a_2.a_3...a_n \).
- For each \( a_i \) (i=1...n), \( i \) represents the depth and \( a_i \) the position number of a node among its siblings.

Ex. Node “1.1.2” is the 2\(^{nd}\) child of node “1.1” and is placed at the 3\(^{rd}\) level of the MXML tree.
MXML Storage (4/5)

Path-based Approach

Attribute Table

<table>
<thead>
<tr>
<th>node_id</th>
<th>path_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.1</td>
<td>3</td>
</tr>
<tr>
<td>1.1.1.1</td>
<td>4</td>
</tr>
<tr>
<td>1.1.1.2</td>
<td>4</td>
</tr>
</tbody>
</table>

Path Table

<table>
<thead>
<tr>
<th>path_id</th>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>#/-&gt;book</td>
</tr>
<tr>
<td>2</td>
<td>#/book</td>
</tr>
<tr>
<td>3</td>
<td>#/book#/@-&gt;isbn</td>
</tr>
<tr>
<td>4</td>
<td>#/book#/@isbn</td>
</tr>
<tr>
<td>5</td>
<td>#/book#/&gt;-&gt;title</td>
</tr>
<tr>
<td>6</td>
<td>#/book#/title</td>
</tr>
<tr>
<td>....</td>
<td>....</td>
</tr>
</tbody>
</table>

Element Table

<table>
<thead>
<tr>
<th>node_id</th>
<th>path_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>2</td>
</tr>
<tr>
<td>1.1.2</td>
<td>5</td>
</tr>
<tr>
<td>1.1.2.1</td>
<td>6</td>
</tr>
<tr>
<td>....</td>
<td>....</td>
</tr>
</tbody>
</table>

Value Table

<table>
<thead>
<tr>
<th>node_id</th>
<th>path_id</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.1.1</td>
<td>4</td>
<td>0-13-110362-8</td>
</tr>
<tr>
<td>1.1.1.2</td>
<td>4</td>
<td>0-13-110370-9</td>
</tr>
<tr>
<td>1.1.2.1.1</td>
<td>6</td>
<td>The C programming language</td>
</tr>
<tr>
<td>....</td>
<td>....</td>
<td>....</td>
</tr>
</tbody>
</table>
Path-based Approach

Path representation

Ex.

**XPath**: /book//picture

**SQL1**:
select.. from.. where path LIKE ‘/book%/picture’

case1: ‘/book/cover/picture’ (match) correct
case2: ‘/booklet/cover/picture’ (match) error

**SQL2**:
select.. from.. where path LIKE ‘#/book#/picture’

Cases like case2 above could not happen.
Question

How can we represent in a Relational Database the set of worlds which are contained in a context specifier, for each MXML node?
Context Representation (2/5)

Ordered-Based Representation of Context

**Basic idea**: Total ordering of worlds based on:

- Total ordering of dimensions
- Total ordering of dimension values

For $k$ dimensions with each dimension $i$ having $z_i$ possible values, we may have $n = z_1 * z_2 * \ldots * z_k$ possible ordered worlds.

Each world is assigned a unique integer value between 1 and $n$ ($w_1$ to $w_n$).
Context Representation (3/5)

Ordered-Based Representation of Context

Dimensions ordering

Dimension values ordering

Possible worlds ordering

1. gr
   - stud
     - w1=(gr,stud)
   - lib
     - w2=(gr,lib)

2. en
   - stud
     - w3=(en,stud)
   - lib
     - w4=(en,lib)

Levels:
- Level1 (Edition)
- Level2 (C_type)
Context Representation (4/5)

**Ordered-Based Representation of Context**

**World Vector:**
- A binary number representing a context specifier. The position of every bit corresponds to the position of a world in the total ordering of all possible worlds.
- Each bit of the world vector has two possible values: 1 if the corresponding world exists in context specifier or 0 if it does not.

<table>
<thead>
<tr>
<th>binary digit for W1</th>
<th>......</th>
<th>binary digit for Wi</th>
<th>......</th>
<th>binary digit for Wn</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 or 0: world exists or not</td>
<td>......</td>
<td>n=possible worlds number</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Ex: world_vector** of the expl. context of **node 1.1.6.1 = 0011**
Explicit Context Table:

Assigns an **explicit context** (expressed in binary format according to world vector representation) to a MXML node.

Inherited Context Coverage Table:

Assigns an **inherited context coverage** (expressed in binary format according to world vector representation) to a MXML node.
MXPath:

- An extension of XPath able to easily express context-aware queries on MXML data.
- Both explicit context (ec) and inherited context coverage (icc) are used to navigate over multidimensional elements and attributes.
- Conditions on the explicit context at any point of the path are allowed.
- Both multidimensional and context nodes can be returned.
MXPath example:

\[\text{icc() >= "-"}, /\text{child::book} /\text{child::cover[ec() >= "ed=gr"]}/\text{child->picture}\]

Query in English:

*Find the (multidimensional) sub-element picture of element cover of the greek edition of the book.*

\(\text{cover[ec() >= "ed=gr"]}\) is an *explicit context qualifier*. The function \(\text{ec()}\) returns the explicit context of a node. The above qualifier says that the ec of the node \(\text{cover}\) must be superset of the context described by the context specifier \([ed=gr]\).
Methodology:

- Give a MXPath query “Q”, we construct the Multidimensional Tree Pattern “G” of Q.
- We divide “Q” in sub-paths according to segmentation rules, which define the appropriate segmentation points in “G”.
- The sub-paths of “Q” and the predicates are used as input for the MXPath to SQL conversion algorithm.
MXPath to SQL conversion algorithm (2/4)

**MXPath example:**
/book[authors[author="Brian W.K."]]/cover[ec()="ed=gr"]/->material

Query in English:
*Find the (multidimensional) sub-element material of element cover of the greek edition of the book which has an author named “Brian W.K.”.*

Notice that the predicate [authors[author="Brian W.K."]]
is a branch.
MXPath to SQL conversion algorithm (3/4)

MXPath query “Q”:
/book[authors[author="Brian W.K."]]
/cover[ec()="ed=gr"]/->material

Multidimensional Tree Pattern

Segmentation (Q):
Segmentation branch ( authors[author="Brian W.K."] ):
Sub_Path_1: #/book#/authors#/author
Value_1="Brian W.K."
**MXPath to SQL conversion algorithm (4/4)**

**MXPath:** `/book[authors[author="Brian W.K."]]/cover[ec()="ed=gr"]/->material`

**SQL:**

```sql
SELECT a3.node_id
FROM Element_Table a1, Element_Table a2,
     Element_Table a3, Path_Table p1,
     Path_Table p2, Path_Table p3,
     EC_Table ec1
WHERE p1.path Like '#/book'
AND a1.path_id = p1.path_id
AND p1.path_id = p1.path_id
AND p1.path_id = p1.path_id
AND p1.value = 'Brian W.K.'
AND a1.node_id Like CONCAT(a1.node_id,'%')
) AS T
) AS a1.node_id AS a_id
FROM Element_Table a1, Path_Table p1,
     Value_Table v1
WHERE p1.path Like '#/book#/authors#/author' AND
     a1.path_id = p1.path_id
     v1.path_id = p1.path_id
     v1.value = 'Brian W.K.'
AND a1.node_id Like CONCAT(a1.node_id,'%')
) AS T
AND a2.node_id Like CONCAT(a1.node_id,'%')
AND p2.path Like '#/book#/cover' AND
     a2.path_id = p2.path_id
AND a2.node_id = ec1.node_id
AND ec1.world_vector LIKE '000111'
AND a3.node_id Like CONCAT(a2.node_id,'%')
AND p3.path Like '#/book#/cover/->material' AND
     a3.path_id = p3.path_id
```

---

**DBIS**

Database & Information Systems Group

Ionian University
Summary

- MXML
- Storing MXML in Relational DB & Context Representation (path-based approach)
- MXML querying using MXPath
- Converting MXPath queries to SQL queries

Future work

- Conversion Algorithm implementation and evaluation
- Further optimization of the relational schema


Thank you..