Multidimensional XPath

Nikolaos Fousteris
Department of Archive and Library Sciences,
Ionian University
Ioannou Theotoki 72,
49100 Corfu, Greece.
nfouster@ionio.gr

Yannis Stavrakas
Institute for the Management of Information Systems (IMIS),
R. C. Athena,
G. Mpakou 17, 11524, Athens, Greece.
yannis@imis.athena-innovation.gr

Manolis Gergatsoulis
Department of Archive and Library Sciences,
Ionian University
Ioannou Theotoki 72,
49100 Corfu, Greece.
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. In this paper, we consider the problem of navigating and querying MXML. Navigating and querying must take into account the additional features of MXML compared to XML. Those features stem from the incorporation of context into MXML. We introduce an extension of XPath called *Multidimensional XPath* (MXPath), which is suitable for navigating in MXML documents, and allows for context-aware querying. We present the syntax of MXPath, we provide examples demonstrating its use and investigate its semantics.

Categories and Subject Descriptors

H.2 [**Database management**]: Languages—*query languages*;
H.2.1 [**Logical Design**]: Data models

General Terms

Semistructured databases

Keywords

Multidimensional semistructured databases, XPath, XML, XML query

1. INTRODUCTION

In recent WWW applications, it is often necessary to manipulate a huge amount of semistructured data, often represented in XML format [1, 19]. 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. Multidimensional XML (MXML) [11] 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 parameters, called *dimensions*.

MXML is an instance of the more general formalism of *Multidimensional Semistructured Data (MSSD)* [16]. MXML (and MSSD) has been proved useful in various applications including the representation of the history of XML documents [10] or semistructured data [17] and the representation and delivery of multidimensional information in the web [12]. In [5], the formalism of MSSD is adopted for representing contextual information within the service directory. In [14], a model and mechanisms for supporting context-dependent information delivery, is proposed. The proposed model is based on an approach which is similar to the MSSD data model.

In this paper we propose an extension of XPath called *Multidimensional XPath* (MXPath) for navigating and querying MXML documents. MXPath uses contexts in order to specify navigation patterns in MXML-graphs. We present the syntax of MXPath and illustrate its use through a number of appropriate example queries over a MXML document. We also investigate the relation of MXPath with the conventional XPath, when considering XML instances of MXML documents. Finally, we explain how MXPath may be used to extend other languages, such as XQuery and XSLT, which are based on XPath expressions.

Until now, a number of query languages for XML have been proposed [2, 4, 13, 15]. XPath [21] provides navigation abilities within a XML document. For that reason, XPath is a basic ingredient in many query languages [18, 20, 22]. On the other hand, some research papers [3, 6, 23] propose extensions of XPath. However, these papers consider only temporal extensions to XPath. More specifically, in [23] an extension of the XPath data model to include valid time, is presented. In [3] a logical data model for representing histories of XML documents is proposed. The proposed model extends the XPath data model, and is capable of representing change histories of XML documents. Also, in [6] a transaction-time XPath (TTPATH) data model and query language is presented. The TTXPath query language extends XPath with a transaction-time axis to enable a query to access past or future states of a XML document, and with constructs to extract and compare times.

2. PRELIMINARIES

2.1 Multidimensional XML (MXML)

In Mutidimensional XML (MXML), data assume different facets, having different value or structure, under different contexts [11, 12]. 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* 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. The syntax of multidimensional elements is as follows:

```
<@element_name  attribute_specification>
  [context_specifier_1]
    <element_name attribute_specification_1>
       element_content_1
    </element_name>
  [/]
  . . .
  [context_specifier_N]
    <element_name attribute_specification_N>
       element_content_N
    </element_name>
  [/]
</@element_name>
```

To declare a multidimensional attribute the following syntax is used:

```
attribute_name =
   [context_specifier_1] attribute_value_1 [/] ...
   [context_specifier_n] attribute_value_n [/]
```

Example 1. *The MXML document shown below describes a specific car model, whose specification vary according to the factory it is produced, and the market it is exported. Two dimensions are used in the document. The dimension* `factory` *whose domain is* {`Japan, Italy`}*, and the dimension* `market` *whose domain is* {`USA, Europe`}*.*

```
<car type=[factory=Japan]"sport"[/]
          [factory=Italy]"family"[/]>
  <@designer>
     [factory=Japan]
        <designer>groupo Bertone</designer>
     [/]
     [factory=Italy,market=Europe]
        <designer>Pedro Seelig</designer>
     [/]
     [factory=Italy,market=USA]
        <designer>Rollo Dixon</designer>
     [/]
  </@designer>
  <@engine>
     [factory=Japan]
        <engine>
           <capacity>1.8lt</capacity>
           <@power>
             [market=Europe]
                <power>180hp</power>
             [/]
             [market=USA]
                <power>200hp</power>
             [/]
           </@power>
        </engine>
     [/]
     [factory=Italy]
        <engine>
           <capacity>1.6lt</capacity>
           <@power>
             [market=Europe]
                <power>120hp</power>
             [/]
             [market=USA]
                <power>140hp</power>
             [/]
           </@power>
        </engine>
     [/]
  </@engine>
  <@performance>
     [factory=Japan]
        <performance>
           <top_speed>250km/h</top_speed>
           <@acceleration>
             [market=Europe]
                <acceleration>0-100 in 6sec
                   </acceleration>
             [/]
             [market=USA]
                <acceleration>0-100 in 5sec
                   </acceleration>
             [/]
           </@acceleration>
        </performance>
     [/]
     [factory=Italy]
        <performance>
           <acceleration>0-100 in 5sec
              <acceleration/>
           <@top_speed>
             [market=Europe]
                <top_speed>200km/h</top_speed>
             [/]
             [market=USA]
                <top_speed>210km/h</top_speed>
             [/]
           </@top_speed>
        </performance>
     [/]
  </@performance>
</car>
```

Notice that the name of a multidimensional element is preceded by the symbol `@` *while the corresponding context elements have the same name but without the symbol* `@`*.*

MXML documents can be represented using a node-based graphical model called *MXML-graph* [7]. The MXML-graph in Figure 1 represents the MXML document of Example 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.

2.2 XPath

XPath (XML Path Language) [21] is a language proposed by W3C for addressing portions of a XML document. XPath is based on a tree representation of a XML document, and provides the ability to navigate through elements and attributes in the XML tree, selecting nodes by specifying a variety of criteria. The basic structural unit of XPath is the *XPath expression*, which may return either a node-set, a string, a Boolean, or a number. The most common kind of XPath expression is the *path expression* (or *location path*

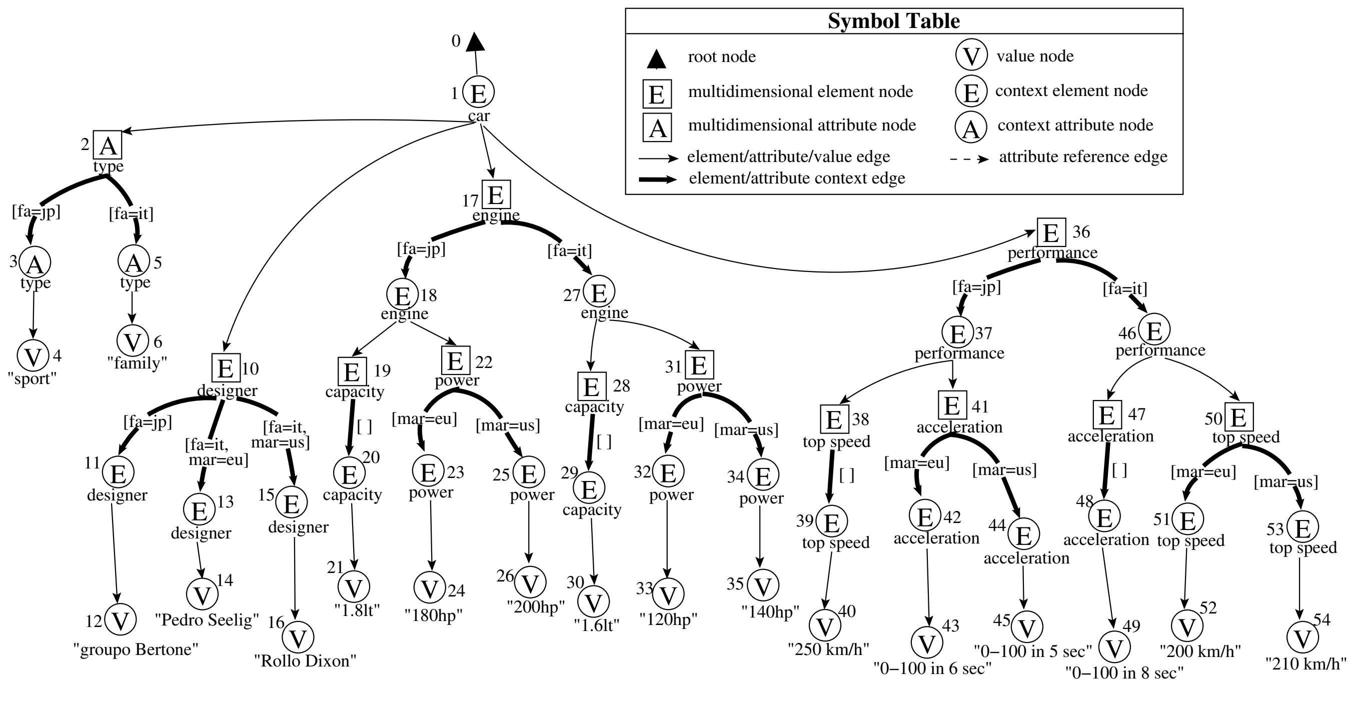

Figure 1: Graphical representation of MXML (MXML-graph)

expression). A path expression is written as a sequence of steps (called *location steps*) to get from one XML node (the current *context node*) to another node or set of nodes. Note that the term "context node" used in XPath refers to the XML tree node considered as the current node for the evaluation of the XPath expression, and has a different meaning than the term "context" when used in MXML and MXPath. The location steps are separated by "/" characters. Using the path expressions, nodes in a XML document may be selected by following the specified location steps. A location step has three components: (a) an *axis* which defines the tree-relationship between the selected nodes and the current node, (b) a *node-test* which identifies a node within an axis, and (c) zero or more *predicates* (to further refine the selected node-set). The syntax of a location step is the following:

```
axisname::nodetest[predicate_1]...[predicate_n]
```

A "/" in front of a XPath denotes the root node of the document. XPath may have an *expanded* or an *abbreviated syntax.*

Example 2. *Consider the following path expression which is written according to the expanded syntax:*

```
child::A/descendant-or-self::node()/child::B
/child::*[position()=1]
```

This expression selects the first element (specified by the predicate "`[position() = 1]`*"), whatever its name (specified by "*`*`*"), that is a child element ("*`child`*" axis) of a* `B` *element that itself is either a child or a child of a descendant (specified by ''*`descendant-or-self::node()`*") of an element* `A` *that is a child of the current context node (as the expression does not begin with a "/"). Notice that in the above "expanded syntax", in each step of the XPath expression, the axis (e.g. "*`child`*" or "*`descendant-or-self`*") is explicitly specified, followed by "*`::`*" and then the node test, such as "A" or "*`node()`*". The above XPath expression can be written in the abbreviated form as follows:*

```
A//B/*[1]
```

In this form, the axis `child` *is omitted, the expression "*`[position()=1]`*" abbreviates to "[1]" and the expression "*`/descendant-or-self::node()/`*" is replaced by "//".*

2.3 Context Specifiers and their properties in MXML

The central notion related to MXML is the notion of *world* defined as follows:

Definition 1. *Let $\mathcal{S}$ be a set of* dimension names *and for each $d \in \mathcal{S}$, let $\mathcal{D}_d$, with $\mathcal{D}_d \neq \emptyset$, be the* domain *of d. A* world *w is a set of pairs (d, u), where $d \in \mathcal{S}$ and $u \in \mathcal{D}_d$ such that for every dimension name in $\mathcal{S}$ there is exactly one element in w. The set of all possible worlds is denoted by $\mathcal{U}$.*

In MXML, 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 $\mathcal{U}$. 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 [16], which 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. The inherited context coverage gives the true context of a node within the frame of a document, taking into account the context of parent and child nodes. 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 [16] which combines two context specifiers and computes a new one representing the union of the worlds specified by the original context specifiers.

EXAMPLE 3. *Figure 2 depicts a fragment of the MXML-graph of Figure 1 annotated with the inherited context coverage (icc) of every node.*

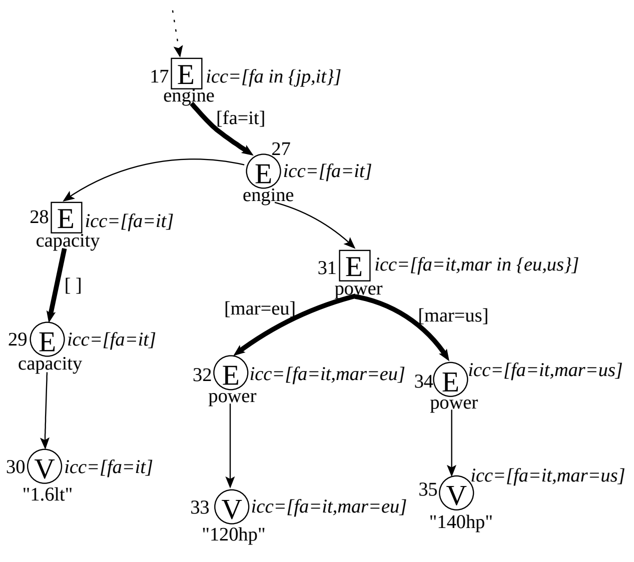

Figure 2: Representing the Inherited Context Coverage

3. MULTIDIMENSIONAL XPATH

In this section we propose *Multidimensional XPath* (MXPath) as an extension of XPath used to navigate through MXML-graphs. In addition to the conventional XPath functionality, MXPath uses the inherited context coverage and the explicit context of MXML in order to select nodes in the MXML document. Similarly to XPath, MXPath uses *path expressions* as a sequence of steps to get from one MXML node to another node, or set of nodes.

In a MXPath, selection criteria concerning the explicit context are expressed through *explicit context qualifiers*. Selection criteria concerning the inherited context coverage are expressed through the *inherited context coverage qualifier*, which is placed at the beginning of the expression.

3.1 MXPath Syntax

An *MXPath expression* contains an *inherited context coverage qualifier* (or *icc qualifier* for short) followed by the *MXPath expression body*. The inherited context coverage qualifier is placed at the beginning of the expression and filters the resulting nodes according to their inherited context coverage. The syntax of an MXPath expression is:

```
[inherited_context_coverage_qualifier],
        MXPath_expression_body
```

An MXPath expression may return either multidimensional nodes or context nodes. In what follows we brake down MXPath expressions, and specify each part separately.

3.1.1 Inherited context coverage qualifier

The syntax of the *inherited context coverage qualifier* is:

```
icc() comparison_op context_specifier_expression
```

where `comparison_op` is one of the operators `=`, `!=`, `<`, `>`, `<=`, or `>=`. Note that it is easy to prove that for the inherited context coverages of the nodes in a path $r, n_1, \ldots, n_k$, from the root r of the MXML tree to a node n_k, it holds that $icc(n_k) \subseteq icc(n_{k-1}) \subseteq \cdots \subseteq icc(r)$. Thus $icc(n_k)$ denotes the worlds under which the complete path holds. The function `icc()` returns the icc of the current node, and, consequently of the currently evaluated path in MXML. This icc is then compared against the *context_specifier*, according to the comparison operator. The operator `=` tests for equality, `<` tests for proper subset, `>` for proper superset, etc. Note that it is actually the sets of worlds represented by the contexts that are compared. In case the comparison returns *false*, the current path is rejected and not considered further. If the inherited context coverage qualifier is omitted in an MXPath expression, the default is implied: `icc() >= "-"`, which evaluates always to *true*.

3.1.2 MXPath expression body

MXPath expression body corresponds to (conventional) XPath expressions. As in XPath, in MXPath we also have two types of expression bodies, namely the *absolute* and the *relative*. An absolute MXPath expression body is a relative one preceded by the symbol "`/`" which denotes the root of the MXML tree. `MXPath_expression_body` is composed by one of more *MXPath steps* separated by "`/`". Thus, the syntax of a relative `MXPath_expression_body` is of the form:

```
MXPath_step_1/MXPath_step_2/.../MXPath_step_n
```

3.1.3 MXPath steps

There are two types of MXPath steps, namely, the *Context MXPath steps* which return context nodes, and the *Multidimensional MXPath steps* which return multidimensional nodes. The syntax of a Context MXPath step is as follows:

```
axis::node_test[pred_1][pred_2]...[pred_n]
```

while the syntax of a Multidimensional MXPath step is as follows:

```
axis->node_test[pred_1][pred_2]...[pred_n]
```

Notice that, both types of MXPath steps contain an *axis*, a *node test* and zero or more *predicates*. The only difference is that in a context MXPath step the axis is followed by the symbol "`::`" which denotes that the step evaluates to context nodes, while in a Multidimensional MXPath step axis is followed by the symbol "`->`" which denotes that the step evaluates to multidimensional nodes.

3.1.4 MXPath predicates

In MXPath a *predicate* consists of an expression, called a *MXPath predicate expression*, enclosed in square brackets. A predicate serves to filter a sequence, retaining some items and discarding others. Multiple predicates are allowed in MXPath expressions. In the case of multiple adjacent predicates, the predicates are applied from left to right, and the result of applying each predicate serves as the input sequence for the following predicate. For each item in the input sequence, the predicate expression is evaluated and a truth value is returned. The items for which the truth value of the predicate is *true* are retained, while those for which the predicate evaluates to *false* are discarded. The operators (logical operators, comparison operators, etc.) used in MXPath predicates are those used in conventional XPath. MXPath predicates may also contain MXPath expression bodies in the same way as XPath expressions are allowed in conventional XPath predicates. Besides these syntactic constructs, *explicit context qualifiers* (or *ec qualifiers*) are also used in MXPath predicates. An ec qualifier may be applied in every step of a MXPath expression and filter the resulting nodes of the corresponding step according to their explicit context. Explicit context qualifiers are of the form:

`ec() comparison_op context_specifier_expression`

The function `ec()` returns the explicit context of the current node. Note that, the predicates assigned to a *context MXPath step* are applied to the context nodes obtained from the evaluation of this step. In the same way, if a MXPath step is a *multidimensional MXPath step*, predicates are applied to the resulting multidimensional nodes.

3.2 MXPath examples

In this section we present a number of MXPath examples and explain their evaluation on the MXML-graph of Figure 1.

Example 4. *Retrieving context nodes according to their inherited context coverage.*

Query: What is the acceleration of a car produced in Japan and sold in USA?

MXPath: `[icc()="factory=Japan,market=USA"],`
`/child::car/child::performance/child::acceleration`

In this example, `[icc()="factory=Japan,market=USA"]` *is the icc qualifier of the MXPath expression. The expression that follows is the MXPath expression body which, in this example, is syntactically similar to a conventional XPath expression. However, there is difference as in MXPath a step describes paths that include two MXML nodes: one multidimensional node followed by one corresponding context node. Recall that, in MXML context elements have the same element name as the corresponding multidimensional element. Consider for example the step* `child::performance` *of our MXPath expression and suppose that the current node on which this step is evaluated is node 1 (obtained by evaluating the previous step). This step looks for children of node 1, crosses over the multidimensional node labeled "*`performance`*" with ID 36 and returns the context nodes labeled "*`performance`*" with IDs 37 and 46. Similarly, the MXPath step* `child::acceleration` *evaluates to the context nodes labeled "*`acceleration`*". In particular, when the step is evaluated by considering as current node the node with ID 37, we get the nodes with IDs 42 and 44. When the same step is evaluated on the node with ID 46, we get the node with ID 48. Now, the* icc qualifier *is applied. As the inherited context coverage of the results must be equal to the context* `[factory=Japan,market=USA]`*, the nodes with IDs 42 and 48 are discarded and the final result is the node with ID 44.*

Example 5. *Retrieving multidimensional nodes according to their inherited context coverage.*

Query: What is the power of the cars which are produced in Italy and sold either in USA or in Europe?

MXPath:
`[icc()="factory=Italy, market in {USA, Europe}"],`
`/child::car/child::engine/child– >power`

In this case, we follow the same navigation rules as in Example 4, but now the MXPath step `child– >power` *is a multidimensional one and thus it returns the multidimensional nodes labeled "*`power`*" with IDs 22 and 31. Then, because of the application of the icc qualifier* `[icc()="factory=Italy, market in {USA, Europe}"]`*, node 22 is discarded and we get node 31 as the result.*

Example 6. *Retrieving context nodes according to their explicit context.*

Query: What is the acceleration of the cars which are sold in Europe?

MXPath:
`[icc()>="-"], /child::car/child::performance`
`/child::acceleration[ec()>="market=Europe"]`

Here, the icc qualifier denotes that the icc of the results must be context superset of the empty context `[-]`*. But as this is always true, this icc qualifier can be omitted (in which case it is implied). The predicate* `[ec()>="market=Europe"]` *is an ec qualifier stating that the set of worlds expressed by the explicit context of the "*`acceleration`*" context nodes must be superset of the set of worlds expressed by the context specifier* `[market=Europe]`*. Thus, although the evaluation of the step* `child::acceleration` *returns the nodes 42, 44 and 48, the final result, after applying the predicate, contains only the context nodes 42 and 48.*

Example 7. *Retrieving multidimensional nodes according to their explicit context.*

Query: What is the power of a car produced in Japan?

MXPath:
`/child::car/child::engine[ec()="factory=Japan"]`
`/child– >power`

As in Example 5, this query returns multidimensional nodes labeled "`power`*". Here because of the ec qualifier in the step* `child::engine[ec()="factory=Japan"]`*, this step returns the context node with ID 18. So, the final result is node 22.*

Example 8. *Retrieving value nodes.*

Query: From those cars that are produced in Italy, what is the top speed of the cars which are sold either in Europe or in USA?

MXPath: `[icc()<="market in {Europe,USA}"],`
`/child::car/child::performance`
`[ec()="factory=Italy"]/child::top_speed`

Here both types of qualifiers are used. Using the navigation rules explained above, we get the nodes with IDs 51 and 53.

Example 9. *Using an MXPath expression in a predicate.*

Query: Which is the top speed of the cars available in the market of USA whose acceleration is "0-100 in 5 sec"?

***MXPath**: `[icc()>="-"],`*
`/child::car/child::performance[child::acceleration`
`[ec()<="market=USA"] ="0-100 in 5 sec"]`
`/child::top_speed`
Here a whole MXPath expression body is contained in the predicate of the second MXPath step. The predicate is:
`[child::acceleration[ec() <= "market = USA"] = "0-100 in 5 sec"]`
and is responsible for retaining the nodes (if such nodes exist) with element name `performance` *which have a child element named* `acceleration` *whose value for the market of* `USA` *is "*`0-100 in 5 sec`*". We can see that by evaluating the above MXPath expression we get the node whose ID is 39.*

4. ON THE SEMANTICS OF MXPATH

In this section we discuss semantic issues of MXPath. In particular, based on the notion of reduction for MXML documents proposed in [11], we show how the semantics of (a fragment of) MXPath can be expressed with respect to (the semantics of) XPath.

4.1 Reduction of MXML to XML

Reduction is a procedure which, given a world w, and a MXML document (MXML-graph) G, produces a conventional XML document (XML-Graph) $G' = \mathcal{R}(w, G)$ which is the holding instance of G under w. The intuition behind reduction is that, given a world w, we can specialize the MXML document G by eliminating its parts that do not hold under w (in other words, by eliminating the parts of the document for which w does not belong to the worlds specified by their inherited context coverage), obtaining in this way a conventional XML document G' which holds under w. The *reduction procedure* consists of the following steps:

1. 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.
2. Eliminate each element context edge (resp. attribute context edge) (p, C, q) of the graph G_1, obtained from G in Step 1, as follows: Let (s, p) be the element edge (resp. attribute edge) leading to the node p. Then:
   - Add a new element edge (resp. attribute edge) (s, q), and
   - discard the edges (p, C, q) and (s, p) and the node p.

The XML document (XML-graph) G', obtained after applying Step 2, is the result of the reduction procedure applied on the MXML document G with respect to the world w.

Example 10. *Consider the MXML document of Example 1 and the world* $w = \{(factory, Japan), (market, USA)\}$. *By applying the reduction procedure to this MXML document we obtain the following XML document:*

```
<car type="sport">
   <designer>groupo Bertone</designer>
   <engine>
      <capacity>1.8lt</capacity>
      <power>200hp</power>
   </engine>
   <performance>
      <top_speed>250km/h</top_speed>
      <acceleration>0-100 in 5sec</acceleration>
   </performance>
</car>
```

The XML-graph of this document is shown in Figure 3.

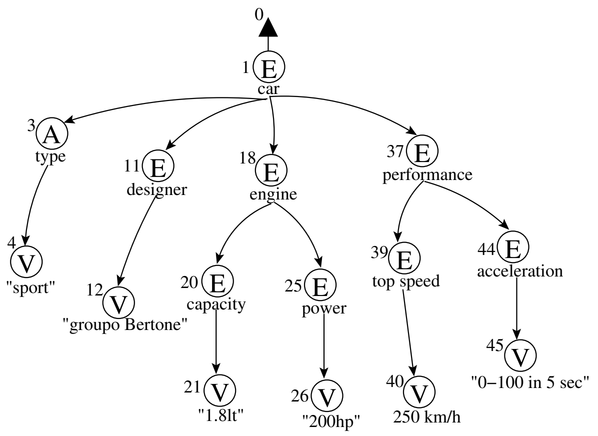

Figure 3: XML-graph obtained by applying reduction to the MXML-graph of Figure 1.

Notice that, by applying the reduction procedure, each MXML document can be reduced to a number of XML documents, each of them holding under a single world. We can thus consider an MXML document as a compact representation of a set of XML documents. However, an MXML document is more than a set of XML documents as it relates the various components of these documents (as being different versions of the same object).

4.2 Semantic relation between MXPath and XPath

In this subsection we investigate the relation between MXPath and XPath and show how we can establish a semantic relation between MXPath and XPath. Example 11 gives some intuition about the content of the remaining section:

Example 11. *In Example 4, we have seen the following MXPath query:*
`[icc()="factory=Japan,market=USA"],`
`/child::car/child::performance/child::acceleration`
and showed that the result of evaluating this query is the node with ID 44. Notice that the nodes obtained by evaluating the MXPath expression body of the query (which are the nodes with IDs 42, 44 and 46) are filtered using the icc qualifier in order to obtain the final result. Observing the icc qualifier, we see that the nodes retained in the final results are those nodes whose icc consists of the world `w = {(factory,Japan), (market,USA)}`*. Notice now that in Example 10, we have applied the reduction procedure to the MXML document with respect to the same world w and we obtained the XML document whose graphical representation is shown in Figure 3). It is interesting to observe that by treating the MXPath expression body of our MXPath expression as a conventional XPath expression*[1] *and applying this expression to the XML-graph of Figure 3, we get the same result (i.e. node 44) as with the original MXPath query.*

Based on this example an interesting question arizes: *Can we use the conventional XML documents obtained by applying* reduction *on an MXML document G together with a*

[1]Here the MXPath expression body is a conventional XPath expression. However, in the general case this is not true.

conventional XPath query obtained from the original MXPath query E, in order to define the meaning of applying the query E on the document G? The following Lemma 1 answers this question for a subclass of MXPath queries. More specifically, Lemma 1 establishes a semantic relation for a subclass of MXPath queries, namely for queries whose MXPath expression body contains neither explicit context qualifiers nor Multidimensional MXpath Steps.

In Lemma 1 we use the following notation: Let E be an MXPath expression (resp. XPath expression) and G be a MXML document (resp. XML document). Then by $E(G)$ we denote the set of the node IDs returned by evaluating E over G. By $\mathcal{U}_G$ we denote the set of all possible worlds related to G. Also, $\mathcal{W}(C)$ denotes the set of the worlds specified by the context specifier C. Finally, recall that $\mathcal{R}(w, G)$ denotes the XML document obtained by applying the reduction procedure on G with respect to a world $w \in \mathcal{U}_G$.

Lemma 1. *Let G be a MXML document and $E = (I, P)$ be a MXPath query, where I is its icc qualifier and P be its MXPath expression body. Suppose that P consists only of Context MXPath steps and does not contain ec qualifiers*[2]. *Suppose also that C is a context specifier such that $C \neq$ "-". Then the following hold:*

- *If I is of the form $[icc() >$ "-"$]$, then:*
  $E(G) = \bigcup_{G' \in S_{\mathcal{U}}} (P(G'))$,
  where $S_{\mathcal{U}} = \{\mathcal{R}(w, G) | w \in \mathcal{U}_G\}$.
- *If I is of the form $[icc() >= C]$, where C is a context specifier, then:*
  $E(G) = \bigcap_{G' \in S_C} (P(G'))$,
  where $S_C = \{\mathcal{R}(w, G) | w \in \mathcal{W}(C)\}$.
- *If I is of the form $[icc() = C]$, where C is a context specifier, then:*
  $E(G) = \bigcap_{G' \in S_C} (P(G')) - \bigcup_{G'' \in S_{\overline{C}}} (P(G''))$,
  where $S_C = \{\mathcal{R}(w, G) | w \in \mathcal{W}(C)\}$ *and*
  $S_{\overline{C}} = \{\mathcal{R}(w, G) | w \in \mathcal{U}_G - \mathcal{W}(C)\}$.
- *If I is of the form $[icc() <= C]$, where C is a context specifier, then:*
  $E(G) = \bigcup_{G' \in S_C} (P(G')) - \bigcup_{G'' \in S_{\overline{C}}} (P(G''))$,
  where $S_C = \{\mathcal{R}(w, G) | w \in \mathcal{W}(C)\}$ *and*
  $S_{\overline{C}} = \{\mathcal{R}(w, G) | w \in \mathcal{U}_G - \mathcal{W}(C)\}$.
- *If I is of the form $[icc()! = C]$, where C is a context specifier, then:*
  $E(G) = \bigcup_{G' \in S_{\mathcal{U}}} (P(G')) -$
  $(\bigcap_{G'' \in S_C} (P(G'')) - \bigcup_{G''' \in S_{\overline{C}}} (P(G''')))$,
  where $S_{\mathcal{U}} = \{\mathcal{R}(w, G) | w \in \mathcal{U}_G\}$ *and* $S_C = \{\mathcal{R}(w, G) | w \in \mathcal{W}(C)\}$ *and* $S_{\overline{C}} = \{\mathcal{R}(w, G) | w \in \mathcal{U}_G - \mathcal{W}(C)\}$.
- *If I is of the form $[icc() > C]$, where C is a context specifier, then:*
  $E(G) = (\bigcap_{G' \in S_C} (P(G'))) \cap (\bigcup_{G'' \in S_{\overline{C}}} (P(G'')))$,
  where $S_C = \{\mathcal{R}(w, G) | w \in \mathcal{W}(C)\}$ *and*
  $S_{\overline{C}} = \{\mathcal{R}(w, G) | w \in \mathcal{U}_G - \mathcal{W}(C)\}$.
- *If I is of the form $[icc() < C]$, where C is a context specifier, then:*
  $E(G) = ((\bigcup_{G' \in S_C} (P(G')) - \bigcup_{G'' \in S_{\overline{C}}} (P(G''))) -$
  $(\bigcap_{G' \in S_C} (P(G')) - \bigcup_{G'' \in S_{\overline{C}}} (P(G''))))$.
  where $S_C = \{\mathcal{R}(w, G) | w \in \mathcal{W}(C)\}$ *and*
  $S_{\overline{C}} = \{\mathcal{R}(w, G) | w \in \mathcal{U}_G - \mathcal{W}(C)\}$.

[2]In this case P is a (conventional) XPath expression.

Lemma 1 implies that we can get the same result nodes by evaluating XPath on XML and by evaluating corresponding MXPath on MXML. Note, however, that although these nodes are identical (have the same IDs) in both situations, they represent different subtrees depending on which XML they come from. It is easy to see that, when a MXML query evaluated on a MXML document G returns a (context) node N, then the occurrences of the same node N returned by applying the MXPath expression body to the instances of G as it is specified by the above lemma, are rooted in XML trees obtained by applying the reduction procedure to the multidimensional subtree of G rooted at N. In this sense, both the MXML document and the MXPath expression can be seen as compact representations of a family of XML documents and a family of corresponding XPath queries on them. Moreover, in order for some of these XPath queries to be answered, several XML documents should be examined in parallel.

As implied from Lemma 1, a single MXPath is expressive enough to specify nodes that correspond to the combination of the results of many XPath variants posed on relevant MXML reduction results. However, MXPath is in fact more than a compact way to express what can be expressed by XPath. Some features of MXPath are inherently multidimensional and can only be explained in relation to MXML. Those features exploit the fact that MXML groups entity facets (context nodes) under the same multidimensional node. Using Multidimensional MXPath steps (steps containing the `->` symbol) we can specify multidimensional nodes and using ec qualifiers we can navigate through their facets. However, this kind of queries seems difficult to be expressed in terms of the XML documents obtained by applying reduction to G.

5. MANIPULATING MXML DOCUMENTS WITH MXPATH

The definition of MXPath is the first step towards manipulating MXML. The use of MXPath can be threefold:

A. MXPath is necessary for updating MXML. In [8] we have defined a number of basic change operations for MXML. Those operations use MXPath expressions as input in order to specify the affected parts in the MXML tree.

B. MXPath will be an essential part of a query language for MXML. XQuery [20] is a language for querying and manipulating XML documents. XQuery uses XPath expressions to navigate through elements in an XML document but adds more functionality to become a full-fledged query language, supporting for example variables, joins and construction of results. Both XQuery and XPath share the same data model and support the same functions and operators. An extension of XQuery (Multidimensional XQuery) could express multidimensional queries over MXML documents, using MXPath expressions. Such a query language would treat context as first class citizen, allowing the expression of context-aware queries.

C. MXPath is useful for transforming MXML. The *Extensible Stylesheet Language* (XSLT) [18], a language for transforming XML documents, makes use of XPath for selecting elements. In the transformation process, XSLT uses XPath expressions to define parts of the source document that should match one or more predefined patterns. When a match is found, XSLT transforms the matching part of the source document into the result document. XPath expressions are used in XSLT for a variety of purposes including (a) selecting nodes for processing, (b) specifying conditions for different ways of processing a node or (c) generating text to be inserted in the result tree. XSLT could be extended through MXPath (to become Multidimensional XSLT) so as to be capable of expressing transformations of MXML documents.

6. DISCUSSION AND FUTURE WORK

In this paper we investigated the problem of navigating and querying MXML documents. For this we propose a Multidimensional extension of XPath, called *Multidimensional XPath* (or MXPath). 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 [7]. In [8, 9], we showed that expressions belonging to a fragment of MXPath are necessary in order to define a set of basic change operations at the level of the MXML-graph.

Our future work will focus on (a) the definition of algorithms for the translation of MXPath queries to SQL queries so as to be evaluated on MXML documents stored in relational databases, (b) the incorporation of MXPath into query languages such as XQuery and XSLT, and (c) the formal definition of the semantics of the complete MXPath.

7. REFERENCES

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, 2000.

[2] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel Query Language for Semistructured Data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.

[3] T. Amagasa, M. Yoshikawa, and S. Uemura. A Data Model for Temporal XML Documents. In *Proc. of DEXA 2000, LNCS vol. 1873*, pages 334–344. Springer, 2000.

[4] A. Bonifati and S. Ceri. Comparative Analysis of Five XML Query Languages. *SIGMOD Record*, 29(1):68–79, 2000.

[5] C. Doulkeridis and M. Vazirgiannis. Querying and updating a context-aware service directory in mobile environments. In *Proc. of 2004 IEEE/WIC/ACM Int. Conf. on Web Intelligence (WI 2004)*, pages 562–565. IEEE Computer Society, 2004.

[6] C. E. Dyreson. Observing Transaction-Time Semantics with TTXPath. In *Proc. of WISE 2001*, pages 193–202, IEEE Computer Society 2001.

[7] N. Fousteris, M. Gergatsoulis, and Y. Stavrakas. Storing Multidimensional XML documents in Relational Databases. In *Proc. of DEXA 2007, LNCS vol. 4653*, pages 23–33. Springer, 2007.

[8] N. Fousteris, M. Gergatsoulis, and Y. Stavrakas. Updating Multidimensional XML Documents. In *Proc. of iiWAS'2007*, pages 257–266, 2007.

[9] N. Fousteris, M. Gergatsoulis, and Y. Stavrakas. Updating multidimensional XML documents. *International Journal of Web Information Systems*, 4(2):142–164, 2008.

[10] M. Gergatsoulis and Y. Stavrakas. Representing Changes in XML Documents using Dimensions. In *Proc. of XSym 2003, LNCS vol. 2824*, pages 208–222. Springer, 2003.

[11] M. Gergatsoulis, Y. Stavrakas, and D. Karteris. Incorporating Dimensions in XML and DTD. In *Proc. of DEXA 2001, LNCS vol. 2113*, pages 646–656. Springer, 2001.

[12] 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 2001, LNCS vol. 2151*, pages 352–365. Springer, 2001.

[13] S. Groppe and S. Bottcher. Query Reformulation for the XML standards XPath, XQuery and XSLT. In *Berliner XML Tage 2004, 11.-13. October 2004 in Berlin*, pages 53–64, 2004.

[14] M. C. Norrie and A. Palinginis. Versions for Context Dependent Information Services. In *Proc. of CoopIS' 03, LNCS vol. 2888*, pages 503–515. Springer, 2003.

[15] J. M. Pérez, M. J. A. Cabo, and R. B. Llavori. XRL: A XML-Based Query Language for Advanced Services in Digital Libraries. In *Proc. of DEXA 2002, LNCS vol. 2453*, pages 300–309, 2002.

[16] Y. Stavrakas and M. Gergatsoulis. Multidimensional Semistructured Data: Representing Context-Dependent Information on the Web. In *Proc. of CAiSE'02, LNCS vol. 2348*, pages 183–199, Springer, 2002.

[17] Y. Stavrakas, M. Gergatsoulis, C. Doulkeridis, and V. Zafeiris. Representing and Querying Histories of Semistructured Databases Using Multidimensional OEM. *Information Systems*, 29(6):461–482, 2004.

[18] W3C CONSORTIUM. XSL Transformations (XSLT). http://www.w3.org/TR/xslt, November 1999.

[19] W3C CONSORTIUM. Extensible Markup Language (XML). http://www.w3.org/XML, May 2007.

[20] W3C CONSORTIUM. W3C XML Query (XQuery). http://www.w3.org/XML/Query/, October 2008.

[21] W3C CONSORTIUM. XML Path Language (XPath) 2.0. http://www.w3.org/TR/xpath20/, January 2007.

[22] 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.

[23] S. Zhang and C. E. Dyreson. Adding Valid Time to XPath. In *Proc. of DNIS 2002, LNCS 2544*, pages 29–42, Springer, 2002.