XML Query Languages

Markus Mårtensson
Department of Computing Science
Umeå University, S901 87 Umeå, Sweden
e-mail: c96mmn@cs.umu.se
phone: +46 (90) 194043 fax: +46 (90) 786-6126
URL: http://www.acc.umu.se/~mortis

Abstract

XML, the eXtensible Markup Language, needs a uniform query language to allow information extraction from XML repositories. There are many language candidates and this paper makes an evaluation of five of them, namely the XML-QL, Lorel, XML-GL, XSL and XQL. The survey is example driven and focused on a list of criteria. Lorel scores the highest points and is declared winner pending an official standard from the W3C.

1 Introduction

In recent years, the XML (eXtensible Markup Language) [TBSM98] has become the de facto standard for storing and exchanging structured data, especially when used in combination with the web.

Structured data, as opposed to flat text, allows for advanced searching, very much like average relational databases. Instead of searching for a single word in a text and returning the lines closest to the match the computer can be instructed to retrieve exactly the information the user requires.

Part of the success of the relational database model can be attributed to the uniform query language SQL. One powerful language allows users ranging from novice to expert to extract the information they need.

The problem at hand is not the lack of XML query languages, but the lack of a uniform standard. The aim of this paper is to evaluate the current candidates for the “SQL of XML”.

69
2 Background

2.1 Brief Introduction to the XML

XML is a tool for storing and exchanging structured data. The language very much resembles the HTML which presently powers the World Wide Web. The reason for this is that XML is a subset of SGML [Int86], the generalized markup language from which HTML is derived.

Example 2.1.1 Running example of XML document (fragment)

```
...<company>
  <employee>
    <ename fname="John" minit="B" lname="Smith"/>
    <ssn>123456789</ssn>
    <bdate>1965-01-09</bdate>
    <address>731 Fondren, Houston, TX</address>
    <dno>5</dno>
    <dependent relation="son">Michael</dependent>
    <dependent relation="daughter">Alice</dependent>
    <dependent relation="spouse">Elizabeth</dependent>
  </employee>
  <employee>
    <ename fname="Franklin" minit="T" lname="Wong"/>
    <ssn>333445555</ssn>
    <bdate>1955-12-08</bdate>
    <address>638 Voss, Houston, TX</address>
    <dno>5</dno>
    <dependent relation="daughter">Alice</dependent>
    <dependent relation="son">Theodore</dependent>
    <dependent relation="spouse">Joy</dependent>
  </employee>
  <employee>
    <ename fname="Alicia" minit="J" lname="Zelaya"/>
    <ssn>999887777</ssn>
    <bdate>1968-07-19</bdate>
    <address>3321 Castle, Spring, TX</address>
    <dno>4</dno>
  </employee>
  <department>
    <dname>Research</dname>
    <dnumber>5</dnumber>
    <dmgrssn>333445555</dmgrssn>
  </department>
...<company>
```
Example 2.1.1 is an excerpt from a company database [EN00] projected into XML. It will serve as an example of what XML may look like, as well as being the basis for various query examples in this paper.

XML documents are basically composed of a hierarchy of elements bounded by tags. Tags can be divided into two categories; those with content and those without. The former category, see for example employee, begins with tagname and ends with tagname. Anything in between is content. The other group, represented by ename is written as tagname. Tags may also have attributes, as demonstrated by ename.

As the XML in itself is not a major issue as far as this paper is concerned, the reader is instructed to find further information in XML literature such as [GP00].

2.2 Query Basics

The purpose of a query language is to allow a user to extract information from data repositories and restructure the information to match the user’s needs [CCD99]. This section is a brief summary of query theory, where the key terms used in the survey will be explained.

The most common query element is perhaps the selection in which predicates narrow down the number of elements by imposing a condition which must be fulfilled. Say, for example, that we only want to list employees born in 1965. This would exclude all employees of Example 2.1.1 except John Smith.

Not all information available for employees may be desirable in the result. If this is the case, then we will project out only the sub-elements we want. Name and social security number would perhaps be sufficient for some applications.

If we want a predicate to include conditions on, or the result to include information from, two different elements, then these two need to be joined first. If, for instance, we want to return pairs of names of employees and departments, we need to join those elements with the join-condition that the employee and the department has matching department numbers (i.e. dno = dnumber).

The information obtained from the select-project-join can be transformed by adding additional information, or by restructuring the data in any number of ways. One example would be to convert an XML document into the HTML.

Some query languages support the concepts of grouping and aggregation. Employees could be grouped by their departments to create a tables with the department names as headings, followed by the associated personnel. A typical aggregation on such a group would be to count the number of member elements, but it could also include various arithmetic operations on the contents.

Further information about query and database theory can, for example, be found in [EN00].
3 Method

The XML Query Working Group has compiled a set of criteria for the designing of a query language [PFR00]. Listed in Table 3.1 are the requirements addressed in this survey, including my own and some from other sources [CCD+99].

<table>
<thead>
<tr>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
</tr>
<tr>
<td>R2</td>
</tr>
<tr>
<td>R3</td>
</tr>
<tr>
<td>R4</td>
</tr>
<tr>
<td>R5</td>
</tr>
<tr>
<td>R6</td>
</tr>
<tr>
<td>R7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Desiderata</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
</tr>
<tr>
<td>D2</td>
</tr>
<tr>
<td>D3</td>
</tr>
</tbody>
</table>

Table 3.1: Query Language Requirements

As information wanted is often spread over several documents $^{R1}$, the language must allow queries spanning multiple sources. There must also be available a means of combining $^{R2}$ related elements as described in section 2.2, as well as aggregation $^{R3}$ and transformation $^{R4}$ which are also covered there. For advanced composite queries set operations $^{R6}$ are also required.

The user effort needed for learning and using the language should be minimal $^{R5}$. Queries should be declarative $^{R7}$, as opposed to procedural. They should not describe how to find the information, only what information is wanted.

For user convenience, it is recommended that the language include support for regular expressions on paths $^{D1}$ and nested queries $^{D3}$, although it is possible to formulate sufficient queries without them. Many applications require a means for making updates $^{D2}$, and choosing to leave out this feature renders the language unusable in these areas.

In the next few sections, I will briefly describe each of the language proposals and then a concluding discussion will follow.
4 Survey

4.1 XML-QL

XML-QL [DFF+99] is a query language proposed by the AT&T\(^1\). It combines the SQL with modern research done on semi-structured data.

**Example 4.1.1** Simple XML-QL query

```xml
WHERE <employee>
    <ename lname="Wong"/>
    <ssn>$s</>
</> IN "company.xml"
CONSTRUCT <result>
    <ssn>$s</>
</>
```

As Example 4.1.1 shows, variables (in this case \$s\) are used to transfer data from the where-clause to the construct-clause. Using the same variable more than once requires that the same information is matched on all of the locations.

**Example 4.1.2** XML-QL joins

```xml
WHERE <employee>
    <ename lname=$l/>
    <dno>$d</>
</> IN "company.xml",
    <department>
        <dname>$n</>
        <dnumber>$d</>
    </> IN "company.xml"
CONSTRUCT <result>
    <lastname>$l</>
    <dptname>$n</>
</>
```

Example 4.1.2 performs a transformation\(^4\) from the original XML document into a document of result elements containing lastname and dptname subelements.

Joins\(^2\) are expressed either by multiple comma-separated sources in the where-clause, as shown in the example, or using a combination of Skolem functions and sub-blocks.

In Example 4.1.2 there are two in-clauses, both pointing to the same file. As this need not be the case, cross-document queries\(^1\) are fully supported.


\(^4\)This reference is not included in the text and is not the standard notation.
Supported set operations\textsuperscript{R6} are union and intersection. Other features include nested queries\textsuperscript{D3} and predicates on tag ordering. Support for grouping and aggregation\textsuperscript{R3} is planned.

### 4.2 Lorel

Lorel [AQM+96] is query language developed at Stanford University\textsuperscript{2}. Originally designed to query semi-structured data, it has recently been adapted to query XML.

There is a striking resemblance between Lorel and the SQL, both in terms of syntax and in power. The select-from-where triplet of SQL exists, for example, also in Lorel:

**Example 4.2.1 Simple Lorel query**

```plaintext
select N.@name, N.@fname
from company.employee(E).ename N
where E.ssn = "123456789"
```

Example 4.2.1 above extracts the last and first names of the company employee having the social security number 12345678. The query sets two shorthand names N and E to \texttt{company.employee.ename} and \texttt{company.employee}, respectively. The dots separate subelements or attributes from their elements.

**Example 4.2.2 Lorel joins**

```plaintext
select N.@name, D.@name
from company.employee(E).ename N, company.department D
where E.dno = D.dnumber
```

As Example 4.2.2 reflects, joins\textsuperscript{R2} are expressed in an SQL-like fashion with an explicit join condition. Joins may span multiple documents\textsuperscript{R1}.

**Example 4.2.3 Lorel updates**

```plaintext
update E.dno := D.dnumber
from company.employee E, company.department D
where E.ename.@name="Zelaya" and D.dname = "Research"
```

Lorel supports updates\textsuperscript{D2} as can be seen in Example 4.2.3. The query reassigns Mrs. Zelaya to the Research department.

Type coercion, i.e. implicit data casting, is supported in Lorel, allowing easy object comparison. Other features include a complete collection of set operations\textsuperscript{R6}, nested queries, and advanced aggregation\textsuperscript{R3}.

\textsuperscript{2}A prototype is available at \url{http://www-db.stanford.edu/lore}

74
XML-GL is a graphical query language, relying on a visual representation of queries rather than a purely textual approach \cite{CCD99}.

**Example 4.3.1 Simple XML-GL query**

![XML-GL Diagram]

The left side of Example 4.3.1 selects all employees belonging to department number five and having one dependent named Alice. The right side can be used for projection but in this case all information for matched employees is returned.

Uniform Resource Identifiers (URI) can be explicitly specified on the left side as can be seen from the example. This allows information from multiple documents $R_1$ to be combined.

Joins $R_2$ are expressed by pointing edges from different elements at a single node, labeling the edges using the names of the attribute or data participating in the join. Updates $R_2$ are visualized by labeling arrows with the initials of insert, update or delete.

### 4.4 XSL

The XSL (eXtensible Stylesheet Language) \cite{ABC00} can be divided into two parts: the XSLT transformation language \cite{Cla99b} and the formatting objects. The latter is of very little interest to this paper, whereas the XSLT does meet some of the requirements listed in Table 3.1.

Selection is performed with the XPath language \cite{Cla99a} described in section 4.4.1. This allows a number of the requirements stated in Table 3.1 to be met.

The above examples would create an HTML heading for each department and present a bulleted list of employees working there. This demonstrates a join $R_2$ and also grouping. The total number of employees per department could have been calculated with the aggregate function $R_3$ \texttt{count()}, but that is left as an exercise for the reader.
Example 4.4.1 Template rules for the company document

```xml
<xsl:template match="department">
  <html:h2><xsl:value-of select="dbname"/>/&gt;&lt;/html:h2>
  <html:ul>
    <xsl:apply-templates select="employee[dno=self::dnumber]"/>
  </html:ul>
</xsl:template>
```

4.4.1 XPath

XPath [Cla99a] uses a UNIX-like path syntax. The XML document is viewed as a tree, with root and nodes. Example 4.4.2 begins at the root (/) and finds all mydoc elements. From all of the info subelements are then selected all descendants (arbitrary depth) using the // operator. All name elements within this set are then sought out and the parent (. ) elements are returned as the path result.

Example 4.4.2 Simple XPath expression

```
/mydoc/info/name/..
```

Predicates and references are demonstrated in Example 4.4.3.

Example 4.4.3 Simple XPath expression

```
/company/employee[name=@name='Smith']/id(dno)/dbname
```

This query finds the department name of employee Smith, if used with the XML document of Example 2.1.1. Predicates are written as [predicate] and element attributes are prefixed by @. The function id('xyz') finds the element which has the ID xyz.

XPath supports the union and negation set operations.

4.5 XQL

The XQL (XML Query Language) [JRS98] was jointly developed by Microsoft, Texcel and webMethods Inc.

The language extends the pattern syntax of the XSL with a few more Boolean comparison operators, the intersection set operation as well as return and sequence operators.

---

There will be just one as the XML allows only one root element.
All XPath expressions, covered in Section 4.4.1, are also valid in XQL. Focus of this coverage is therefore concentrated on the extensions made.

**Example 4.5.1 XQL return operator**

```xml
employee?[@name]/dependant[@relation="spouse"]
```

Example 4.5.1 finds married employees and returns their names and the names of their spouses.

XQL does not support transformations\textsuperscript{R4}, nor are joins\textsuperscript{R2} an included feature. It is, however, possible to perform so-called semi-joins. The lack of features in XQL is both due to, and excused by, its size. The queries are small enough to be embedded into URIs.

It has been suggested [DSR98], that the query power of XQL be combined with the transformation strength of XSL.

### 4.6 Other languages

Among the languages not addressed by this paper are WebSQL/WebOQL, XMAS algebra, SXQL, Xtract, NQL, and XRS.
5 Discussion

For those looking for a language resembling both the power and feel of SQL, Lorel is the place to go. XML-QL is a strong competitor, but as it cannot perform updates, this market share is probably lost.

XML-GL has an expressional power comparable to that of Lorel and XML-QL, but is not likely to please the SQL crowd. It is, however, fairly intuitive and with the right tools, it may find its uses.

For maximum control over transformation XSL is recommended. It is, however, too verbose to be used for interactive queries, an area of use where Lorel and XQL are better suited. Not being able to operate on multiple documents also limits the uses of the language.

XQL on its own provides very little functionality, but is easy to learn and use. The real strength of the XQL lies in its embeddability; none of its competitors can appear encapsulated in URIs.

In Table 5.1 below, there is a requirements matrix showing which languages have support for what features.

<table>
<thead>
<tr>
<th>Requirements</th>
<th>XML-QL</th>
<th>Lorel</th>
<th>XML-GL</th>
<th>XSL</th>
<th>XQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>R2</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>R3</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>R4</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>R5</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>R6</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>R7</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>D1</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D2</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D3</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- ● Fully meets the requirement
- ○ Meets most of the requirement

Table 5.1: Requirements Matrix

The quest for the uniform XML query language standard is an ongoing effort lead by the XML Query Working Group. It is possible that in a near future, a proposal to end all proposals will be put forward. Until then, Lorel is the best choice for most applications.
References


