

XML INTERFACE FOR OBJECT ORIENTED DATABASES

Frank vanLingen

European Organization for Nuclear Research, CH-1211 Geneva 23, Switzerland, fvlingen@cern.ch

Supported by Eindhoven University of Technology and University of the West of England

Key words: XML, Object Oriented Databases, XML Querying, Serialisation

Abstract: *Within CERN (European Organization for Nuclear Research) there are numerous heterogenous data sources (different in format, type and structure). Several of these sources will be flat files or data sources other than databases. To create a common interface to these sources, we decided to use XML (eXtended Markup Language) since XML is becoming the de facto standard for data exchange/integration. A significant part of CERN data is stored within object oriented databases. This document describes a mechanism to access object-oriented databases as an XML document by using serialization on demand and rewriting XML queries to OQL queries. To accomplish this the Xpath specification is extended and mapped on the object oriented paradigm.*

1. INTRODUCTION

At CERN [CERN], the European laboratory for particle physics, the fundamental structure of matter is studied using particle accelerators. The study of matter with accelerators is part of the field of high-energy physics (HEP). The CMS (Compact Muon Solenoid) [CMS] detector is one of the two general-purpose detectors of the LHC (Large Hadron Collider) [LHC] accelerator. Because of the autonomous approach in design and construction, and the different lifecycle phases, there are numerous types of data sources, and many different interfaces to these sources. Retrieving data from multiple sources requires knowledge about the interfaces and query dialects. This process would be simplified if there were a uniform interface of accessing the data sources.

Open Database Connectivity (ODBC) [Geiger 1995] is such a uniform interface. It is a widely accepted application programming interface (API) for database access. However, ODBC relates to relational databases, rather than data sources in general. At this moment data sources at CERN are relational databases, object oriented databases, and structured flat files. But this could be extended in the future with simulation programs and detector hardware that generates data. Therefore, it is important that there exists an interface that supports these different types of sources.

Nowadays XML [XML] is becoming increasingly popular, especially in the area of data integration and exchange. The XML format can be used to create the basic data description. In a later stage XSL [XSL] or queries can be used to create a view on the data in many ways.

Because different organizations (or even different parts of the same organization) rarely standardize on a single set of applications, and therefore data formats, it takes a significant amount of work for data sources operated by two groups to communicate. XML provides standardized mechanisms for the exchange of structured data. The self-describing format of XML enables different organizations (or units within organizations) to display/manipulate the same XML data in different ways, using for example, XSL. The organizations can use the same tools to manipulate this data. The big advantage is that it is vendor neutral and becomes accepted as a format for data exchange.

As a consequence of this, many database vendors are trying to make their data "XML compliant". A common strategy for this is to serialize data and exchange this with other sources. Within CMS users frequently want to retrieve and compare parts of data from different data sources.

These sources have different query mechanisms. It would be convenient if a user could query all these sources using one interface. An approach for this could be to serialize all the sources into XML. However, this means that the serialized sources have to be consistent with the original source (replica management). Furthermore, this

requires more disk space. Serialization can also hide or lose information.

Figure 1 shows how a reference of objects (within an object oriented database) can be hidden in XML (when serialized). Within an object database it would be straightforward to retrieve object2 using the object1 and the reference to object2. Within an XML file this is more difficult to express. Furthermore, methods of objects would not be accessible anymore. Instead of serializing the complete data source, parts could be serialized (serialization on demand). These parts are based on queries from users. Within these queries, users could use concept of methods and references common for an object oriented database.

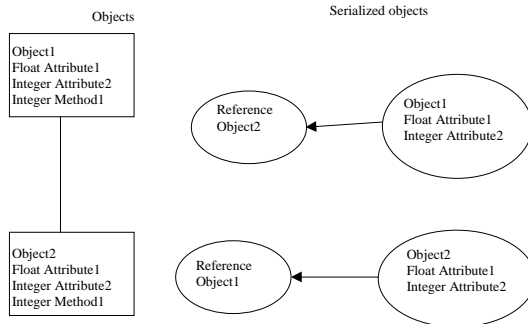


Figure 1. Hiding & losing information

Instead of serializing data sources into XML data sources can be “wrapped” into XML. This paper discusses how an object oriented database can represent itself as an XML document by mapping OO concepts to the XML specification. It enables users to construct XML queries on multiple sources, using the same interface. The XML interface forms a wrapper for the data sources. The paper focuses on how XML queries can be translated to OQL [Cattell et al. 2000] queries. To accomplish this the Xpath specification [Xpath] is extended in syntax, with rules and constraints.

Section 2 discusses related work. Section 3 describes specific characteristics of object oriented databases and discusses extensions on Xpath. Section 4 discusses a mapping from objects to XML nodes. Section 5 describes how Quilt queries are rewritten into OQL queries by using the mapping discussed in section 4. Section 6 contains the conclusions.

Throughout the paper the Quilt query specification [Robbie et al. 2000], Xquery [Xquery], and Xpath specification [Xpath] will be used. Xpath is a navigation mechanism for XML documents. It allows selection of nodes using conditions on the attributes, parent and child nodes. Quilt is a proposal for a query language for XML documents. It is based on Xpath for selecting nodes, but allows joins over (multiple) documents, recursive queries and returns the result as an XML document. Furthermore, it contains standard functions such as *min*, *max*. That are found in numerous query languages. Its structure is based

on features found in query languages like SQL [Levene & Loizou 1999], and OQL. Kweelt [Sahuguet 2000] is one implementation of Quilt.

2. RELATED WORK

Much work has been done on serialising, and de-serialising data from relational databases, object oriented databases, and other sources. Given a DTD (document type definition), the XML Access Server Lightweight Extractor (XLE) [XLE] allows a user to annotate the DTD to associate its various components with underlying data sources, and when requested, extracts data from the data sources and assembles the data into XML documents conforming to that DTD.

The WISDOM project [Bhatti et al. 2000] serialises objects for Objectivity databases [Objectivity]. KOML, the KOala XML serialisation tool [KOALA], provides an easy way to serialise and deserialise any Java objects in an XML document. XMOP allows interoperation between object technologies such as Java, Microsoft COM and CORBA [XMOP]. XMOP is unique in available object serialization that is not tied to a particular system. XML-CORBA Link enables new or existing CORBA systems to integrate with standards-based, web-enabled applications.

With large data sets one might only want to serialize what is needed and XLE can serialize on demand. However, you need to construct a DTD for this. [Lee & Chu, 2000] show how to store XML documents in a relational database. It discusses how constraints within a DTD can be preserved within an relational schema and describes algorithms that automate this process.

Another possibility is to represent the database either as an XML document, or as data published from a database as XML document. The former uses an XML query language to access a data source. This XML query is then translated to the query dialect. The latter uses the query dialect from that source and translates the result into XML. [Shanmugasundaram 2000] describes how to publish relational data as XML. It compares strategies of adding tags and structuring the results of a query. Parts of this can be done inside the Query engine, or outside the query engine. [Carey et al. 2000] describes an architecture for querying object relational databases using XML-QL [Florescu et al. 1999]. [Manolescu et al. 2000] describes a system that allows users to query data using Quilt based on domain specific views. The underlying relational data sources are wrapped in XML.

3. OBJECT CHARACTERISTICS

This section will describe several characteristics of object oriented databases and gives some informal examples of its use within queries using a Quilt syntax. The next sections will then focus on a more general mapping from objects to nodes.

An object within an object-oriented database does not only contain attributes, but also methods. Methods have arbitrary return types. These can be float, integer, strings or objects. Within the Xpath specification there are several standard functions that can be used (`parent()`, `firstchild()`). These functions relate to the XML format. Instead of these standard functions it is possible to use the methods of the objects. A method can be considered as an attribute if it returns always one string, float, or integer. A method can be considered as a node if it returns a set of objects or a set of floats, integer, or strings.

Suppose there is a database with geometry objects. A geometry object can consist of other geometry objects. Furthermore, there is a method `volume()` that calculates the volume and a method `coordinates()` that returns a coordinate object. The `coordinate` object contains an integer attribute `Xcoordinate`. It is possible to create the following OQL query (`childgeometries` returns the child geometries of `p`):

```
SELECT q.coordinates().Xcoordinate FROM
geometry p, p.childgeometries() q WHERE
p.volume()<10 AND q.volume()<5
```

Within an Xpath context, `coordinates()` and `childgeometries()` would represent other nodes, while `volume()` would represent an attribute. The OQL query can be rewritten in the following Xpath syntax:

```
//geometry[@volume()<10]/childgeometries()[@volume()
<5]/coordinates()/@Xcoordinate
```

Methods can have input variables. Either these variables are hardwired numbers, or are bound to a variable. Consider the following example. Suppose a geometry object contains a method `physicalproperties($a)`, which takes as input a variable `$a` that represents the density of a material. This method calculates the weight of the geometry object and the conductivity through the material and returns these in an object “`prop`”. Furthermore, the database contains `material` objects that have an attribute `density`. Using a Quilt syntax it is possible to query for geometries that have a density smaller than 10 and a weight larger than 100:

```
FOR $a IN //material[@density <10]/@density
$b IN //geometry/physicalproperties($a
```

```
WHERE $b/@weight >100
```

```
RETURN $b/@conductivity
```

This results in the following OQL query:

```
SELECT q.conductivity FROM material p,
geometry.physicalproperties(p.density) q
WHERE p.density <10 AND q.weight >100
```

Within an object oriented database it is possible to have multiple attributes within an object that are references to the same object type. For example there can be an object “`store`”. This object contains two lists. “`Computers sold`”, “`Computers in stock`”. Both lists contain the object `computer`. The object `computer` contains attributes: `price`, `model`, `date` (this is date sold or purchase date by the store). The sold computers are in a list referenced by attribute `soldcomputers`. The computers in stock are in a list referenced by an attribute `computersinstock`. The following OQL query returns the models and prices of sold computers with a price less than 3000:

```
SELECT p.price, p.model FROM store q,
q.soldcomputers p WHERE p.price <3000
```

If Quilt and Xpaths are to be used, it requires a specification of which reference is used. This can be done by using a prefix to the objects/nodes. This leads to the following notation using a Quilt query:

```
FOR $a IN store/[soldcomputers]computers
WHERE $a/@price <3000
RETURN $a/@model, $a/@price
```

The extended Xpath relates to simple OQL queries. Within Quilt there is the possibility to perform nested queries. These extended Xpaths could be used within these nested queries. Most of the special functions within Quilt are derived from OQL, and SQL.

4. OBJECTS TO NODES

This section introduces a mapping between object instances and XML documents. Consider an instance of a database schema. It consists of objects, references to objects (objects that serve as attributes within the other objects are also considered here), and attributes (string, integer, float). Consider an object as a node. The attributes of a node are the attributes of the object and the

children of a node are the references to other objects. Note that we do not have a notion of parent because in arbitrary databases there is no unique parent. It is possible to introduce a notion of parent by explicitly defining certain references as parent child relationships. However, we will only use the child relationship with respect to references of objects. It is now possible to navigate through our database using the notion of nodes and Xpath. The following example (Figure 2) shows the representation of objects as an XML document that has the notion of children but not of parents. In the representation, Object 1 with ID 1 has children: Object 1, ID 2 and Object 3, ID 3. While Object 1, ID 2 has children: Object 1, ID 1 and Object 3 ID 4.

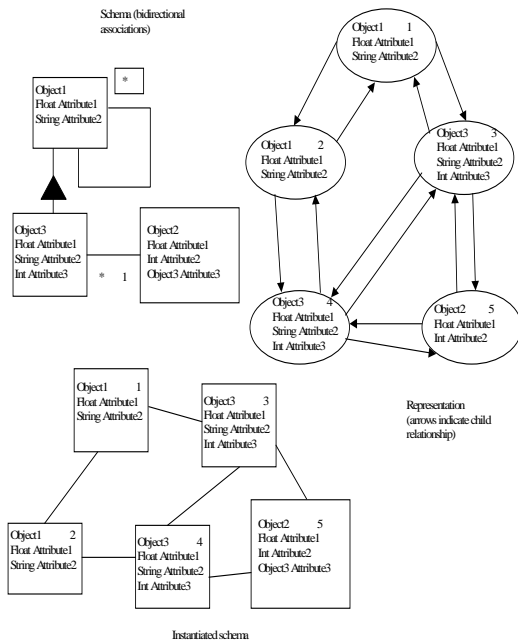


Figure 2. Objects as nodes

Notice that an XML document can be represented as a finite tree (finite number of edges and nodes). Within our example (Figure 2), and in most object oriented databases this is not possible if we use the previous mapping of concepts from object to nodes. However, it is possible to create a mapping to an infinite tree by using the following mapping: Create a root node. For every instance in the database create a node that is a child of the root node. The children of a node “N”(and corresponding instance “I”) in this tree are the objects mapped to a node, that can be reached by a reference from “I” within the database, that is an attribute within instance “I” of object type (not float, string or integer), or are objects returned by methods of “I”. Attributes of instance “I” are attributes, or methods

that have a return type of type float, integer, string. If a database contains cycles of references this tree will not be finite. This tree will never be constructed but shows the equivalence between a collection of object instances and references in a database (a graph), and an XML document (a tree). Notice that within this tree it is only possible to use the notion of children and not parents.

The mapping shows the resemblance between XML documents and object oriented schemas. Furthermore, it creates the possibility of using Xpaths on object instances in the same way as in XML documents. The question arises if it is possible to transfer every query on an object-oriented database into a Quilt query. Queries within OQL always deal with paths of objects in the database with constraints on the attributes (string, integer, float), or conditions on objects. Using the previous mapping from object instances to a tree these queries map to Xpaths and the Quilt syntax.

5. QUERY REWRITING

This section explains how Xpath and Quilt queries for an object oriented database can be rewritten. Currently we consider simple quilt queries (no recursion). An OO database schema, can be represented as a tuple: $O(SA,OA,SM,OM,OR)$, in which SA is the set of attributes that have a type float, integer, or string (types that can be used within Xpath and Quilt). OA are attributes that have an object type. SM are methods that have a return type that is a float, integer or string. OM are methods that have an object return type. OR is the set of references to other objects. The Xpath grammar is based on the document object model (DOM) and deals with nodes and attributes.

To support an object database there needs to be a mapping from the nodes and attributes to objects, attributes and methods. The following rules and constraints are added to the Xpath grammar:

- Nodes within an Xpath are represented by objects and elements of the sets OR, OA, OM of objects.
- Every Xpath starts with an object
- If a node N is an element of OR, OA, or OM of an object O. The predecessor of that node is either object O or a node M that is an element of OR, OA or OM of an object P and has return type O.
- The previous constraints relate a node within an Xpath to an object type. Attributes within an Xpath of node N of type O are represented by elements of SA and SM of object O.

Based on these rules Xpaths can be rewritten into OQL queries. Consider the following Xpath (N_i are nodes,

C_1 constraints on the attributes of that node and x an attribute of node N_n):

$N_1[C_1]/N_2[C_2]/\dots/N_n[C_n]/@x$

Rewriting is straightforward. The constraints are as found in the Xpath specification. However, the attributes in these constraints can also be methods with a float, string, or integer return type. For example C_1 could be: $f_j=10$ AND $a_j=5$. In which : f_j is a method with return type integer, and a_j an attribute of node N_j . For convenience we write this as $N_j.C_1$ in the OQL query. The Xpath results in the following OQL query:

SELECT $X_{n,x}$ **FROM** $N_1 X_1, X_1.N_2 X_2, .. X_{(n-1)}, N_n X_n$
WHERE $X_1.C_1$ AND $X_n.C_n$

Note that if there are no constraints attached to a node it is not necessary to bind it to a variable within the OQL query. Within Quilt queries, bindings to Xpaths are used to build a query. As a consequence part of an Xpath can be used in different Xpaths. Consider the following example:

FOR $\$a$ **IN** $store/[soldcomputers]computers$
WHERE $\$a/@price < 3000$
RETURN $\$a/@model, \$a/@price$

Two Xpaths can be identified:

$store/[soldcomputers]computers/@price$
 $store/[soldcomputers]computers/@model$

Rewriting this to an OQL query should only rewrite the first part of the Xpath once. When parsing a Quilt query this can be done by introducing the following rule:

- If a binding is used within an Xpath, the Xpath that is binded to this variable is already rewritten.

The binding variable is attached to the last variable used to rewrite the Xpath as a label. If the binding variable is used within another part of the Quilt query it is possible to trace the variable used within the OQL query. WHERE statements in Quilt are rewritten in the WHERE part of the OQL statement. If the WHERE statements contains Xpaths with constraints on attributes this is put in the select part of the OQL query. The following shows an example of how an quilt query is rewritten (N_1 are nodes, C_1 constraints and x,y,z attributes):

FOR $\$a$ **IN** $//N_1/N_2[C_2]/N_3,$
 $\$b$ **IN** $//N_4[C_4]/N_5/N_6[C_6]$

WHERE $\$a/N_5/@x = \$b/N_7[C_7]/@y$
RETURN $\$a/@z$

Parsing first line (binding to $\$a$ rewritten):

SELECT
FROM $N_1.N_2 p, p.N_3 q$ (label $\$a$)
WHERE $p.C_2$

Parsing second line (binding to $\$b$ rewritten):

SELECT
FROM $N_1.N_2 p, p.N_3 q$ (label $\$a$), $N_4 r, r.N_5.N_6 s$ (label $\$b$)
WHERE $p.C_2$ AND $r.C_4$ AND $s.C_6$

Parsing the third line (t is introduced since C_7 are conditions on attributes of N_7 and have to appear in the WHERE statement):

SELECT
FROM $N_1.N_2 p, p.N_3 q$ (label $\$a$), $N_4 r, r.N_5.N_6 s$ (label $\$b$), $s.N_7 t$
WHERE $p.C_2$ AND $r.C_4$ AND $s.C_6$ AND $t.C_7$ AND $q.N_5.x=t.y$

Parsing fourth line and removing labels:

SELECT $q.z$
FROM $N_1.N_2 p, p.N_3 q, N_4 r, r.N_5.N_6 s, s.N_7 t$
WHERE $p.C_2$ AND $r.C_4$ AND $s.C_6$ AND $t.C_7$ AND $q.N_5.x=t.y$

This query will then return the desired result. The nodes in the Xpath can represent methods with input variables. The following rules are introduced for input variables:

- Input variables are either hardwired (e.g. a number) or bind to a variable. The binding is with an Xpath. This rule allows input variables of methods to be rewritten in a straightforward manner.

Note that within XML documents it is possible to return a node within the RETURN statement. This will return all the sub tree nodes of this node. Because XML documents contain a tree structure this will be a finite list. However, the translation of Xpaths used in this paper for objects could result in an infinite answer. To offer some possibility of retrieving children of objects one could specify the depth that one wants to view in the sub tree (as

defined in this paper). No depth will only serialize the node (e.g. RETURN \$a). RETURN \$a(3) would return the "sub tree" nodes up to level 3, of nodes binded to \$a. Note that object schemas can contain cycles. This can result in duplicates of nodes.

Data within OQL queries are type sensitive and XML data are plain ASCII. Returning nodes from a Quilt query on top of an object oriented database, would mean that objects need to be serialised. Two types of serialisation are possible. The first one serialises the values of the attributes. This serialisation loses the type information of the object. The other one includes type information.

For example: Suppose an Quilt query contains **RETURN** \$a(1). Translated to OQL, this \$a is a binding to object O₁(float x, string y, O₂ z). If type information is serialised, this would result in a collection consisting of the following serialised components:

```
<Object type=O1>
  <Attribute name="x" type="float">
    "value of x"
  </Attribute>
  <Attribute name="y" type="string">
    "value of y"
  </Attribute>
  <Attribute name="z" type="O2">
    <Attribute name="....."
      "attribute values of z (if not of an object type)"
    </Attribute>
  </Attribute>
</Object>
```

Type information is useful if data will be used within other applications or if manipulations are done on certain types of data. However, if Quilt is used within a programming language, data does not have to be serialised and can be treated in the same manner as results from OQL queries.

It is possible to model a relational schema using UML (Muller 1999), thus implying that it is also possible to represent a relational database as an XML document, based on the previous mapping. A lot of research has been done in this field ([Shanmugasundaram 2000],[Carey et al. 2000],[Manolescu et al. 2000]). Therefore we will not discuss this subject.

6. CONCLUSIONS

This document described how object-oriented databases could be represented as XML documents without serializing the complete database. This is

realized by a mapping from objects to nodes. This mapping is never constructed, but is used in relating queries in OQL to Quilt. Currently the Xpath specification is based on XML files. However we expect that in the near future XML will not only be important as a format but also as an interface. This paper considered the case of object-oriented databases in which nodes are mapped to objects and methods.

The approach in this paper prevents complete serialization and replica management on the source. Instead users can use queries to locate specific pieces of data. Furthermore, within Xpaths and Quilt queries, methods and attributes that are not integer, strings or floats can be used. Supporting the navigational capability needed to query an object oriented database. A drawback of this approach is that every time a user accesses data, it needs to be serialized. If several users access the same data this will result in a serialization overhead. Serialization on demand is desirable if data in the database changes frequently and if users access different parts of the data. Complete serialization can be beneficial if data does not change much, and data is accessed frequently. Within SQL there is the possibility to make updates on the data. Xupdate [Xupdate] is a project that tries to achieve this for XML. This Xupdate could be mapped to the SQL update statements. However, at the moment there is no update syntax within OQL.

Quilt allows for recursive queries. This was not addressed in this paper. Recursive queries are part of future work on this subject. The paper dealt with queries on one object database. However, the Quilt specification allows for queries on multiple sources. This raises questions about how to rewrite and split queries in local parts and join the result afterwards.

The WISDOM project at CERN [Bhatti et al. 2000] looked at (de)serialization of Objectivity [Objectivity] databases in XML. Furthermore, an OQL query engine is being developed for objectivity. Currently the project is investigating the problem of generic XML queries of object oriented databases.

ACKNOWLEDGEMENTS

The author likes to thank CERN, University of the West of England and Eindhoven University of Technology for their support and Christoph Koch, Richard McClatchey, Peter v/d Stok, and Ian Willers for their input in writing this paper.

REFERENCES

-Bhatti, N. Le Goff, J.M. Hassan, W. Kovacs, Z. McClatchey, R. Martin, P. Stockinger, H. Willers, I. 2000, "*Object Serialisation and Deserialisation Using XML*", 10th International Conference on the management of Data (COMAD 2000) India,

-Carey, M. et al. 2000 "*XPERANTO: Publishing Object-Relational Data as XML*", Int. Workshop on the Web and Databases, Dallas, Texas

-Cattell, R.G.G. Barry, D. K. Berler, M. Eastman, J. Jordan, D. Russell, C. Schadow, O. Stanienda, T. Velez, F. (ed.) 2000 "*ODMG 3.0*", ISBN 1-55860-647-5

-CERN, URL: <http://cern.web.cern.ch/CERN/>

-CMS, URL: <http://cmsinfo.cern.ch/Welcome.html>

-Florescu, D. Deutsch, A. Levy, A. Fernandez, M. Suciu, D. 1999 "*A Query Language for XML*" In Proceedings of Eight International World Wide Web Conference

-Geiger, K. "*Inside ODBC*", 1995 Microsoft Press, ISBN 1-55615-815-7

-Kim, W. 1995 "*Modern Database Systems*", Addison-Wesley.

-Koala Object Markup language URL:

<http://www.inria.fr/koala/XML/serialization/xmlindex.htm>

-Lee, D. Chu, W. 2000 "*Constraints-preserving Transformation from XML Document Type Definition to Relational Schema*" Proc. 19th Int'l Conf. on Conceptual Modeling (ER), Salt Lake City, Utah,

-Levene, M. Loizou, G. 1999 "*A Guided Tour of Relational Databases and Beyond*" ISBN 1-85233-008-2

-LHC, URL: <http://lhc.web.cern.ch/lhc/>

-Manolescu, I. Florescu, D. Kossman, D. Xhumari, F. Olteanu, D. 2000 "*Agora: Living with XML and relational*", VLDB pp 623-626.

-Muller, R.J. 1999, "*Database Design for Smarties: Using UML for Data Modeling*" Morgan Kaufmann Publishers, ISBN 1558605150

-Objectivity, URL: <http://www.objectivity.com>

-Robie, J. Chamberlin, D. Florescu, D. 2000 "*Quilt: an XML query language*", XML Europe 2000,

-Sahuguet, A. 2000 "*Kweelt the making of: Mistakes made and lessons learnt*" Technical report, Penn database research group

-Shanmugasundaram, J. Shekita, J.E. Barr, R. Carey, Lindsay, M.B. Pirahesh, H. Reinwald, B. 2000 "*Efficiently Publishing Relational Data as XML documents*" VLDB pp65-76

-XMOP: XML Metadata Object Persistence. URL:

<http://www.openhealth.org/XMOP.htm>

-Xpath. URL: <http://www.w3.org/TR/xpath.html>

-Xupdate. URL: <http://www.xmldb.org/xupdate/xupdate-req.html>

-XLE, URL: <http://www.alphaworks.ibm.com/tech/xle>

-XML, URL: <http://www.w3.org/XML>

-Xquery, URL: <http://www.w3.org/XML/Query>

-XSL, URL: <http://www.w3.org/XML/Linking>