W3C Progress towards standards for XML databases

Maria Chinwala, Rakesh Malhotra, John A. Miller
{chinwala, malhotra, jam}@cs.uga.edu
Department of Computer Science, University of Georgia, Athens, GA 30602

Abstract

XML is becoming increasingly popular as a means of exchanging a wide variety of data on the web. It is anticipated that in the future, many websites will be built from XML documents. XML databases would be required to manage these websites and also provide a way for users to search their contents. A standard XML Query Language and Algebra are required, which should be based on a standard Data Model. The XML Query Working Group of the W3C, established in September 1999 is working towards developing these standards for XML databases. This paper briefly describes the Data Model proposed by the W3C and discusses the work of major research groups in developing a Query Language and Algebra for XML. We conclude by picking the Query Language and Algebra, we consider most likely to be chosen by the W3C as a standard.

1. Introduction

XML is becoming increasingly popular as a means of exchanging a wide variety of data on the web. It is anticipated that in the future, a large number of websites will be built from XML documents. XML databases would be required to manage these websites and also provide a way for users to search their contents. Any XML database would require a Data Model, query language and query algebra. In this paper we will focus on the progress of W3C towards developing standards for XML databases and discuss research that has been carried out in this area .We will examine and compare different query languages and algebra for XML proposed by major research groups and pick out the ones we think will be chosen by the W3C as the standard.

The W3C established the XML Query Working Group in September 1999.The mission of the XML Query working group is to provide flexible query facilities to extract data from real and virtual documents on the web. The XML Query Working Group published its Requirements Document [1] in January 2000 and updated it in August of the same year. This document specifies goals, requirements, and usage scenarios for the W3C XML Query Data Model, algebra, and query language. The Working Group also published a Data Model draft [2] in May 2000.

This paper is organized as follows. In Section 2 we discuss the W3C Data model. In Section 3 we examine and compare the various XML query languages that have been proposed. In section 4 we briefly discuss the work done towards developing an algebra for XML queries. We conclude in section 5.

2. W3C Data Model

Before describing the W3C Data Model we briefly mention DTD'S, XML Schemas and the XML Infoset. XML 1.0 [3] supplies a mechanism, the Document Type Definition (DTD), to define the structure of an XML document with a list of legal elements. However, automated processing of XML documents required more rigorous and comprehensive facilities in this area. There was also a need for specifying the structure of an XML document using XML like tags rather than using DTDs. The XML Schema Working Group was thus established by the W3C and the specification documents [4,5,6] for an XML Schema were published in October 2000. The purpose of a schema is to define a class of XML documents, and so the term "instance document " is often used to describe an XML document that conforms to a particular schema. XML Schema specification documents state that neither instances nor schemas need to exist as documents *per se* -- they may exist as streams of bytes sent between applications, as fields in a database record, or as collections of XML Information Set "Information Items". The XML Information Set [6] provides the description of the information available in a well-formed XML document. The XML Infoset was developed as it was expected that any XML-based standard would always be defined at a higher, logical level and refer to abstract objects like *element* and *data*. The XML Infoset describes these abstract XML objects and their properties. Thus, when the XML Query Working Group published the draft of their Query Data Model [2] they also provided a mapping from the XML Infoset to the Data Model.

The XML Query Data Model formally defines the information contained in the input to an XML Query Processor. In other words, an XML Query Processor evaluates a query on an instance of the XML Query Data Model. Later, we explain the Data Model with the help of an example. Because XML documents are tree-structured, the XML Query Data Model is defined using conventional tree terminology. The Data Model defines the structure of various types of tree nodes; functions to construct tree nodes, called *constructors*; and functions to access nodes' structure called *accessors*.

Consider the following XML document:

```
<dbgroup>
   <member>
      <name>Smith</name>
      <age>28</age>
      <office><building>A</building><room>6</room></office>
   </member>
     :
   <member>
      <name>Clark</name>
      <age>35</age>
      <office><building>A</building><room>7</room></office>
   </member>
</dbgroup>
```

A graphical representation of the data-model for this document is shown in Figure 1. For simplicity, this diagram shows information only about one member of the dbgroup.

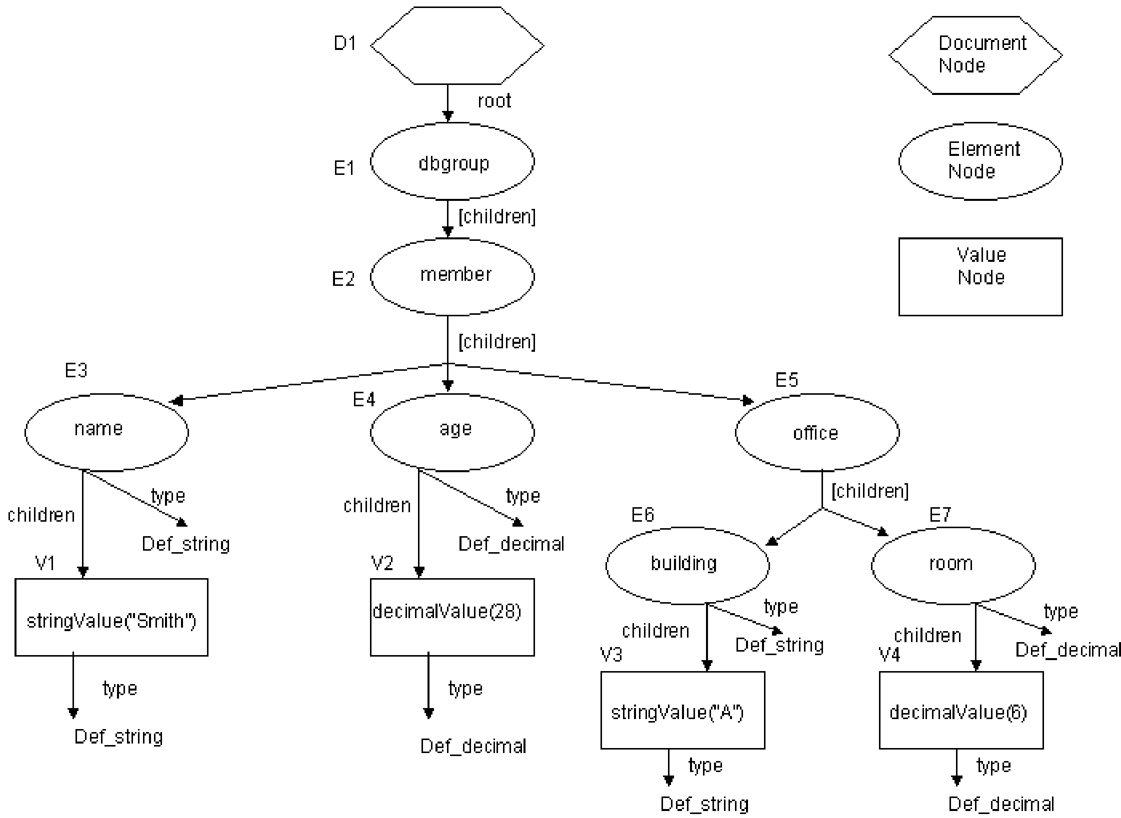


Figure 1.

The basic concept in the Data Model is a *Node*. A *Node* is one of eight *node types*: *document, element, value, attribute, namespace, processing instruction, comment*, or *information item*. A data-model instance is a set, a list, or a bag of one or more nodes. An XML document is represented by it's distinguishing documen*t* node. Each *document* node references a unique *element* node, which is also known as *root*. *Element* nodes in turn refer to other *element* nodes, *value* nodes etc. The Data Model provides constructors for the various primitive XML Schema datatypes such as string, boolean, float etc. For example the constructor for decimals is decimalValue(decimal,Ref(Def_decimal)). Value nodes are created using these constructors.

The data-model is constructed using *constructor* functions and accessed using *accessor* functions. For the data model in Figure1, the *accessor* functions are listed below. To avoid repetition we only list *accessors* for nodes D1,E1,E2 and E3.

children(D1) = [Ref(E1)]

```
root(D1)      = Ref(E1)

name(E1) = QNameValue(null,"dbgroup",Ref(Def_Qname))
children(E1) = [ Ref(E2) ]
attributes(E1) = { }
namespaces(E1) = { }
type(E1) = { }
parent(E1) = Ref(D1)

name(E2) = QNameValue(null,"member",Ref(Def_Qname))
children(E2) = [ Ref(E3) , Ref(E4) , Ref(E5) ]
attributes(E2) = { }
namespaces(E2) = { }
type(E2) = { }
parent(E2) = Ref(E1)

name(E3) = QNameValue(null,"name",Ref(Def_Qname))
children(E3) = [ Ref(StringValue("Smith",Ref(Def_string)))]
attributes(E3) = { }
namespaces(E3) = { }
type(E3) = Ref(Def_string)
parent(E3) = Ref(E2)
```

3. XML Query Languages

The Requirements Document [1] published by the W3C outlines the requirements of a query language for XML data. Although a variety of query languages have been proposed, none has yet been chosen as a standard by the W3C. We will discuss four languages - Lorel, XML-QL, XQL and Quilt.

In the late nineties researchers from the database community independently proposed two query languages for XML, namely XML-QL and Lorel. XML-QL [20] was developed by researchers from the University of Pennsylvania, AT&T Labs, INRIA, and the University of Washington and submitted to the W3C in August 1998. Researchers at Stanford University published a paper in June1997 [21] describing Lorel, a query language for semistructured data, which they had developed for their Lore Database Management System. They later extended and modified Lorel [22], so that is was capable of querying XML data. XQL [23] was a language that originated from the document processing community. It was proposed in September 1998 by researchers from Texcel, Inc., webMethods, Inc. and Microsoft Corporation. In June 2000, researchers from Software AG, IBM and INRIA proposed Quilt [24], a query language which combines the best features of the earlier languages and is particularly influenced by XML-QL and XQL.

It is interesting to note that all these languages were proposed before the W3C published its Data Model draft [2].Hence, XML-QL is based on a graph data model and XQL and Quilt are based on a document or tree model similar to the W3C Data Model. Lorel supports both graph and tree based models.

XML-QL

The graph based XML data model, on which XML-QL [20] is based, is a variation of the semistructured data model [25]. XML-QL has a SELECT-WHERE construct like SQL and borrows features from query languages for semistructured data.It also uses XML like tags to specify constarints or conditions. Consider the query Q1 based on the XML document of the previous section. Assume that this document is at www.a.b.c/dbgroup.xml.

Q1: Find all members of the database group whose age is less than 30 years.

This query can be expressed in XML-QL as follows:

```
WHERE <member>
                <name> $n </name>
                <age> $a </age>
                <office> <building> $b </building> <room> $r </room> </office>
        </member> ELEMENT_AS $m IN "www.a.b.c/dbgroup.xml"
        a < 30
CONSTRUCT $m
```

We now briefly list some of the features of XML-QL. In XML-QL it is possible to query several XML sources simultaneously and produce an integrated view of their data. Skolem functions are used to help transform or restructure results of a query. XML-QL provides regular path expressions which allows querying of elements to an arbitrary depth in the XML-QL data model. The operators that the regular path expressions provide are the alternation (|), concatenation (.) and Kleene-star (*) which are similar to those used in regular expressions. Ordering of results in a XML-QL query can be specified by means of an ORDER BY clause. A special feature of this language is that it allows us to translate data between different ontologies, meaning that we can write queries that extract data from the database for one DTD and transform it into elements conforming to another DTD.

Lorel

In Lorel [21] , an XML data model can be represented either as a literal tree, where the IDREF(s) attributes are just text strings or as a semantic graph where the IDREF(s) attributes are actual links. Lorel query language is an extension of OQL with the SQL/OQL style of querying. The query Q1 is expressed as

```
SELECT db.member
FROM dbgroup db
WHERE db.member.age < 30
```

We briefly list some features of Lorel. It provides language constructs to transform data and return structured results. One of the constructs is the WITH clause used with the SELECT-FROM-WHERE form. It causes the query result to replicate all data selected by the SELECT clause along with all data reachable via path expressions in the WITH clause. The query language also supports Skolem functions for expressive data restructuring. It supports joins on XML documents from different sources, but not much information is available on it currently. Path expressions in Lorel can include wild cards or regular expression operators. Lorel also provides an Order-By clause. We thus see that both Lorel and XML-QL have many similarities since most of the features discussed above are also provided by XML-QL. However, unlike XML-QL, in Lorel range qualifiers can be added to path expression components or variables. For example the query "select y from dbgroup.member x, x.office[1-2] y" returns the first two office subelements of every group member. A special feature of Lorel is its support of a declarative, expressive update language which allows creation of new atomic or complex elements, updation of values of complex and atomic elements etc. Another special feature of Lorel is its use of DataGuides when a DTD is not supplied. DataGuides are a feature of the Lore DBMS and provide concise and accurate structural semantics of the underlying database.

XQL

Unlike XML-QL and Lorel , XQL [23] is a concise language since it was developed as an extension of the XSL pattern language (now known as XPath [26]) .The XSL pattern language describes the types of nodes to look for using a simple pattern modeled after directory notation. For example, book/author means find author elements contained in book elements. XQL builds upon this capability of identifying classes of nodes, by adding boolean logic, filters, indexing into collections of nodes, and more. Our query Q1 can be expressed as

```
member [age < 30]
```

Since XQL originated from the document processing community and was designed specifically for XML documents, it is different from XML-QL and Lorel. Unlike XML-QL and Lorel, XQL does not provide for join conditions across XML documents corresponding to different DTDs but it does provide AND, OR and NOT operators for querying within documents corresponding to a single DTD. Also, XQL does not use any regular expression operator, but does have the (*) operator as a wild card character to specify any arbitrary path in the query. However, like Lorel, it also has a provision for a subscript operator which allows a range of elements to be returned. The result of an XQL expression always preserves document ordering and hierarchy. Other special features of XQL are that the XQL query may be evaluated at any level of a document, without having to navigate from the root. Also, since XQL specification does not indicate the output format; the result of a query can be a node, or a list of nodes, an XML document or just any other structure.

Quilt

Quilt [24] was developed as a small, implementable language that met the requirements specified by the W3C XML Query Working Group [1]. It was developed because its authors felt that the earlier XML query languages tended to be designed either for documents or for data and there was a need for a language that supported queries on information drawn from various sources. Hence they chose to take some of the best ideas from these earlier query languages (particularly XML-QL and XQL), plus some ideas from SQL and OQL and integrated them with a fresh syntactic approach. The design process involved adapting features from existing languages and assembling them to form a new design-hence the name "Quilt". As mentioned earlier Quilt is based on a tree-like data model. A simple form of a Quilt query consists of FOR,WHERE and RETURN clauses. FOR is used to bind the values of one or more variables. WHERE clause applies a filter to these variables and the RETURN clause generates a new document using the selected values. Our query Q1 would be expressed in Quilt as :

```
FOR $m IN document ("www.a.b.c/dbgroup.xml") // member
WHERE $m/age < 30
RETURN  $m
```

From XML-QL, Quilt borrows the concept of constructing a document from bound variables. However unlike XML-QL, where the result of a WHERE clause is a relation composed of scalar values, the result of a WHERE clause in Quilt preserves all the information about the hierarchic and sequential relationships among the components of the original language. From XQL, Quilt borrows its powerful syntax for navigating a hierarchic document and selecting a set of nodes that satisfy some complex condition. This syntax, as we have already discussed, is similar to the abbreviated syntax of XPath [26] .Since an XPath expression, in general, returns a "forest" of nodes on which hierarchy and sequence are preserved, Quilt is able to express queries based on hierarchy and sequence more easily than languages such as XML-QL. We now list some other special features of Quilt. Quilt allows nested queries. It also allows comparison of elements that have substructure. It does so by serializing the content and markup and removing leading and trailing whitespace and then comparing the serialized strings. Also, a Quilt query can begin by defining one or more functions that may be used in the body of the query. Quilt can be used to retrieve data from objects, relational databases, or other non-XML sources, provided this data is expressed in an XML view. From the features listed above, it is clear that Quilt is a versatile language. It is neither document-oriented nor database-oriented, but has the best features of both and can be used to express queries against diverse sources. This makes it our choice for the language most likely to be selected by W3C as a standard.

Other XML query languages

XML-GL [27] is a graph-based query language with both its syntax and semantics defined in terms of graph structures and operations. It allows the formulation of queries to extract information from XML documents and for restructuring such information into novel XML documents. XQuery [28] is not based on any traditional query language, rather its syntax is built directly on XPointer [29] as XPointer is specifically designed for XML and has proven reliable and robust over a long period amongst other reasons to be chosen as a basis for XQuery.

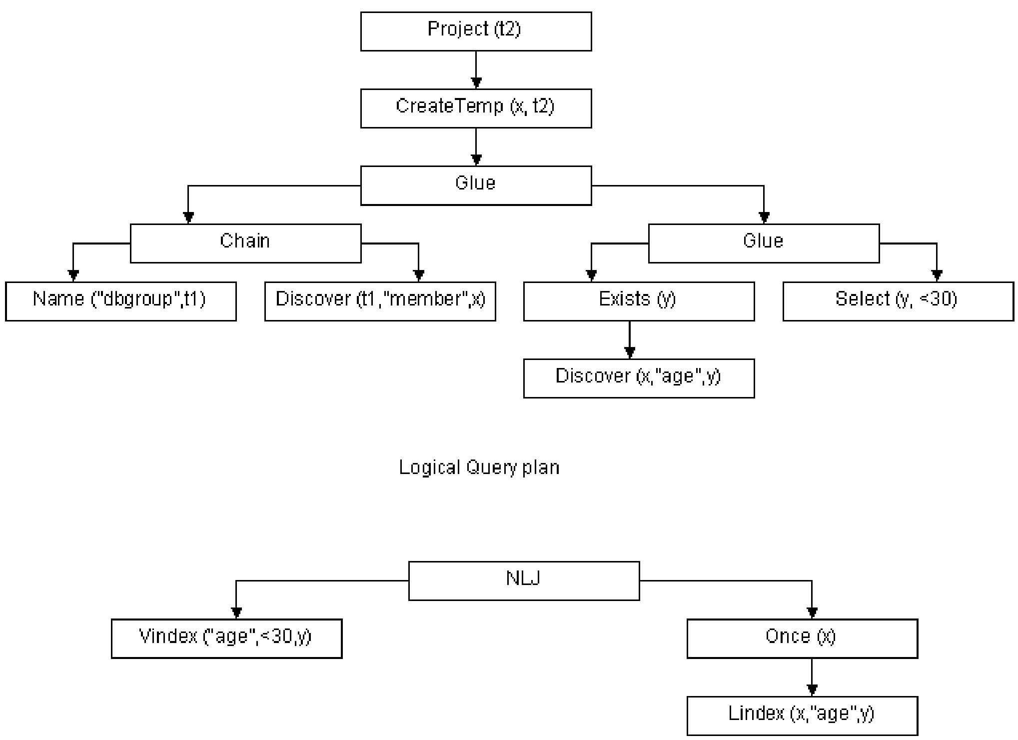
4. XML Algebras

The W3C XML Query Data Model [2] is the foundation on which the W3C wants the XML Query Algebra to be based. As of now there is no specification document for an XML Algebra published by the W3C. Several independent researchers have worked on developing an algebra for XML queries and we will briefly discuss some of this work.

We will concentrate on the work of four major research groups. In August 1999 a paper [8] describing the query processor for Lore, a DBMS for XML-based data was published by the research group at Stanford University. Although in this paper the authors have not explicitly proposed an algebra for XML, they have described logical operators such as *Select, Project* etc., which are used to construct logical query plans. In September 1999 researchers from Oracle, IBM and Microsoft proposed a formal data model and algebra for XML [9]. In May 2000 another group of researchers proposed an algebra [10] suitable for any semistructured or XML query language and especially efficient for XML-based integration systems. Finally, in June 2000 researchers from AT&T and Bell Labs proposed an XML Algebra [11] for which a demo version is also available online at http://www.cs.bell-labs.com/~wadler/topics/xml.html#algebra. An updated version of this paper was published in December 2000 [12] and the latest version is [13].

Lore Algebra

The Lore system at Stanford is a complete DBMS designed specifically for semistructured data. The main focus of Lore's query processor is on cost-based query optimization, specially the efficient evaluation of path expressions. The basic optimization techniques have been adapted from previous work in this area [14,15] and several new types of indexes for efficient traversals through data graphs have been proposed. Each query is transformed into a logical query plan using logical operators such as *Select, Project, Discover, Name,* etc. which can be considered algebra operators. Each logical query plan can give rise to more than one physical query plan and a cost based approach is used to select the best physical plan. The logical query plan and one of the several possible physical query plans for query Q1 are shown in Figure 2.



Logical Query plan

Physical Query plan

Figure 2.

Each simple path expression in the query is represented as a *Discover* node, which indicates that in some fashion information is discovered from the database. Simple path expressions are grouped together into a single path expression as a tree of *Discover* nodes connected via *Chain* nodes. It is the responsibility of the *Chain* operator to optimize the entire path expression represented in its left and right subplans. Places where independent components meet are called *rotation points* since during the creation of physical query plans the order between independent components can be rotated to get vastly different physical query plans. Each rotation point is represented by a *Glue* node. The *CreateTemp* and *Project* nodes at the top of the plan are responsible for gathering the satisfying evaluations and returning the appropriate objects to the user. One of the physical operators is *Vindex(l,Op,Value,x)* which accepts a label *l*, an operator *Op* and a *Value* and places into *x* all atomic objects that satisfy the "*Op Value*" condition and have an incoming edge labeled *l*. The *Lindex(x,l,y)* operator places into *x* all objects that that are parents of *y* via an edge labeled *l*. The *Once(x)* operator only allows an evaluation to be passed to its parent node if the object bound in x has never been seen before. NLJ is the nested-loop join operator which denotes a dependent join where the left subplan passes bound variables to the right subplan. Thus, we see that the main physical operators are index operators, which are used to represent very specific types of indexes maintained in the Lore system. The main drawback of this algebra is that since the logical operators have been developed keeping in mind the physical index operators, they become very specific to the Lore system.

IBM Algebra

The IBM Algebra [9] proposed at about the same time by researchers from IBM, Oracle and Microsoft is not as system specific as Lore. They provide algebra operators that operate on a graph-based data model, proposed by them. In contrast to Lore, this is purely a logical model and nothing is specified about the underlying storage representation or physical operators. This algebra has more expressive power than Lore since in addition to standard retrieval and querying operators such as path navigation, selection, join, etc., they also provide what they call reshaping operators, which help create new XML documents from fragments of selected documents. Also, the data model of Lore, although graph based, does not represent ordered sub elements, attribute-value lists etc. This is not the case with the data model proposed in the IMB Algebra paper since it adheres to the XML Infoset Specification [R] with slight modifications. The path navigation operator (ϕ) is expressed as

ϕ [*edgetype,name*](*vertex-expression*)

The *edgetype* can be E(element), A(attribute) or R(reference). The *vertex-expression* describes the collection of vertices in the data model graph from which edges with the given *edgetype* and *name* should originate. The (ϕ) operator returns this set of edges. For simplicity, ϕ[E,b](a) is written as a/b and the query Q1 is expressed as shown below:

σ[*value*(x/member/age/~data)<30](x:*child*(/dbgroup))

Here (σ) is the selection operator and *child* is the property of an edge that returns the vertex referred to by the edge. The selection operator returns a collection of vertices. The main drawback of this algebra is that no optimization strategies or rules have been discussed.

YAT Algebra

The algebra proposed in [10] was developed by the authors for YAT, their integration system which integrates data from different sources such as object databases or semi structured web repositories. The authors have proposed only two new operators , namely the *Bind* and the *Tree* operators. All others are standard operators drawn from earlier work on Relational Algebra and Object Algebra[16]. Starting from an arbitrary XML structure, the *Bind* operation is applied, whose purpose is to extract the relevant information and produce a structure called *Tab*, which is comparable to a 1NF relation. Then, on the *Tab* structures, standard operators such as *Join, Select, Project* can be applied. Finally, the inverse operation to *Bind*, called *Tree* generates a new XML Structure. Thus our query Q1 can be represented as shown in Figure 3.

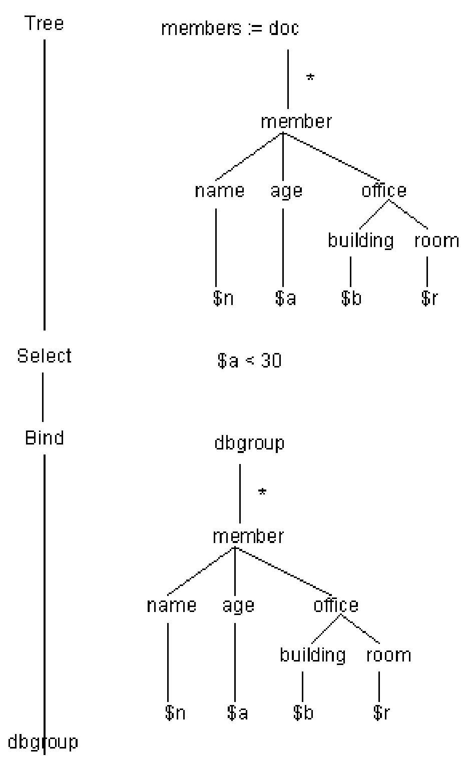


Figure 3.

This algebra is independent of any underlying storage structure. Since the main focus of this paper is on efficient query evaluation for an integrated system the authors have discussed optimization/rewrite rules only for the *Bind* and *Tree* operators. Bind is quite a powerful operation providing support for type filtering and horizontal and vertical navigation. User queries are optimized by generating integration views locally or by pushing queries to the external source. The *Tree* operation is a sequence of *Group, Sort* and *Map* operations which are Object algebra [16] operators on which optimization techniques of Object algebra can be used. This XML algebra is better than the IBM Algebra since it is optimizable. The drawback is that the authors have developed it in the context of an integration system using their own data model and type system and not provided a well defined list of operators and explicit optimization rules.

AT&T Algebra

In contrast to YAT Algebra, AT&T Algebra [13] has a well-defined list of operators and explicit optimization rules. It has been inspired by SQL, OQL and Nested Relational Algebra. The authors have also implemented a type checker and an interpreter for the algebra in OCaml, which is a functional programming language. No storage structures have been discussed. In this algebra most of the operations are based on the iteration operation. The iteration operation is a *for* expression that iterates over elements in a document given the path expression. The selection operation is a *for* expression with a *where* clause and the join operation is nested *for* expressions. The operator for path expressions(/) or the Project operation is also built from *for* expressions, the *children* function and the *match* expression. For example, the expression :

```
dbgroup/member
```

is equivalent to the expression

```
for c in children(dbgroup) do
   match c
        case a : member[UrType] do a
        else()
```

where UrType is the type of the element.

Using this algebra, query Q1 is expressed as follows:

```
for m in dbgroup/member  do
     where m/age/data() < 30 do
          m
```

The authors have also described a simple type system that captures the essence of XML Schema [4,5,6]. They have specified a series of optimization rules which simplify the queries by eliminating unnecessary *for* expressions or by reordering or distributing the computations. The main advantage of this algebra over the others is its simplicity. The algebraic expression is very query like and this makes it very intuitive and easy to learn. This, in addition to the simple type system and simple optimization rules, makes this our choice for the algebra most likely to be selected by the W3C as a standard.

Other XML Algebra

In addition to the work done by the four major research groups discussed above, other people have also worked on developing an algebra for XML. In [17] the author proposes an XML data model based on lists. Queries are transformed into an algebraic form called OPAL (Ordered List processing Algebra) and are processed by iterating over lists. The author describes how Finite State Automata can be used to process and optimize queries. In [18] the author introduces an XML query algebra based on a tree automata model. A new data model named XOM is proposed in [19] which is based on a rooted connected directed graph both cyclic and acyclic, whose nodes are labeled and edges are ranked. A new algebra called XOM Algebra, which includes six operations, i.e., object union, difference, select, project , paste and cut is also proposed.

5. Conclusion

In this paper we discussed the progress of the XML Query Working group of W3C towards developing standards for XML databases. We described the tree-like Data Model proposed by W3C on which it wants the XML Query Language and Algebra to be based. We discussed the features of four query languages namely XML-QL, XQL, Lorel and Quilt. We concluded that Quilt would most likely be selected by W3C as a standard query language due to its versatility and due to the specific document-oriented or database-oriented nature of the other query languages proposed. We also examined the algebra for XML proposed by four major research groups namely the Lore Algebra, IBM Algebra, YAT Algebra and AT&T Algebra. Each of these algebra were developed using very different approaches but we concluded that the AT&T Algebra would most likely be selected as the standard by the W3C due to its simplicity and well defined operators and optimizing rules.

References

1. World-Wide Web Consortium XML Query Requirements, Working Draft, August 2000. http://www.w3.org/TR/*query-datamodel*
2. World-Wide Web Consortium, XML Query Data Model, Working Draft, May 2000. http://www.w3.org/TR/*query-datamodel*
3. World-Wide Web Consortium, Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation, October 2000. http://www.w3.org/TR/REC-xml
4. World-Wide Web Consortium, XML Schema Part 0: Primer, W3C Candidate Recommendation, October 2000. http://www.w3.org/TR/xmlschema-0/
5. World-Wide Web Consortium, XML Schema Part 1: Structures, W3C Candidate Recommendation, October 2000. http://www.w3.org/TR/xmlschema-1/
6. World-Wide Web Consortium, XML Schema Part 2: Datatypes, W3C Candidate Recommendation, October 2000. http://www.w3.org/TR/xmlschema-2/
7. World-Wide Web Consortium, XML Information Set, W3C Working Draft, July 2000. http://www.w3.org/TR/xml-infoset
8. J. McHugh and J. Widom. Query Optimization for XML. *In Proceedings of International Conference on Very Large Databases(VLDB)*, Edinburgh, Scotland, August 1999.
9. D. Beech, A. Malhotra and M. Rys. A formal data model and algebra for xml. Communication to the W3C, September 1999.
10. V. Christophides, S. Cluet, and J. Simeon. On wrapping query languages and efficient XML integration. In *ACM SIGMOD Conference on Management of Data, 2000*. (to appear).
11. Mary Fernandez, Jerome Simeon, Philip Wadler. An Algebra for XML Query, Draft manuscript, June 2000.
12. Mary Fernanadez, Jerome Simeon, Philip Wadler. An Algebra for XML Query, In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2000)*, New Delhi, December 2000.
13. Mary Fernanadez, Jerome Simeon, Philip Wadler. A semi-monad for semi-structured data. To appear in *International Conference on Database Theory*, London, January 2001.
14. G. Gardarin, J. Gruser, and Z. Tang. Cost-based selection of path expression processing algorithms in object-oriented databases. In *Proceedings of the Twenty-Second International Conference on Very Large Data Bases*, pages 390--401, Bombay, India, 1996.

15. W. Sun, W. Meng, and C. T. Yu. Query optimization in object-oriented database systems. In *Database and Expert Systems Applications*, pages 215--222, Vienna, Austria, August 1990.
16. S. Cluet and G. Moerkotte. Nested Queries in Object Bases. In *Proceedings of International Workshop on Database Programming Languages*, pages 226-242, New York City, USA, August 1993.
17. Hartmut Liefke, Horizontal Query Optimization on Ordered Semistructured Data, *WebDB'99.*
18. Boris Chidlovskii. Using Regular Tree Automata as XML Schemas. In *Proceedings of the IEEE Advances in Digital Libraries 2000* (ADL 2000).
19. Dell Zhang and Yisheng Dong. A Data Model and Algebra for the Web. In *Proceedings of the 10th International Workshop on Database and Expert Systems Applications*.
20. Deutsch, A., Fernandez, M., Florescu, D., Levy, A., and Suciu, D. (1998). XML-QL: A Query Language for XML. Technical report, W3C. http://www.w3.org/TR/1998/NOTE-xml-ql-19980819.
21. Abiteboul, S., Quass, D., McHugh, J., Widom, J., and Wiener, J. L. (1996). The Lorel Query Language for Semistructured Data. Technical report, Stanford University. http://www-db.stanford.edu/pub/papers/lorel96.ps
22. Goldman, R., McHugh, J., and Widom, J. (1999). From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. Technical report, Stanford University. http://www-db.stanford.edu/pub/papers/xml.ps.
23. Robie, J., Lapp, J., and Schach, D. (1998). XML Query Language (XQL).Technical report, W3C. http://www.w3.org/TandS/QL/QL98/pp/xql.html
24. Jonathan Robie, Don Chamberain,Daniela Florescu. Quilt: an XML Query Language. Presented at XML Europe,Paris. June 2000. http://www.almaden.ibm.com/cs/people/chamberlin/quilt_euro.html
25. Buneman, P. (1997). Tutorial: Semistructured Data. In *Proc. ACM Symposium on Principles of Database Systems (PODS)*, pages 117-121, Tucson, AZ.
26. World-Wide Web Consortium, XML Path Language (XPath), W3C Recommendation, November 1999. http://www.w3.org/TR/xpath.
27. Ceri, S., Comai, S., Damiani, E., Fraternali, P., Paraboschi, S., and Tanca, L. (1998). XML-GL: a Graphical Language for Querying and Restructuring XML Documents. Technical report, University of Milano. http://www2.elet.polimi.it/people/ceri/Xml-gl.
28. DeRose, S. J. (1998). XQuery: A Unified Syntax for Linking and Querying General XML Documents.Technical report,W3C. http://www.w3.org/TandS/QL/QL98/pp/xquery.html

29. DeRose, S. J. (1999). XML XPointer Requirements, Version 1.0, W3C Note 24-Feb 1999. Technical report, W3C. http://www.w3.org/TR/NOTE-xptr-req