1-5-2002

CXQuery: A novel XML query language

Peter Revesz
University of Nebraska-Lincoln, prevesz1@unl.edu

Yi Chen
University of Nebraska-Lincoln

CXQuery: A Novel XML Query Language

Yi Chen, Peter Revesz

***Abstract*— XML is becoming the data exchange standard on the Internet. Previously proposed XML query languages, such as XQuery, Quilt, YALT, Lorel, and XML-QL, lack schema definition of the query result; therefore, they are limited for defining views, integrating data, updating, and further querying, all of which are often needed in e-Business applications. We propose a novel XML query language called *CXQuery*, which defines the schema of the query results explicitly and can easily define views, and integrate, update, and query XML data. In addition, *CXQuery* can express spatial and spatio-temporal queries using a constraint-based querying approach.**

***Index Terms*—XML, Query Language, Constraint Database**

I. Introduction

XML is the emerging standard for data representation and exchange on the Internet. E-Business applications demand a powerful query language with view definition and data integration capabilities. XQuery[2] developed by W3C as the XML query language standard lacks these capabilities. It is derived from several previous proposals of XML query languages (eg, XML-QL[8], YALT[7], Lorel[1] and Quilt[3]), but these languages lack schema information in the query result, XSL[5] is also regarded as an XML query language which allows to define views, but it has a very complicated structure. The limitation of these languages can be summarized as follows:

- The schema information of the XML query result is not explicitly specified. This makes these languages unsuitable to define views for further processing.
- These languages do not reuse the already known DTD of the source document. The user has to build a very complex XML pattern in the query, even though the result shares much common schema information with the source document.
- These languages use patterns to construct the query result. This makes updates of XML documents very complicated. No pattern-based XML query language supports XML updates in a simple fashion. To update a small part of the document, these query languages have to build the complete pattern which obeys the original schema. Besides, the user needs to write the complete schema in the query.

XQuery and its ancestors use XPath[6] to navigate the XML hierarchical structure and use patterns to generate the query output. The output XML document does not have an explicit schema in this approach. DTD reference algorithms were proposed but why not reuse the source DTD in the query result? No schema generation algorithm can be more simple then one which specifies the DTD in the query language explicitly. Besides, the DTD for XML documents can be regarded as the view definition for the data in the document.

To overcome the above limitations, we propose a novel XML query language: *CXQuery*. Our goal is to design an easy-to-learn XML query language which is declarative, has powerful XML query meta data query functionalities, and can define views on XML documents and update XML documents in a simple fashion. Moreover, it can also query XML-based spatial information.

To achieve this goal, we derive our language from constraint query languages, and borrow features from both SQL and several XML query languages. We take advantage of the DTD of XML documents to explicitly specify the schema of the query result. We borrow the schema definition function of the SELECT clause in SQL, and aggregation operators, such as AVG and SUM, in the schema definition part of the query language. We borrow the `document()` function from XQuery to specify the XML document to be queried. XPath path expressions are also used in CXQuery, but only to avoid of name conflicts. CXQuery is similar to XPathLog[10] in that both of them are rule-based query languages. In XPathLog, a rule atom is built upon XPath expressions, whereas in CXQuery, we build our rule atoms on DTD, which gives a horizontal view of schema which can be easily cast and fit into the constraint data model. A source wrapping and declarative integration language is proposed in [4] but it does not provide appropriate updating power on the XML document. The CXQuery query language makes the following contributions:

- Strong schema definitions in queries, which allows it to define XML views.
- More powerful meta data manipulation based on DTD, which allows users to query and update the meta data, and to use the meta data to integrate heterogeneous data sources.
- When combined with a constraint database system, CXQuery can support spatio-temporal queries on XML documents.

II. The CXQuery XML Query Language

CXQuery (Constraint XML Query Language) is a declarative, Datalog-style language for querying and updating XML documents. It employs the syntax and semantics of constraint query languages[9]. The input of a CXQuery is a set of XML documents. The output of a CXQuery query is also an XML document. When CXQuery is used to define views, the query result is not materialized.

A CXQuery expression contains a rule head and a rule body, with a ":-" symbol between them. The rule body contains a set

P. Revesz is a Professor of Computer Science and Engineering Department, University of Nebraska - Lincoln, Nebraska, USA. E-mail: revesz@cse.unl.edu, Web: http://cse.unl.edu/ revesz.

of predicates, which are separated by semicomas. The semicomas stand for the logical operation "AND."

To simplify the CXQuery expression, we employ a subset of functionality of XPath to navigate the hierarchical structure of XML documents and to avoid namespace conflicts. Since most XML documents exchanged in e-Business have relatively restricted structures, we only consider those XML documents that have internal DTD definitions or have external DTD definition connections.

A. XPath and DTD

XPath is a simple XML query language. We use the XPath expression to navigate the hierarchical structure of XML documents. An XPath expression uses the symbol "/" to denote the root node or the children of the current node. We use the function `document()` to denote the root of the XML document. In the following example, $Q1, Q2, Q3$ are all valid CXQuery expressions.

(Query 1) Find the building elements in the campus map in the document "campus.xml".

```
(Q1) document("campus.xml")
     Building(name, dept, spatial);

(Q2) document("campus.xml")
     //Building(name, dept, spatial);

(Q3) document("campus.xml")
     /CampusBuilding
     /Building(name, dept, spatial);
```

When only one XML document is involved, we may use $Q1$ to simplify the expression. When only one document is involved, $Q2$ has exactly the same meaning as $Q1$ has. $Q3$ is stronger then $Q2$ in that it specifies the absolute path from the root of document.

B. DTDs in CXQuery for XML matching

DTD is a simple schema definition for XML documents. It employs a revised syntax of Prolog. If we regard the DTD of a document as a set of predicates, it is actually a view defined on the whole document. In CXQuery, we use DTD in the rule body to match the pattern of XML documents and use a DTD-like rule head to generate the schema of the query result. When processing the CXQuery query, the DTD attached to the XML source is extracted during the parsing phrase. It is regarded as the set of "default" views and transferred to the *DTD Matcher*. All predicates in the CXQuery query body are matched by the *DTD matcher* during querying. The schema of the query result, which is expressed as the predicates appear in the CXQuery head, are merged with the original DTD by *DTD Generator*. When the generated schema of the query result matches one in the original DTD, the query is regarded as an XML update (see Section III).

Each XML document is modeled as a labeled tree structure, the leaves in the tree are defined as primitive type data, for example, integer numbers or strings, but the data type of node

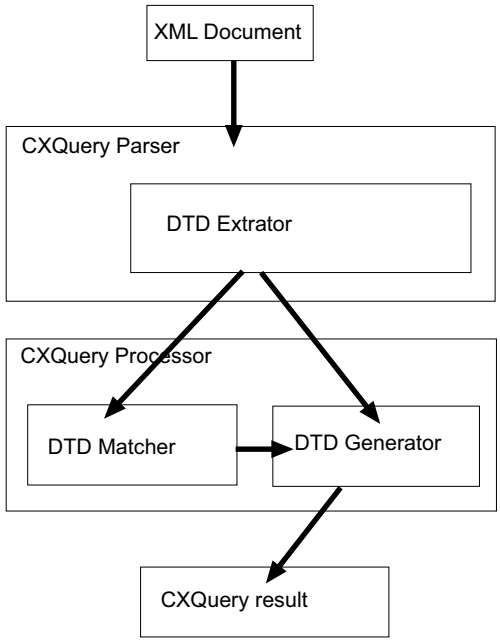

Fig. 1. DTD in CXQuery

is more complex. We borrow the notation of DTD and allow variables in CXQuery to be bound to nodes. That is, CXQuery variable names that match the node (tag in the document) in the XML document denote the subtree of the XML document with this node as the root.

(Query 2) Find the building in "campus.xml" which has the name of "Ferguson Hall".

```
document("campus.xml"),
building(name, dept, spatial),
name = "Ferguson Hall".
```

Query 2 finds all "building" nodes in the "campus.xml" document that has the name "Ferguson Hall". When the function `document(''campus.xml'')` is loaded, the corresponding DTD is also loaded and converted to constraints, which act as the default view definition of the campus document. The second predicate `building(name,dept,spatial)` has the same form with the element definition of building element in the DTD. Since the default DTD is already loaded, the query evaluator matches this "building" element with that in a predefined DTD. The DTD-like predicates shown in the CXQuery body is used for matching. Thus we require that they are in the same form. For example, Figure 2 shows a fraction of the original DTD and the "view" defined on this fraction by constraints.

C. DTDs in CXQuery for query result schema definition

One of the major drawbacks of other XML query languages is that they lack the schema information in the query result. Without the schema information it is difficult to define views using those query languages. In each CXQuery, the schema of the result is explicitly specified in the query head. That is, Prolog like predicates before ":-" are used to construct a fragment

```
DTD:
 <!Element Building (name, dept, spatial)>
 <!Element name          (#PCDATA)>
 <!Element dept          (#PCDATA)>
 <!Element spatial       (X, Y)>
 <!Element X             (#PCDATA)>
 <!Element Y             (#PCDATA)>

Constraints:
 spatial(X, Y):- X(string),Y(string),
 Building(name, dept, spatial):-
       name(string), dept(string),
       spatial(X, Y).
```

Fig. 2. Translate DTDs to Constraints

of the DTD. The query processor will merge this DTD with the original DTD and construct a complete DTD for the output document.

For example, suppose we already have a "view" that contains the building information of the Computer Science department, and we need to find all information related to the building named "Ferguson Hall". We may express this query in CXQuery from scratch, but we may also reuse the "view" as the query source, if we know that the Computer Science department is located in Ferguson Hall. For example, in Query 3 the source of the second rule is the result of the first rule.

(Query 3) Find all the buildings where the computer science department is located, and then find the building information of "Ferguson Hall."

```
buildingview(name,dept,spatial):-
   document("campus.xml"),
   building(name,dept,spatial),
   dept/department="Computer Science";
CSBuilding(name,dept,spatial):-
   buildingview(name,dept,spatial),
   name = "Ferguson hall".
```

In this example, a new element "CSBuilding" was created for the query result. *DTD Generator* will check the variable bindings and create the new element definition:

```
<!Element CSBuilding(name,dept,spatial)>
```

Since name, dept and spatial subelements match those elements in the original DTD, their element definitions are copied directly by the *DTD Generator*. Thus, the DTD of "CSBuilding" is:

When the created element name also matches the element in the source DTD, we consider that an XML update. We discuss updating XML documents using CXQuery in Section III.

D. Functions

CXQuery provides a library of built-in functions for use in general XML queries. We may use SQL aggregation functions (`avg, sum, count, max` and `min`) in the CXQuery queries. We also provide spatial and spatio-temporal operators, such as area and length, to support queries on XML-based spatial and spatio-temporal data. The operands of these spatial functions are the spatial information elements. Since different XML encodings of spatial information have different structures, we need a "spatial data wrapper" to aggregate the spatial data into a spatial object. The proposed functions provide a standard interface to access differently structured spatio-temporal information encoded in XML, thus greatly enhance the power of the query language. For example, Query 4 replaces the spatial element of the building element with an "area" element that represents the area of the building.

```
<!Element CSBuilding (name,dept,spatial)>
<!Element name       (#PCDATA)>
<!Element dept       (#PCDATA)>
<!Element spatial    (X, Y)>
<!Element X          (#PCDATA)>
<!Element Y          (#PCDATA)>
```

(Query 4) Substitute the spatial subelement of building element with the area of the building. The new tag name is building area

```
building(name,dept,
buildingarea:area(spatial)):-
    document("campus.xml"),
    building(name, dept, spatial).
```

III. Updating XML Using CXQuery

Few XML query languages can be used as data manipulation languages to update the original XML document. For example, in XQuery, one of the design goals is to be able to query XML when the schema is unavailable. The schema of the query result thus totally depends on the user constructed pattern. Even if the schema information is available, the "update" operation can only be accomplished by constructing a totally new XML document, where the schema of the new document has to be constructed in the "RETURN" clause, no matter how complex it is. In e-Business applications, most XML documents have a strict structure, that is, having a DTD attached to the XML document. In CXQuery, we assume that the DTD is internally or externally connected to the document, and when updating an XML document, the original DTD can be retrieved from the source document.

A. Updating XML data

Query 5 below shows how a CXQuery can update the name of the building "Ferguson Hall" to "Ferguson Building." This update is impossible to do in XQuery. Here we introduce the ":" operator as an assignment operator in CXQuery.

In Query 5 the query head does not specify the new element name. Hence Query 5 does not construct new elements but only updates the source document.

(Query 5) Update the "campus.xml" document, changing the name of the building "Ferguson Hall" to "Ferguson Building."

```
building(name:"Ferguson Building",
dept,spatial):-
     document("campus.xml"),
     building(name, dept, spatial),
     name="Ferguson Hall".
```

B. Updating XML Metadata

Since schema is explicitly defined in CXQuery queries, CXQuery can also update the metadata and schema of XML documents. The following example shows how to use the ":" operator to change the metadata. This is similar to the "AS" operation in SQL SELECT clauses.

(Query 6) Change the "name" subelement of the building element into buildingname.

```
building(buildingname:name,
dept,spatial):-
      document("campus.xml"),
      building(name,dept,spatial).
```

By default, the variable names in the rule head are bound to the element or variable shown in the rule body. In this example, the "name" element in the "Ferguson" node will bind to the "name" element in the "building" node.

IV. Querying Spatial XML documents

There are several proposals for encoding spatial information in XML. The best known is GML[11] (Geographic Mark-up Language), which is recommended by OpenGIS as the standard to encode spatial information in XML documents. Compared to traditional encoding of spatial information, XML encoding of spatial data has the following features:

- *Spatial data is mixed with property data*
  In traditional spatial database systems, spatial data were stored in separated fields in relational databases, or stored in the file system with a pointer link to the file. But XML is a semi-structured data model, and spatial data can appear in any element and in any layer of the XML tree.
- *Unreadable for users and difficult for direct processing*
  The XML encoding for spatial data could be very complicated, which makes it often unreadable for users. For example, a bus route could be encoded in an XML document as a collection of a hundred line segments. Clearly, such an encoding would be impossible to read by users.

Due to these difficulties, to date there is no query language proposal which supports querying spatial XML documents. Since both CXQuery and many constraint query languages are based on Prolog they can be easily combined. Since constraint query languages can express spatio-temporal queries, the combination leads to a query language for XML documents that contain spatio-temporal data.

Moreover, combination can be easily implemented on top of a constraint database system. As mentioned before, CXQuery itself is a good view definition language. The global view of the document is defined in CXQuery.

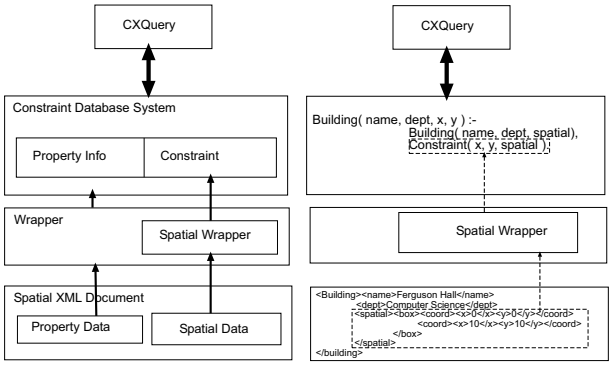

Fig. 3. Architecture for combining CXQuery and Constraint Databases to query Spatial XML Documents.

Figure 3 shows the architecture to query spatial XML documents using a combined CXQuery and Constraint Query Language. The Wrapper sits between the spatial XML document and a constraint database system, which translates the document into constraints. This wrapper is developed for every XML-encoding of a spatial document. Its output is a standard constraint database representation of the XML data. Therefore, CXQuery can use a unified spatial operator to query spatial data and does not need to know the representation details of the spatial data in XML document. In this example, the spatial wrapper implements the function `Constraint(x, y, spatial)`, which converts the "spatial" element into a constraint database representation. This architecture makes possible the integration of spatial XML documents.

Query 7 shows a spatial query. The first two rules construct the constraint representation of the spatial data from the XML documents. The third rule uses a spatial function *Contains()* to test the spatial relation of two spatial objects.

(Query 7) Find all buildings located in citycampus and belonging to the Computer Science department.

```
citycampus(id,constraint):-
   document("citycampus.xml"),
   citycampus(id, departments, buildings,
   BoundedBy),
   constraint(x, y, BoundedBy);

Building(name, dept, constraint):-
   document("campus.xml"),
   Building(name, dept, spatial),
   constraint(x, y, spatial);

Building(name, dept, constraint):-
   Building(name, constraint),
   citycampus(id, constraint),
   contains(citycampus/constraint,
            Building/constraint),
   Building/dept = "Computer Science".
```

V. Integrating Heterogeneous XML Documents

CXQuery can be used to query through multiple XML documents and is a very powerful XML integration tool. Suppose in XML document "citycampus.xml" the buildings are encoded in the `<building>` element which has the schema building(name, dept, spatial). In another XML document "eastcampus.xml" the buildings are encoded in the `<campusbuilding>` element which has a different schema definition: campusbuilding(id, department, location). Query 8 shows a solution to integrate these two schemas into one schema building(name, dept, spatial, location). We use the ":" operator to handle the difference between the two schemas.

(Query 8) Build an XML document with the schema building(name, dept, spatial, location) from the "citycampus.xml" and "eastcampus.xml" documents.

```
building(name, dept, spatial,
location:NULL):-
    document("citycampus.xml"),
    building(name, dept, spatial);

building(name:id, dept:department,
spatial:NULL, location):-
    document("eastcampus.xml"),
    campusbuilding(id, department,
    location).
```

VI. Conclusions

We presented a novel XML query language, called CXQuery. This query language takes advantage of DTDs and uses the DTD-like rules to define the schema of the query result. We show that this language is powerful for querying, updating and integrating XML documents, and it also can be combined with constraint query languages that are also based on Prolog-like rules. We are currently implementing the architecture in Figure 3 on top of a constraint database system, MLPQ/Presto that was developed by our research group.

References

[1] 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** (1997), no. 1, 68–88.

[2] D. Chamberlin, J. Clark, D. Florescu, J. Robie, J. Simon, M. Stefanescu (Eds), Xquery: A query language for xml, *W3C Working Draft, June 2001. http://www.w3.org/TR/xquery/*

[3] D. Chamberlin, J. Robie, and D. Florescu, Quilt: An XML query language for heterogeneous data sources, *WebDB (Informal Proceedings)*, 2000, pp. 53–62.

[4] V. Christophides and S. Cluet and J. Simèon, On Wrapping Query Languages and Efficient XML Integration, *In Proceedings of ACM SIGMOD Conference on Management of Data*, Dallas, Texas, May 2000.

[5] J. Clark and S. Deach, Extensible stylesheet language (XSL), *http://www.w3.org/TR/WD-xsl*

[6] J. Clark and S. DeRose, XML Path Language (XPath) version 1.0, *W3C Working Draft*, July 1999.

[7] S. Cluet and J. Simèon, YATL: A Functional and Declarative Language for XML, Draft manuscript, Mar. 2000.

[8] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu, XML-QL: A query language for XML, *W3C note, http://www.w3.org/TR/NOTE-xml-ql*, 1998.

[9] P. C. Kanellakis and G. M. Kuper and P. Revesz, Constraint Query Languages, *Symposium on Principles of Database Systems*, 299-313, 1990.

[10] W. May, Integration of xml data in xpathlog, *CAiSE Workshop Data Integration over the Web (DIWeb'01)* (Interlaken, Switzerland), 4–5 2001.

[11] OPEN GIS CONSORTIUM, Geography Markup Language (GML), v1.0 ed. 35 Main Street, Suite 5, Wayland, MA 01778 USA, April 2000.

Peter Revesz obtained his B.S. degree, summa cum laude, in Computer Science from Tulane University in 1985, and M.S. and Ph.D. degrees in Computer Science from Brown University in 1987 and 1991, respectively, under the supervision of the late Prof. Paris Kanellakis. He has been a postdoctoral fellow at the University of Toronto during the academic year 1991-92, before joining the University of Nebraska-Lincoln, where he is currently a full professor in Computer Science and Engineering. His research interests are database systems in general, especially constraint and spatio-temporal databases, and XML document management.

Yi Chen obtained his B.S degree in Computer Science from Hunan University in 1997, and M.S degree in Computer Science from Huazhong University of Science and Technology in 2000. He is now working with Dr. Peter Revesz for his Ph.D degree at the University of Nebraska - Lincoln. His research interests are database systems, especially constraint database and XML document management.