XQuery By Example

Daniele Braga Alessandro Campi Stefano Ceri Enrico Augurusa

{braga|campi|ceri|augurusa}@elet.polimi.it

Dipartimento di Elettronica e Informazione, Politecnico di Milano
Piazza Leonardo da Vinci 32, 20133 Milano, Italy

ABSTRACT

XQuery, the standard query language for XML, is gaining popularity among users with a SQL background; indeed, formulating XQuery and SQL queries requires comparable skills. However, this nucleus of programmers is not vast, and the availability of simpler XQuery "dialects" could be valuable for establishing its success.

With this motivation in mind, we designed XQBE, a *visual* dialect of XQuery inspired by QBE (Query by Example). QBE was initially proposed as alternative to SQL and has gained popularity as the language supported by MS Access, currently presented to users with a very limited experience of query languages.

Coherent with the XML data model, XQBE uses one or more hierarchical structures to denote the input XML documents and one structure to denote the output document. Similar to QBE, structures are annotated to express selection predicates; explicit bindings between these structures visualize the input/output mappings.

1. MOTIVATION AND DESIGN PRINCIPLES

The W3C (World Wide Web Consortium) provides two textual languages to query and transform XML documents, XQuery and XSLT respectively [2, 3]. These languages, however, are far too complicated for occasional or unskilled users; they are usually aware of the basics of the XML data model and possibly conscious of the structure of the documents they have to manage. Such basic knowledge should be sufficent to allow them to express simple queries and transformations with a suitably simple language.

XQBE (*XQ*uery *B*y *E*xample) is a user friendly interface, based on a visual query language, developed for addressing a wide spectrum of users, including those with minimal or no computer programming skills.

The success of QBE [4] demonstrates that a visual language is effective in supporting intuitive query formulation when the basic constructs of the language are close to the visual abstractions of the underlying data model. Accordingly, while QBE (a relational query language) is based on the use of tables, XQBE is based on the use of trees, coherent with the hierarchical XML data model. XQBE was designed with both the objectives of being intuitive and of being directly mapped to XQuery, thus representing a GUI capable of running on top of any XQuery engine.

2. XQUERY BY EXAMPLE

XQBE includes most of the expressive power of XPath (except some functions and operators, such as `position()` or `instance of`). XQBE allows for arbitrarily deep nesting of XQuery statements, supports the construction of new XML elements and permits to restructure existing documents. However, its expressive power

```
<bib> <book year="1994">
         <title> TCP/IP Illustrated </title>
         <author> <last> Stevens </last> <first> W. </first> </author>
         <publisher> Addison-Wesley </publisher>
         <price> 65.95 </price>
      </book>
  ... <book year="2000">
         <title> Data on the Web </title>
         <author> <last> Abiteboul</last> <first> Serge </first> </author>
         <author> <last> Buneman </last>  <first> Peter </first> </author>
         <author> <last> Suciu </last>    <first> Dan </first>   </author>
         <publisher> Morgan Kaufmann Publishers </publisher>
         <price> 39.95 </price>
      </book> ... </bib>
```

Figure 1: A sample document - http://www.bn.com/bib.xml

is limited in comparison with XQuery, which is Turing-complete. As an example, XQBE does not support user-defined functions, as we believe that a user confident with this abstraction can directly use XQuery. These limitations are precise design choices: we believe that a complete but too complex graphical language would fail both in replacing the textual one and in addressing most users' needs. The graphical constructs evolved to the actual set due to many experiments, organized with users of several kinds.

XQBE can be considered a successor to XML-GL [1], however with several new features, dictated by the peculiarities of XQuery. To introduce its look and feel and its basics, we show the XQBE version of some simple queries taken from the W3C *XMP* XML Query Use Cases ([2] sec. 1.1.9), based on the fragment of Fig. 1.

2.1 Sample queries

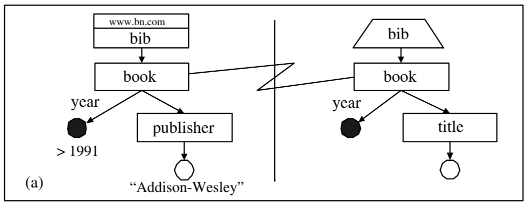
Query q1 (Q1 in [2]) states "*List books published by Addison-Wesley after 1991, including their year and title*". In XQuery:

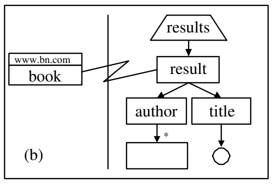
```
<bib>
{ for $b in document("www.bn.com/bib.xml")/bib/book
  where $b/publisher="Addison-Wesley" and $b/@year>1991
  return <book year="{$b/@year}"> {$b/title} </book> }
</bib>
```

The XQBE version is in Fig. 2a. All queries have a vertical line separating the *source* part (on the left) and the *construct* part (on the right, for a "natural" reading order). Both parts contain labelled graphs that represent XML fragments and express properties of such fragments (conditions on values, ordering...): the source part describes XML data to be matched to construct the query result, while the construct part specifies which parts are to be retained and (optionally) which new XML items are to be inserted. Correspondences between input and output data are expressed by *bindings* that cross the vertical line and connect the nodes of the source part to the nodes that will take their place in the output document.

XML *elements* are depicted as labelled rectangles, *attributes* are depicted as black circles (with the name on the incoming arc), and PCDATA contents are depicted as empty circles. All circles (*value* nodes) may be labelled with conditions on the values they represent. In Fig. 2a the source part matches all the `book` elements with a `year` attribute whose value is greater than 1991 and with a `publisher` whose PCDATA equals "Addison-Wesley".

In the construct part, the paths that branch out of a bound node indicate which of its sub-items are to be retained in the output, thus





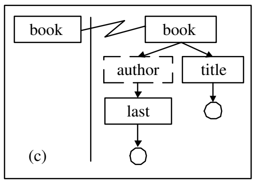


Figure 2: **Sample queries q1 (a), q2 (b) and q3 (c)**

"projecting" the bound element (in q1 only the title and year of the selected `book`s are retained). The binding between the `book` nodes states that the query result shall contain *as many* `book` elements *as* those matched in the source part. The trapezoidal `bib` node represents a newly generated element and means that all the generated `book`s are to be contained into a single `bib` element. *New elements* are always depicted as trapezia, with their short edge on the upper side or on the bottom side; these two modes impose different cardinality constraints on the newly generated elements[1].

2.1.1 *Element projection and renaming*

Query q2 (Q3 in [2]) states "*For each book in the bibliography, list the title and authors, grouped inside a* `result` *element*":

```
<results>
 { for $b in document("www.bn.com/bib.xml")/bib/book
   return <result> { $b/title } { $b/author } </result> }
</results>
```

Here there are no conditions upon values, but only a "projection and renaming" of all the `book` elements (see Fig. 2b): the binding between the `book` and the `result` elements causes the construction of a `result` element for each `book`. The `author` and `title` nodes below `result` will extract the corresponding subelements of `book`s, thus projecting the `book` elements. The unlabelled node below `author` states that all its subelements are to be included in the result, while the asterisk on the incoming arc extends the inclusion to all levels of depth (the closure of the containment relationship).

The previous query projects `book` elements "in breadth", but dealing with trees also requires the ability to project them "in depth" (i.e. to take far descendants of a given element and place them as direct subelements of that element, pruning the elements in the middle). As an example consider a query stating "*For each book list only the title and the surnames of the authors*". In XQuery:

```
  for $b in //book
  return <book> { $b/title } { $b/author/last } </book>
```

where `author` elements are pruned from the generated result: they are represented in XQBE depicted in dashed lines (q3 in Fig. 2c); `last` elements are directly inserted within `book` elements.

2.1.2 *Join between two documents*

Query q4 (Q5 in [2]) constructs a joint catalogue with the data of two documents: "*For each book found at both bn.com and amazon.com, list the title of the book and its price from each source*":

```
<books-with-prices>
{ for $b in document("www.bn.com/bib.xml")//book,
      $a in document("www.amazon.com/review.xml")//entry
  where $b/title = $a/title
  return <book-with-prices>  { $b/title }
      <price-amazon> { $a/price/text() } </price-amazon>
      <price-bn>     { $b/price/text() } </price-bn>
   </book-with-prices>  }  </books-with-prices>
```

[1]When the short edge of the trapezoid is above, one new element is generated to contain all the items corresponding to the nodes reached by the outgoing arcs. When it is below, each such item is contained in a different instance of the newly generated element.

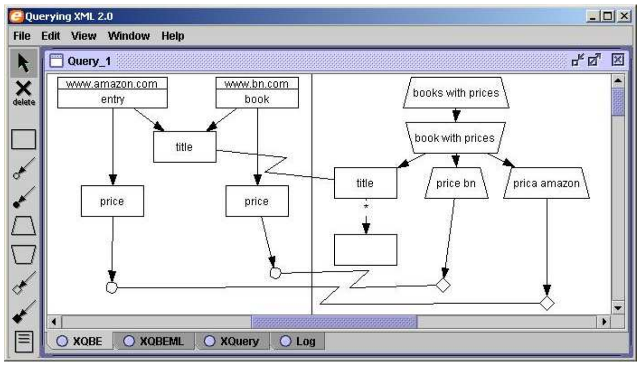


Figure 3: **Query q4 within a snapshot of our interface**

This query performs the "inner join" of the books based on their title. In the XQBE query of Fig. 3 the equality between the values is expressed by means of the confluence into a *single* node, that represents the `title` elements of *both* the `entry` and `book` elements. Confluences can also occur on value nodes, and XQBE allows to label such "join" nodes with binary predicates (<, <=, ! =, ...). Equality is assumed as default if unspecified).

The `price-amazon` and `price-bn` elements are depicted as trapezia because they are new nodes. Their PCDATA content cannot be automatically determined, so the user has to explicitly bind appropriate PCDATA nodes in the match part.

2.1.3 *Advanced transformations*

XQBE allows to express many kinds of transformations with the constructs illustrated so far. Due to space limitations, we only mention the possibility of flattening hierarchical structures, imposing (in the construct part) constraints that do not intervene in the match of the source data, sorting the extracted XML items, expressing negated conditions and computing arithmetic expressions.

2.2 Our implementation of XQBE

We last describe our implementation of XQBE (a snapshot of the tool is in Fig. 3). Graphs are built choosing the graphical constructs from the toolbar on the left, any portion of these graphs can be cut and pasted from a query to another, and the queries can be compiled and executed with a single click. The graphical constructs and the graphs themselves are internally represented as XML data. XQBE queries can thus be saved and exported as XML data.

Once the users complete their queries they can compile them to the corresponding XQuery statement and execute it on any XQuery engine. The translation algorithm first processes the *source* part, so as to compute once for all the variable definitions and the predicative terms that will be assembled into the clauses of the XQuery statement. The statement is then built by a recursive visit of the *construct* part that takes advantage of the pre-computed terms.

The tool assists the user in the editing process with syntactic feedback in many ways, to facilitate the drawing of correct queries. Many incorrect configurations are prevented "on line" by not allowing to connect two nodes or to draw a component where it makes no sense. The syntactic feedback is not limited to "topological" errors, but also makes default corrections to typical or frequent errors, both during the editing process and when the translation is generated.

3. REFERENCES

[1] S. Comai, E. Damiani, and P. Fraternali. Computing graphical queries over xml data. *ACM TOIS*, 19(4):371–430, 2001.

[2] World Wide Web Consortium. XML Query Use Cases, W3C Working Draft, November 2002. http://www.w3.org/TR/xmlquery-use-cases.

[3] World Wide Web Consortium. XQuery 1.0: An XML Query Language, W3C Working Draft. http://www.w3.org/XML/Query, November 2002.

[4] Moshé M. Zloof. Query-by-example: A data base language. *IBM Systems Journal*, 16(4):324–343, 1977.