

Design and Implementation of a Graphical Interface to XQuery

Enrico Augurusa, Daniele Braga, Alessandro Campi, Stefano Ceri
Politecnico di Milano
P.za L. da Vinci 32, I-20133
Milano, Italy

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

ABSTRACT

As the use of XML is rapidly growing, a growing number of users without programming skills will need to query XML data. Although designed to be easily understood by humans, XQuery, the XML standard query language, has the typical syntax of programming languages, which most users dislike. In this paper we describe a graphical language (XQBE) inspired by “Query By Example” (QBE), a popular relational query language used by MS Access. XQBE covers a significant subset of XQuery and is supported by a prototype enabling the formulation of queries on a graphical interface and their translation into XQuery, thus providing non-trivial querying capabilities to a wide spectrum of users. Simple queries are easily represented in XQBE, but many “complex” queries allow as well for an intuitive graphical representation.

1. INTRODUCTION

The widespread use of XML in many applicative domains sets a strong need for providing XML query capabilities to the widest audience, including those users who lack in computer programming skills. This paper describes our effort in designing a user friendly and intuitive visual interface for this purpose.

The World Wide Web Consortium (W3C) provides two textual languages to formulate XML queries and express document transformations, XQuery [19] and XSLT [18] respectively. However, they are both rather cryptical for occasional or unskilled users, who only know the basics of the XML data model; in principle, however, this only prerequisite should be sufficient to allow for basic usability of an XML query language.

As demonstrated by the QBE paradigm [20], the major benefit of a visual language is supporting of an intuitive query formulation, based on a paradigm where the user is asked to express queries in the format of a given data model.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC 2003 Melbourne, Florida, USA

Copyright 2003 ACM 1-58113-624-2/03/03 ...\$5.00.

While QBE is a relational query language, based on the use of tables, XQBE (*XQuery By Example*) is based on trees¹, so as to adhere to the hierarchical XML data model.

XQBE was designed with the two-fold objective of being intuitive, according to the aforementioned principles, and of being easily and directly translatable into XQuery, so that a GUI could be implemented on top of an existing XQuery engine. XQBE includes most of the expressive power of XPath², covers queries that require nested FLWR expressions, supports the construction of new XML elements and allows for restructuring existing documents (for example, by modifying the containment relationships).

The expressive power of XQBE, however, is limited in comparison with that of XQuery, which has been proved to be Turing-complete [13]. Nevertheless we believe that a QBE language is useful when it serves the needs of the majority of the users in expressing the majority of their queries, which are typically simple. As an example, XQBE does not include the *if-then-else* construct, as we believe that a user capable to handle such a construct can directly formulate a textual query; moreover there is an intrinsic difficulty in expressing alternatives by means of graphical constructs. More in general, a complete but too complex graphical language would fail both in replacing the textual language and in addressing all the users' needs.

In this paper Section 2 presents the language by means of a sequence of scaled examples, so as to introduce the simpler constructs first. Section 3 compares XQBE with XML-GL, its main predecessor. Section 4 presents a prototype of a GUI integrated with a translator. Related activities and contributions are discussed in Section 5, while Section 6 concludes the paper and points to open issues and future work.

2. XQUERY BY EXAMPLE

In order to demonstrate the features of XQBE we show the graphical representation of some queries based on the XML fragment of Figure 1, describing the catalog of a supermarket augmented with the locations of each product.

¹Although in its full capability XML supports cyclic references (with ID/IDREF couples), its practical use is most often associated with trees (rather than with cyclic graphs).

²Some of the XPath core functions (such as `position()`) and operators (such as `instance of`) are not covered. Disjunction capabilities are limited as well.

```

<supermarket>
  <item id="3456">
    <description> desktop </description>
    <location> <corridor> Informatics </corridor> <shelf> 27 </shelf> </location>
    <supplier> IBM </supplier> <price> 1000 </price>
  </item>
  <item id="3457">
    <description> Printer </description>
    <location> <corridor> Informatics </corridor> <shelf> 56 </shelf> </location>
    <supplier> Canon </supplier> <price> 125 </price>
  </item>
  <item id="3678">
    <description> Brush </description>
    <location> <corridor> Dress </corridor> <shelf> 45 </shelf> </location>
    <location> <corridor> ... </corridor> <shelf> 97 </shelf> </location>
    <supplier> HWER </supplier> <price> 39.95 </price>
  </item>
</supermarket>

```

Figure 1: Running example - <http://www.sm.com/sm.xml>

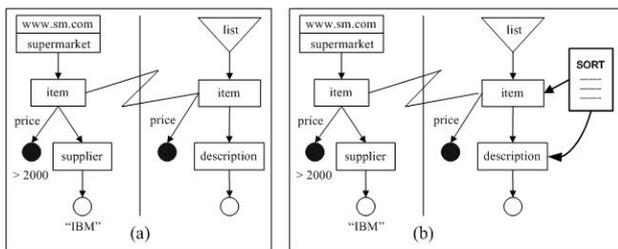


Figure 2: Queries with match conditions and sorting

2.1 Simple queries

The first query, q_1 states “List items supplied by IBM with a price greater than 2000, including their id and description”. Here follows the query in XQuery, while its graphical version is depicted in Figure 2.

```

<list>
{for $p in document("http://www.sm.com/sm.xml")/supermarket/item
 where $p/supplier = "IBM" and $p/price > 2000
 return <item id="{ $p/id }">{ $p/description }</item> }
</list>

```

This first example allows us to describe most of the basic syntax and semantics of XQBE. A query always has a vertical line in the middle, that separates the *source* part (the one on the left) from the *construct* part (that on the right). Both parts contain labelled graphs that represent XML fragments and express properties of such fragments (conditions upon values, ordering properties, etc.). More specifically, the source part describes the XML data to be matched in order to construct the query result, while the construct part specifies which parts are to be retained in the result and (optionally) allows the user to introduce newly generated XML items. In this way the query has a “natural” reading order from left to right. The correspondence between the components of the two parts is expressed by means of explicit binding edges 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.

All the XML *element* nodes are depicted as labelled rectangles, their *attribute* nodes are depicted as black circles (with the attribute name on the arc between the black circle

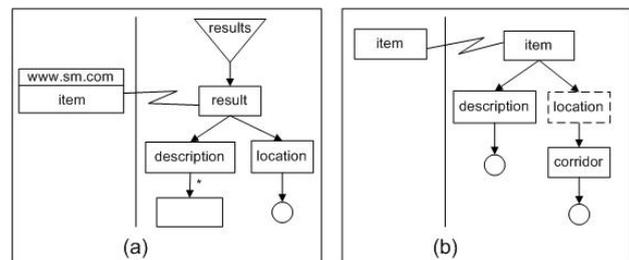


Figure 3: Renaming and breadth/depth projecting

and the rectangle), and their PCDATA content is always depicted as an empty circle. The black and empty circles (the only nodes that represent values) are named *value* nodes. Value nodes may be labelled, so as to express conditions on the values they represent.

In Figure 2a the source part matches all the *item* elements in document <http://www.sm.com/sm.xml> that contain a *price* element whose PCDATA content is greater than 1991 and a *supplier* whose PCDATA content equals “IBM”.

The binding edge between the *item* nodes states that the query result shall contain *as many item* elements as those matched in the source part. In the construct part, the paths that branch out of a bound node indicate which of their sub-items are to be retained (thus “projecting” the bound node). In q_1 only description and publication year of the selected *items* are retained. The triangular node above the *item* node means that all the generated *items* are to be contained into a single *list* element. This node represents a newly generated element, and *new* elements are always depicted as triangles in XQBE.

If we want to sort the generated *item* elements in lexicographic order, XQuery requires the addition of a *sortby* clause to the previous query:

```

... return <item id="{ $p/@id }"> { $p/description } </item>
      sortby (description) } ...

```

Accordingly, in the graphical representation we just need to change q_1 in q_2 (depicted in Figure 2b), by adding a *SORT* node (depicted in bold).

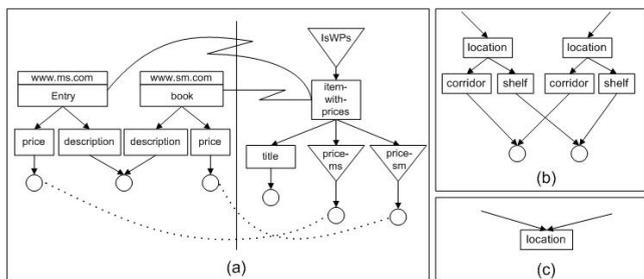


Figure 4: Join between two documents (q5)

2.2 Element projection and renaming

We now show a query that states “For each item, list description and location, grouped inside a result element”, and translates to:

```
<results>
{for $p in document("http://www.sm.com/sm.xml")/supermarket/item
return <result> { $p/description } { $p/location } </result> }
</results>
```

Here there are no conditions upon values, but only a “projection” with “renaming” of all the `item` elements (q3 in Figure 3a): the binding edge between the `item` element and the `result` element causes the construction of a `result` element for each `item`. This is a simple renaming of the elements: the `location` and `description` elements below `result` will extract the corresponding subelements of `item`, thus projecting the element bound to the one they will be contained in. The unlabelled node below `location` states that all its subelements are to be included in the generated result, while the asterisk on the incoming arc extends the inclusion to any level of depth. This configuration is therefore used to synthetically include entire fragments in the result.

The previous query projects the `item` elements “in breadth”, but dealing with trees also requires the ability to project them “in depth” (i.e. to take a far descendent of an element and put it as one of its direct subelements, pruning the elements in the middle). As an example consider a query that states “For each item list only the description and the corridors where it is located (maintaining the items in the order of the original document)” and maps to the following XQuery statement:

```
for $p in //item
return <item> { $p/description } { $p/location/corridor } </item>
```

In this case the `location` elements are pruned from the generated result, and are thus depicted in the construct part with dashed lines (q4 in Figure 3b); `corridor` elements are directly inserted into the `item` elements.

2.3 Join between two documents

Query q5 constructs a joint item catalogue, collecting information from different documents. It states “For each item found at both `sm.com` and `ms.com`, list the description of the item and its price from each source”:

```
<items-with-prices>
{ for $p in document("www.sm.com/sm.xml")//item,
  $a in document("www.ms.com/sm.xml")//item
  where $p/description = $a/description
  return <item-with-prices>
    { $p/description }
```

```
<price-sm> { $p/price/text() } </price-sm>
<price-ms> { $a/price/text() } </price-ms>
</item-with-prices> }
</items-with-prices>
```

This query performs the “inner join” of the items based on their description. In the XBQE query of Figure 4a the equality between the values is expressed by means of the confluence into a *single* value node, that represents the PCDATA content of *both* the `description` elements. More generally, the language allows to specify conditions by labelling such “confluence” nodes with binary predicates (<, <=, !=, ...), while the equality predicate is assumed as default.

The `price-sm` and `price-ms` elements are depicted as triangles because they are new nodes. The PCDATA content of these nodes cannot be automatically inferred, so the user has to explicitly bind their content to the appropriate PCDATA nodes in the match part. This bindings are represented with dotted lines, so as to distinguish them from the “z-shaped” edges that impose the “as many as” constraint.

The join in the previous query is based on a single value, but XQBE provides a useful and intuitive shorthand for all the cases in which equality has to be imposed on arbitrarily complex fragments. If we consider a join based on the `location`, an exhaustive representation would be that of Figure 4b, while the compact notation supported by XQBE is that of Figure 4c (where one node stays for eight). The general principle states that whenever a confluence occurs on a rectangular node this requires that the entire fragments that originate from that node be equal³.

3. COMPARISON WITH XML-GL

The basic ideas and most of the syntax of XQBE derive from XML-GL [4], an early and self-standing visual query language for XML, designed far before XQuery. However, the peculiarities of XQuery imposed us to revise or prune some constructs of XML-GL, while some other constructs have been introduced from scratch. Several queries, which are very easily expressible with XQuery (and with XQBE) are not expressible with XML-GL; in addition, the semantics has significantly changed in going from XML-GL to XQBE, in order to facilitate the equivalence with (and translation into) XQuery.

XQBE has a major expressive power in comparison with XML-GL. For example, consider again query q4, described in Section 2.2. This query is very easily expressible both in XQuery and XQBE (see Figure 3b), but it is not expressible in XML-GL, because of the ordering requirement. Moreover, XML-GL does not provide the capability to project “in depth” the extracted fragments, nor that of specifying conditions in the RHS of the queries.

For some other queries XQBE is much simpler than XML-GL. For example, assume to have a different DTD, defining this book elements containing the title, a list of authors or a list of editors, a (possibly empty) sequence of reviewers’ remarks, a publisher and a price:

```
<ELEMENT book ( title, ( author+ | editor+ ),
  reviewer*, publisher, price ) >
<ELEMENT reviewer ( last, first?, remark ) >
```

³The semantics of this *deep equality* is the same as that of the = operator in XQuery, when applied to nodes with complex content.

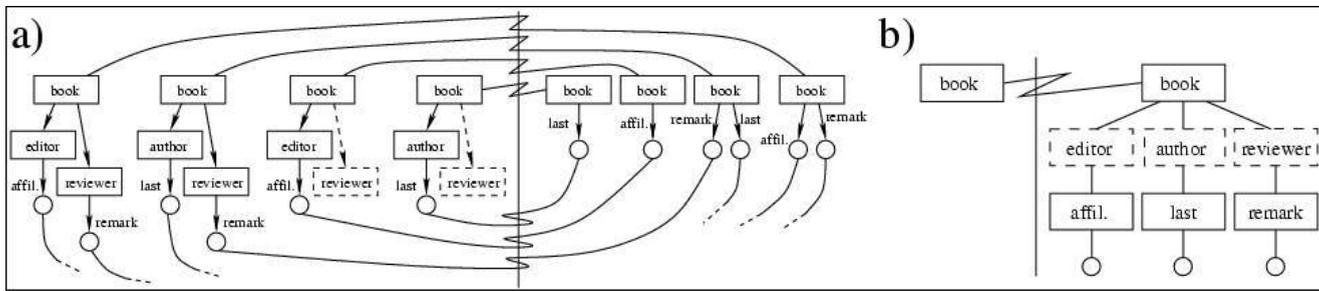


Figure 5: Comparison between XML-GL and XQBE

Then the following (quite simple) query cannot be easily and compactly expressed in XML-GL:

```
for $b in //book
return <book> { $b/editor/affiliation }
           { $b/author/last }
           { $b/reviewer/remark } </book>
```

The tricky point is that all `book` elements should contain the (possibly empty) sequence of editor affiliations, the (possibly empty) sequence of authors' surnames and the (possibly empty) sequence of reviewers' remarks.

The correct query in XML-GL is that of Figure 5a, but XQBE allows for a much simpler and more compact representation (Figure 5b), because it allows us to factorize the constraints in a breadth/depth projection. Since the `editor`, `author`, and `reviewer` elements are not to be retained, they cannot be depicted in the RHS, neither can they be depicted all together as the descendants of a "shared" `book` in the LHS, because this would match only those books that *do have* at least one element of each kind (an impossible condition, since `editors` and `authors` are mutually exclusive). Four different cases must therefore be distinguished, taking into account the optional presence of some reviewers and the alternative presence of authors or editors.

4. IMPLEMENTATION OF THE INTERFACE AND THE TRANSLATION

In the visual interface that we have implemented (a snapshot is shown in Figure 6), queries are displayed in a window composed of two parts (source and construct). The query graphs are built using the buttons listed in the toolbar on the left, and portions of graphs can be cut and pasted from a query to another.

Further graphic features, such as automatic working areas detection, non-overlapping controls and grids, help the users. While the user is drawing a query, an on-line syntactic feedback prevents some "topological" errors. Thus, the tool offers a rapid way to generate and manage the XQBE query components.

The graphical constructs and the graphs themselves are internally represented as XML data. XQBE queries can also be saved and exported as XML data.

The translation is based on giving to all XQBE queries a canonical (XML-based) representation and then generating the corresponding XQuery code. It consists of three main steps:

- check of the syntactic correctness of the query: all errors that cannot be detected during the drawing pro-

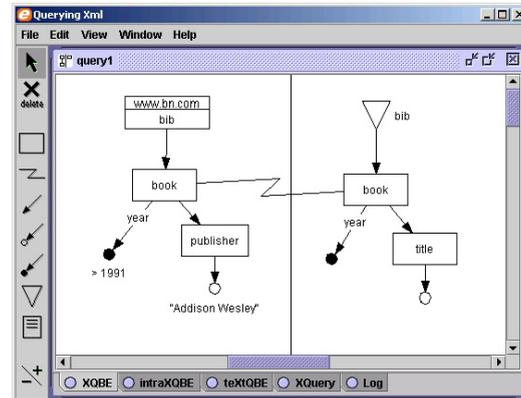


Figure 6: A snapshot of our interface

cess are now detected; if they are fatal errors the process is aborted, otherwise default correction policies are adopted and warnings are generated;

- identification of variables and conditions: the source part is parsed in order to individuate those graphical configurations that require variables in the XQuery translation (bindings, confluences, multiple ramifications, ...), while both the source and construct parts are parsed to extract the predicates.
- assembling the query: a recursive visit of the construct part starts from the root nodes and descends the structure along the directed arcs. The XQuery statement is built by assembling the previously generated conditions and by nesting FLWR expressions whenever a binding edge is encountered. Such expressions are generated according to the analysis of appropriate subgraphs of the match part, reached following the current binding edge.

The user launches the translation process when the query is complete: the generated XQuery statement is shown and can be executed on any XQuery engine. Intermediate results of the translation process can be accessed as well.

5. RELATED WORK

Several XML textual query languages have been proposed and extensively analyzed in the literature [8, 10] before the proposal of XQuery [19] as a standard. XQBE comes after a long stream of research on graph-based logical lan-

guages. The ideas in this field started years ago with the QBE paradigm [20].

Usually in graphical query languages, such as Graphlog [5], Good [16] or G-Log [11], graphs are patterns to be “matched” against a graph-based representation of the target data. The ancestors of these languages were G [6] and G+ [7]. Graphlog [5] is a direct descendant of G+. Good [16] proposed a uniform notation for alternative models within object databases where: nodes represent objects and edges represent relationships. G-Log [11] is a logic-based graphical language that uses a Good-like notation for representing and querying complex objects by means of directed labelled graphs. This language evolved into WG-Log [17] to query internet pages and semi-structured data, by adding to G-Log some hypermedia features.

XML-GL [4] is a direct descendent of WG-Log. This is a graph-based language that allows for a natural representation of complex queries over XML data. The user can express queries containing joins among several input documents, arithmetic and aggregate functions, union, difference, and cartesian product. It also allows for the construction of new XML items.

QSBY (Query Semi-structured data By Example [9]) is a graphical interface that represents data as nested tables and extends the QBE paradigm to deal with semi-structured data.

MiroWeb Tool [1] uses a visual paradigm based on trees that implements XML-QL. QBEN is a graphical interface to query data according to the nested relational model; the users specify their queries with the operations of the nested relational algebra [12].

Equix [2] is a form-based query language for XML repositories, based on a tree-like representation of the documents, automatically built from their DTDs. Intra-document relationships cannot be visually rendered. Equix has limited restructuring capabilities. In [3] a new syntax for Equix is proposed, more user-friendly but limited to searching the Web.

BBQ [15] (Blended Browsing and Querying) is a graphical user interface proposed for XMAS [14], a query language for XML-based mediator systems (a simplification of XML-QL).

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a graphical query language that offers a visual interface to query XML documents. We showed how a user can express queries or restructure documents in a natural and intuitive way exploiting the well-understood metaphor of the XML fragments as trees. This paper is moving ahead of the W3C in defining a visual interface to XQuery; we hope that this contribution may stimulate academic and industrial research in a field that we deem fundamental to the success of XQuery for a wider audience.

We have also briefly presented a prototype that implements our proposed interface. This tool translates graphical queries into XQuery, so that it can be used in conjunction with any XQuery engine.

There are several potential opportunities for future work. From a technical viewpoint, several improvements and extensions to our work are possible, and among the various uncovered features of XQuery we will give priority to adding

support for the XML typing system, extending the set of supported core functions and adding the capability of querying the document order.

Acknowledgements

In this work, Daniele Braga and Alessandro Campi are supported by the “Virtual Campus” Microsoft Research Grant.

7. REFERENCES

- [1] L. Bouganim, T. Chan-Sine-Ying, Tuyet-Tram Dang-Ngoc, J. L. Darroux, G. Gardarin, and F. Sha. Miro web: Integrating multiple data sources through semistructured data types. In *Proc. of 25th International Conf. on Very Large Data Bases (VLDB'99)*, Edinburgh, Scotland, UK, pages 750–753. Morgan Kaufmann, Sept. 1999.
- [2] S. Cohen, Y. Kanza, Y. A. Kogan, W. Nutt, Y. Sagiv, and A. Serebrenik. Equix easy querying in XML databases. In *WebDB (Informal Proceedings)*, pages 43–48, 1999.
- [3] S. Cohen, Y. Kanza, Y. A. Kogan, W. Nutt, Y. Sagiv, and A. Serebrenik. Combining the power of searching and querying. In *Fifth International Conf. on Cooperative Information Systems*, Sept 2000.
- [4] S. Comai, E. Damiani, and P. Fraternali. Computing graphical queries over xml data. *ACM TOIS*, 19(4):371–430, 2001.
- [5] Mariano P. Consens and Alberto O. Mendelzon. The graphlog visual query system, 1990.
- [6] Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. A graphical query language supporting recursion, 1987.
- [7] Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. G+: Recursive queries without recursion. In *Second Int. Conference on Expert Database Systems*, pages 355–368, 1988.
- [8] M. Fernandez, J. Siméon, P. Wadler, S. Cluet, A. Deutsch, D. Florescu, A. Levy, D. Maier, J. McHugh, J. Robie, D. Suciu, and J. Widom. Xml query languages: Experiences and exemplars. 1999.
- [9] Irna M. R. Evangelista Filha, Alberto H. F. Laender, and Altigran S. da Silva. Querying semistructured data by example: The qsbeye interface.
- [10] Z. G. Ives and Y. Lu. Xml query languages in practice: an evaluation. In *WAIM'00*, 2000.
- [11] P. Peelman J. Paredaens and L. Tanca. G-log a declarative graph-based language. *IEEE Trans. on Knowledge and Data Eng.*, 1995.
- [12] G. Jaeschke and H. J. Schek. Remarks on the algebra on non first normal form relations. In *Proc. of 1st ACM SIGACT-SIGMOD Symposium on the Principles of Database Systems*, pages 124–138, 1982.
- [13] S. Kepser. A proof of the turing-completeness of xslt and xquery. Technical report SFB 441, Eberhard Karls Universitat Tubingen, May 2002.
- [14] B. Ludaescher, Y. Papakonstantinou, P. Velikhov, and V. Vianu. View definition and dtd inference for xml. In *Proc. Post-IDCT Workshop*, 1999.
- [15] K. Munroe, B. Ludäscher, and Y. Papakonstantinou. Blended browsing and querying of xml in a lazy mediator system, March 2000.
- [16] J. Paredaens, J. Van den Bussche, M. Andries, M. Gemis, M. Gyssens, I. Thyssens, D. Van Gucht, V. Sarathy, and L. V. Saxton. An overview of good. *SIGMOD Record*, 21(1):25–31, 1992.
- [17] R. Posenato S. Comai, E. Damiani and L. Tanca. A schema based approach to modeling and querying www data. In *FQAS'98*, May 1998.
- [18] World Wide Web Consortium. Extensible Stylesheet Language (XSL) Version 1.0, W3C Recommendation. <http://www.w3c.org/TR/xsl/>, October 2001.
- [19] World Wide Web Consortium. XQuery 1.0: An XML Query Language W3C Working Draft. <http://www.w3.org/XML/Query>, June 2001.
- [20] M. M. Zloof. Query by example: A database language. *IBM Systems Journal*, 16(4), 1977.