A Graphical XQuery Language Using Nested Windows

Zheng Qin, Benjamin Bin Yao, Yingbin Liu, and Michael McCool

University of Waterloo
School of Computer Science, Waterloo, Ontario, Canada N2L 3G1
{zqin, bbyao, ybliu, mmccool}@uwaterloo.ca

Abstract. A graphical XQuery-based language using nested windows, GXQL, is presented. Definitions of both syntax and semantics are provided. Expressions in GXQL can be directly translated into corresponding XQuery expressions. GXQL supports `for`, `let`, `where`, `order by` and `return` clauses (FLWOR expressions) and also supports predicates and quantifiers. This graphical language provides a powerful and user-friendly environment for non-technical users to perform queries.

1 Introduction

XML is now being used extensively in various applications, so query languages have become important tools for users from many different backgrounds. However, the use of query languages can sometimes be difficult for users not having much database training. A graphical query language can potentially be very helpful. With a graphical interface, users do not have to remember the syntax of a textual language, all they need to do is select options and draw diagrams.

In this paper, a graphical XQuery-based language is described. Early graphical query languages for XML included G [7], G+ [8], G+'s descendant Graphlog [6], G-Log [11], WG-Log [4], and WG-Log's descendant XML-GL [3,5]. In these visual languages, a standard node-edge graphical tree representation is used to visualize the hierarchical structure of XML documents. The nodes represent elements and attributes in the documents, and the edges represent relationships between the nodes. Research has also been performed into form-based query languages, such as Equix [2], and nested-table based query languages, such as QSByE (Query Semi-structured data By Example) [9]. The BBQ language used a directory tree visualization of the XML tree [10].

Most of these visual languages were developed before XQuery. A recent graphical XQuery-based language, XQBE (XQuery By Example) [1], extends XML-GL to XQuery, and also overcomes some limitations of XML-GL. The XQBE query language is good at expressing queries, but there are some problems with it. First, XQBE defines many abstract symbols. For instance, there are two kinds of trapezoids, lozenges of two different colors, circles of two different colors, and so on. It is difficult to remember which abstract symbol represents what concept. Second, all relationships are mapped onto a uniform tree structure. This is also true of other systems otherwise similar to ours (such as BBQ).

Representing all relationships with a common visual formalism can lead to confusion. For instance, when a node points to another node via an edge, does it mean a parent-child relation, a cause-result relation, or an attribute relation? Third, there are some XQuery expressions that cannot be easily expressed by XQBE, for example, quantifiers.

We have designed a nested window XQuery-based language, called GXQL (Graphical XQuery Language). GXQL has fewer symbols than XQBE, and these symbols are visually suggestive. We use nested windows to represent parent-child relationships. Child elements and attributes are also visually distinguished. The visualization of a document in GXQL in fact resembles a real document. The query interface in GXQL is user-friendly. Users do not have to input everything textually or need to draw queries from scratch. Like BBQ, in our visual notation windows and icons can be dragged around to construct new nodes or copy nodes. The interface also allows users to visualize only the parts of the document structure they need to perform a query. GXQL is also more expressive than XQBE. Some queries hard to express in XQBE are easy in GXQL, and some queries impossible to express in XQBE are possible in GXQL. For instance, in XQBE predicates of a node in a `return` clause can only affect its parent node, whereas in GXQL, predicates can affect arbitrary nodes.

Since we want to compare GXQL directly with XQBE, the sample XML document and the example queries in this paper are taken from the XQBE paper [1]. Due to space limitations only two simple examples are included here; the rest of the examples are available in a technical report [12].

2 Visualization Interface

The schema of the sample document we will be using for our example queries is represented by GXQL in Figure 1 (a). Each rectangle represents an element that can have a URI, attributes and subelements. The URI indicates the location of the document. In the sample document, element `<bib>` is at the outermost level, and element `<book>` includes attribute `year` and some children.

Rectangles representing children are enclosed completely in their parent rectangle. The borders of these rectangles can be drawn in various styles. These will be explained in the next section.

Initially, only the parent node and its immediate children are represented. However, users can expand elements inline by double clicking on them. Already expanded elements can be zoomed to fill the window by double-clicking on them again. When an attribute is expanded, information about that attribute, such as its data type, will be added to the representation. When an element is expanded, it will remain the same width but will get longer, and its attributes and children will be drawn nested inside it. If an element is zoomed, its corresponding rectangle and all its children will zoom out to fill the window. If the window is not big enough to display all its elements, a scroll bar will be added to the window and users can scroll to view all the elements. Right clicking on an attribute or element will pop up a right click menu. Choosing the "`predicate`" menu item

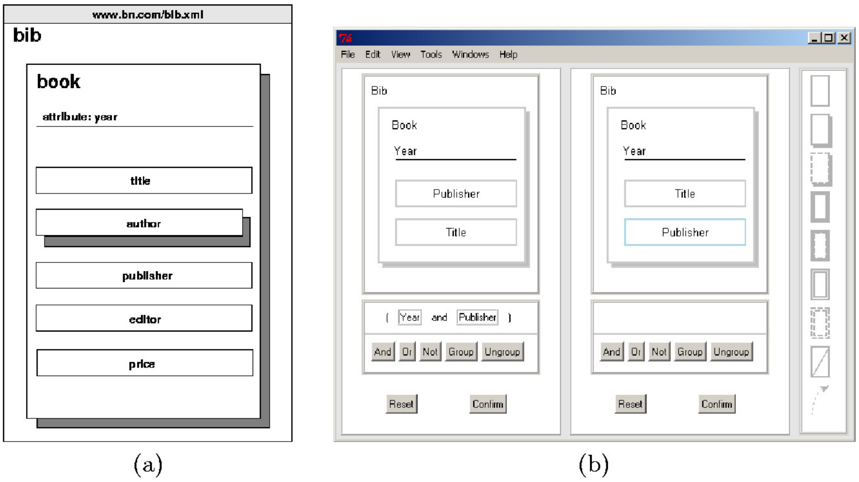

Fig. 1. (a) GXQL representation of the sample document. (b) Query interface of GXQL. Retrieval is on the left, construction is on the right.

will bring up a window showing information (such as name, type and full path) about that attribute or element and allows the entry of predicates.

Drag actions are also used as part of the query interface, but these are distinguished from the clicking actions described above because in a drag action, the button up event happens outside the window.

3 Query Interface

The query interface of GXQL looks like Figure 1 (b). There are three parts in the main window. On the left, the retrieval pane represents the schema or input document. It allows users to select the subset of the input they want to query. In the middle, the construction pane allows users to structure the query results. On the very right of the interface there is a symbol bar containing all the symbols used in GXQL. These are used to create new elements from scratch in the construction pane.

In the retrieval pane, when users choose a document or document schema, GXQL will visualize its structure. At first, only the outermost node and its children are shown, but users can zoom into or expand subelements to see detail. We chose this design because we want the interface to give users some way to browse the document structure, so they do not have to remember the names of elements and attributes, but we do not want to overwhelm them with detail. Our design also allows large documents to be visualized incrementally.

We will call elements or attributes “nodes”. Users can select (or deselect) any node by left clicking on it. Selecting nodes by clicking avoids errors caused by misspelling. By default, all nodes are first drawn with a light blue color indicating that they exist, but have not been selected yet. Selecting nodes will change their color to black. After users set up a query, clicking on the "`confirm`" button executes the query. All selected nodes will participate in the query, while unselected elements will be ignored.

When users want to input predicates for nodes, they need to right click on a node. A window will pop up asking for the predicate, and will provide a menu of options. After the predicates are confirmed, each predicate will be shown in a panel. Both the retrieval pane and construction pane have their own predicate panel.

In the construction pane, there are two ways to construct a node: either by dragging a symbol from the symbol bar, or by dragging a node from the retrieval pane. After dragging a symbol from the symbol bar, the new element is empty, and the user must input the name for it. When dragging a node from the retrieval pane, the node (including all its descendants) are dragged into the construction pane, forming a new node there. Users can then select the nodes they want or delete (by a right click menu selection) the ones not needed. Users can also drag the nodes around and switch their order. The results will be given based on this order. The frame border of nodes can also be changed via a right click menu.

Some rectangles have single-line frames and some have shadowed frames. Other frame styles are possible; a complete set of symbols representing the relations between nodes used in GXQL is given in Figure 2. Each frame style has a specific meaning suggested by its visual design. Symbol 1 indicates that node B is the single immediate child of node A. Symbol 2 indicates there are multiple B subelements wrapped within one A node, and all Bs are immediate children of A. Symbol 3 has the same meaning as symbol 2, except when users set up predicates for B, only some elements B satisfy the predicates. Symbol 4 indicates that the B subelements are descendants of A. There may be multiple Bs that are descendants of A. They do not have to be immediate children of A. Symbol 5 has the same meaning as symbol 4, except that when users set up predicates for B, only some elements B satisfy the predicates. Symbol 6 indicates that the B subelements are descendants of A with only one intermediate path in between. There may be multiple Bs that are descendants of A. Symbol 7 has the same meaning as symbol 6, except that when users set up predicates for B, only some elements B satisfy the predicates. Symbol 8 has the same meaning as symbol 1, except that when users set up predicates for B, they want the complement of the results. This is just one example of complementation. Any symbol from 1 to 7 can be complemented in the same way.

4 Examples

The sample XML document and the example queries used in this paper are taken from the XQBE paper [1]. We are going to show how two of these queries, 1 and 5, are expressed in GXQL, with modifications in Query 5 to demonstrate queries not supported by XQBE. The rest of the queries are demonstrated in our technical report [12].

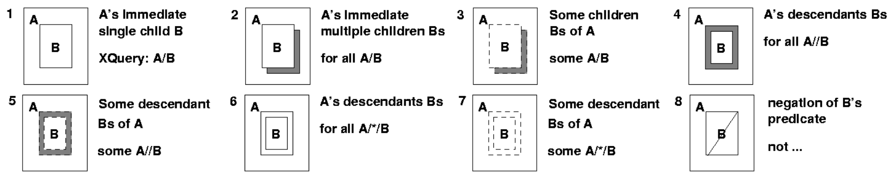
1 A B A's immediate single child B XQuery: A/B
2 A B A's immediate multiple children Bs for all A/B
3 A B Some children Bs of A some A/B
4 A B A's descendants Bs for all A//B
5 A B Some descendant Bs of A some A//B
6 A B A's descendants Bs for all A/*/B
7 A B Some descendant Bs of A some A/*/B
8 A B negation of B's predicate not ...

Fig. 2. Symbols used in GXQL.

Query 1: *List books published by Addison-Wesley after 1991, including their year and title.*

This query shows how to represent "`for`" "`where`" and "`return`" in GXQL. In the XQuery textual language, this query can be expressed as follows:

```
<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>
```

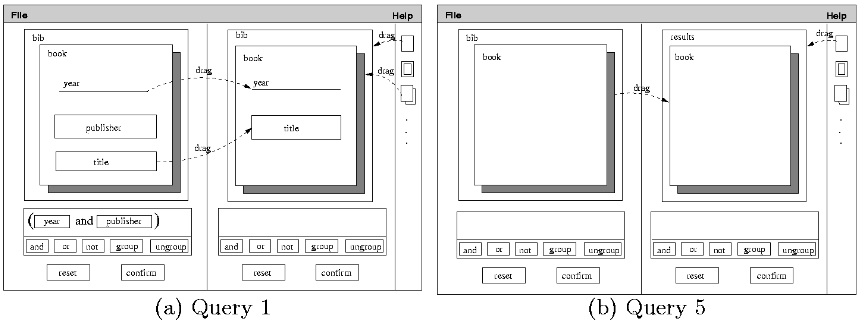

Fig. 3. GXQL expressions for queries 1 and 5.

Query 1 is represented by GXQL as in Figure 3 (a). In the retrieval pane, users first zoom into `<book>`, so the attribute `year` and all subelements will show up. Right clicking on `year` will pop up a window. This window will show the name (with full path) and data type of the attribute and will prompt for predicates. Once a predicate is set, the predicate object will show up in a predicate panel below the main figure. Predicates can be combined together by Boolean operations. All Boolean operations are supported, such as `or` and `not`. This cannot be done in XQBE, which can only represent `and` relations.

To express the example query, in the construction pane users first drag an icon with a single frame from the symbol bar to create a new element `<bib>`, then drag an icon with a shadowed frame for element `<book>`. Then users can drag `year` and `<title>` from the retrieval pane to the construction pane. When the "`confirm`" button is clicked, appropriate textual XQuery language will be generated and passed down to the processing pipeline. The query should be read from the outermost rectangle toward the innermost rectangles.

Query 5: List all the books in element `<bib>` and wrap them within one `<results>` element.

We modified this example so that it uses the `let` clause. The `let` clause is not supported in XQBE, so there is no example using `let` in the XQBE paper. XQBE can wrap multiple elements within a single element, but the query is always translated into `for` clause. The modified XQuery is given as follows:

```
let $b := document("www.bn.com")/bib/book,
return <results> { $b } </results>
```

Query 5 is represented by GXQL in Figure 3 (b). In this example, the `<book>` element is first dragged from the retrieval pane to the construction pane. Note that the `<book>` rectangle has a shadowed frame. This means all the retrieved `<book>` elements will be wrapped together in one `<results>` element in the result.

5 Semantics

To implement a query in GXQL, we have to translate a given GXQL diagram into a corresponding XQuery FLWOR expression. In the construction pane, when users set up rectangles by dragging icons from the symbol bar, it corresponds to constructing new nodes in the result. In the retrieval pane, each shaded double-line frame, if not dragged to the construction pane, corresponds to a "`for`" clause with a "`//`" path, e.g. "`for $b in bib//book`". If such a frame is dragged to the construction pane, it corresponds to a "`let`" clause, e.g. "`let $b = //book`", and the result of "`$b`" is wrapped within a single parent tag. Each double-line unshaded frame works the same way as shaded double-line frame, except that it represents the path "`/*/`". Each shadowed frame, if not dragged to the construction pane, also corresponds to a "`for`" clause with a path containing only "`/`", e.g. "`for $b in bib/book`". If such a frame is dragged to the construction pane, it corresponds to a "`let`" clause, e.g. "`let $b = /bib/book`", and the result of "`$b`" is wrapped within a single parent tag. Each single-line frame corresponds to a child "`/`", e.g. "`$f = bib/book`". If a frame has a dashed border, it corresponds to use of the "`some`" quantifier, e.g. "`some $b in //book satisfies`". If a rectangle is crossed, it corresponds to the use of "`not`" in all predicates, e.g. "`not ($b = ‘‘Jack’’)`".

So to perform translation, the construction pane should be analyzed first to find out what nodes are new and which nodes are copied from the retrieved results. The next step is to analyze the retrieval pane, going from the outermost rectangle to the innermost rectangle and binding variables to expressions according to how they are going to be used in the "`return`" clause. The last step is to construct FLWOR expressions based on the construction pane.

6 Conclusions

In this paper, we have described the design of GXQL, a graphical query language using nested windows to visualize hierarchy. Representations in GXQL can be directly translated into corresponding "`FLWOR`" clauses. GXQL also supports predicates, different path patterns, and quantifiers. GXQL is also easy to expand to support more XQuery features.

More features of XQuery might eventually be supported in GXQL, such as conditional expressions, type casting, functions, and so on. However, being both powerful and clear is a challenge to graphical languages. The system should not have so many features added to it that it becomes too difficult for a user to learn. For future work, we need to complete the implementation and perform user testing to validate our design.

We would like to thank Frank Tompa for suggesting that we submit this paper for publication. He also suggested the notation for negation.

References

1. D. Braga and A. Campi. A Graphical Environment to Query XML Data with XQuery. In *Fourth Intl. Conf. on Web Information Systems Engineering (WISE'03)*, pp. 31–40, 2003.
2. S. Cohen, Y. Kanza, Y. A. Kogen, W. Nutt, Y. Sagiv, and A. Serebrenik. Equix Easy Querying in XML Databases. In *WebDB (Informal Proceedings)*, pp. 43–48, 1999.
3. S. Comai, E. Damiani, and P. Fraternali. Computing Graphical Queries over XML Data. In *ACM Trans. on Information Systems, 19(4)*, pp. 371–430, 2001.
4. S. Comai, E. Damiani, R. Posenato, and L. Tanca. A Schema Based Approach to Modeling and Querying WWW Data. In *Proc. FQAS*, May 1998.
5. S. Comai and P. di Milano. Graph-based GUIs for Querying XML Data: the XML-GL Experience. In *SAC, ACM*, pp. 269–274, 2001.
6. M. P. Consens and A. O. Mendelzon. The G+/GraphLog Visual Query System. In *Proc. ACM SIGMOD*, 1990, pp. 388.
7. I. F. Cruz, A. O. Mendelzon, and P. T. Wood. A Graphical Query Language Supporting Recursion. In *Proc. ACM SIGMOD*, 1987, pp. 323–330.
8. I. F. Cruz, A. O. Mendelzon, and P. T. Wood. G+: Recursive Queries without Recursion. In *2nd Int. Conf. on Expert Database Systems*, pp. 335–368, 1988.
9. I. M. R. Evangelista Filha, A. H. F. Laender, and A. S. da Silva. Querying Semistructured Data by Example: The QSByE Interface. In *2nd Int. Conf. on Expert Database Systems*, pp. 335–368, 1988.
10. K. D. Munroe and Y. Papakonstantinou. BBQ: A Visual Interface for Integrated Browsing and Querying of XML. In *5th IFIP 2.6 Working Conf. on Visual Database Systems*, 2000.
11. P. Peelman, J. Paredaens and L. Tanca. G-log: A Declarative Graph-based Language. In *IEEE Trans. on Knowledge and Data Eng.*, 1995.
12. Z. Qin, B. B. Yao, Y. Liu and M. McCool. A Graphical XQuery Language Using Nested Windows. Technical Report CS-2004-37, School of Computer Science, University of Waterloo, August, 2004.