A Graphical XQuery Language Using Nested Windows

Technical Report: CS-2004-37

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 supported by a user-friendly interface 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. In addition to the node-edge representation, 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, especially for non-technical users. Second, the representation is not visually obvious, as 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? Users who are unfamiliar with XML or XQuery cannot tell at first sight. Third, there are some XQuery expressions that cannot be easily expressed by XQBE, for example, quantifiers. XQBE could be extended to support these features, but such an extension would involve even more abstract symbols, edges and labels, which would make the representation harder to understand and use.

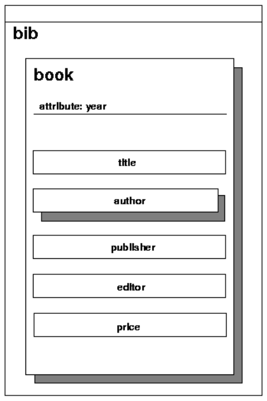
We have designed a nested window XQuery-based language, called GXQL (Graphical XQuery Language). GXQL has fewer symbols than XQBE, and these symbols are visually suggestive and therefore easier to remember. We use nested windows to represent parent-child relationships. Child elements and attributes are also visually distinguished. The visualization of a document in GXQL in fact resembles a real document in appearance. 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. These will be shown in the examples.

In Section 2, we will show how GXQL visualizes the structure of XML documents. Section 3 describes how queries can be set up using the user interface of GXQL. Examples to show how this language is used are given in Section 4. 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]. Some modifications were made to Query 5 to demonstrate a query not supported by XQBE. The semantics of GXQL are analyzed in Section 5. Conclusions and future work are given in Section 6.

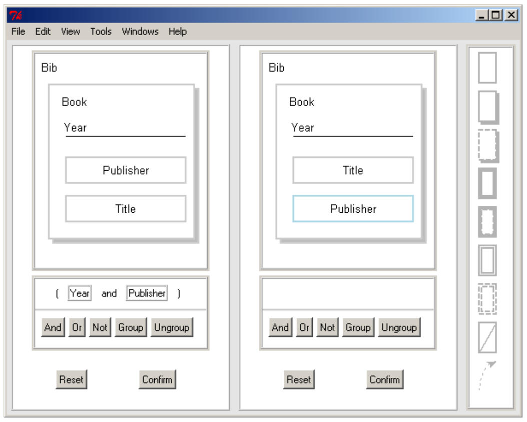
2 Visualization Interface

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, element `<book>` includes attribute `year` and some children.

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



(a)



(b)

Fig. 1. Visualization Interfaces. (a) GXQL representation of the sample document (b) Query interface of GXQL

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 "`predicate`" item 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. Expanded or zoomed elements will have a close icon in the upper right corner. Left clicking on this icon will close that element (unzoom it or unexpand it as required). We have not shown these icons in our diagrams to avoid clutter.

Drag actions are also used as part of the query interface, but these are distinguished from the actions described here 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. This process can be repeated until the nodes of the innermost level (with no children) are reached. We chose this design because we want the interface to give users some way to browse the structure of the document, so that users do not have to remember the names of elements and attributes themselves, but are not overwhelmed with detail. Our design also does not require loading the entire document, so large documents can 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 light blue color indicating the nodes exist, but have not been selected yet. Selecting nodes will change their color to black. After the users set up a query, clicking on the "`confirm`" button will confirm and execute 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 just need to right click on a node, and a window will pop up asking for the predicate, and providing a menu of options. After the predicates are confirmed, each predicate will be shown in the predicate panel below as an object. Both retrieval pane and construction pane have their own predicate panel.

In the constructions 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 along into the construction pane, forming a new node there (expanded to the same level as in the retrieval pane). 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. When nodes are dragged from the left part, their frame border can also be changed via a right click menu item.

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 meanings 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 meanings 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. Symbol 9 is used in our diagrams to indicate actions in the user interface. It is not a visual symbol in GXQL.

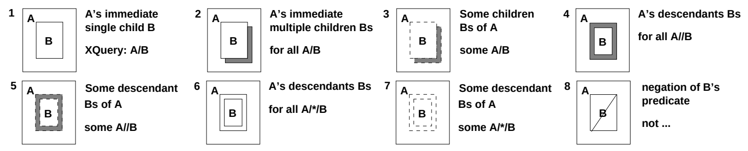

1 A B A's immediate single child B XQuery: A/B
2 A B A's immediate multiple children Bs for all A/B
3 A B Some children Bs of A some A/B
4 A B A's descendants Bs for all A//B
5 A B Some descendant Bs of A some A//B
6 A B A's descendants Bs for all A/*/B
7 A B Some descendant Bs of A some A/*/B
8 A B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

Fig. 2. Symbols used in GXQL

4 Examples

The sample XML document and the example queries used in this paper are taken from the XQBE paper [1] with modifications in Query 5 to demonstrate queries not supported by XQBE. We are going to show how each of these queries is expressed in GXQL. We will elaborate the first example, but quickly go through the rest of the queries.

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

This query is to show 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>
```

Query 1 is represented by GXQL as in Figure 3 (a). In the retrieval pane, users first zoom into element `<book>`, so the attribute `year` and all subelements will show up. Right clicking on `year` to pop up a window as in Figure 4 (a). This window will show the name (with full path) and data type of the attribute and will prompt for predicates. The symbol ">" is chosen from a cascade menu. For the predicate, user can either input it manually or drag a node from the document. Right clicking on the `<publisher>` element to pop up a window as in Figure 4 (b) will show the name (with full path) and data type of the content of this element, and will ask for predicates. Once a predicate is set, the predicate

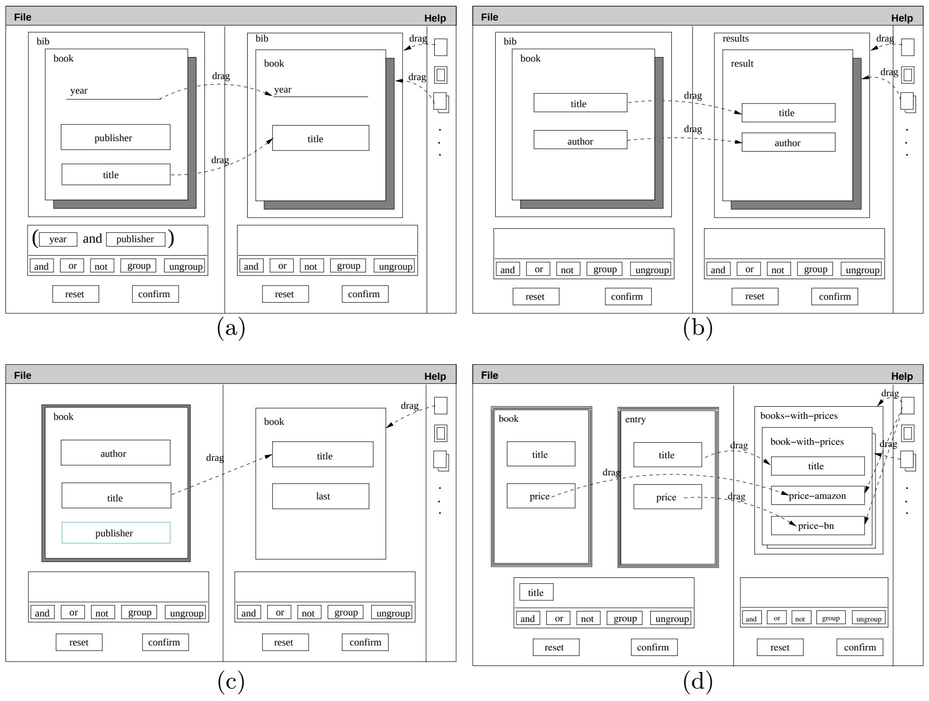


Fig. 3. GXQL expressions for queries 1 through 4

object will show up in the predicate panel below the main figure as shown in Figure 4 (c). Predicates can be combined together by boolean operations. For example, first the user clicks on `and` button and then clicks on objects `year` and `<publisher>`. The predicates for `year` and `<publisher>` will be combined by `and`. They are also grouped together (represented by the parenthesis around them) so that the grouped object can participate in further boolean operations. The user can also ungroup the predicates by the `ungroup` button. The predicate objects can be dragged around to switch orders, which supports arbitrary combinations of the predicates. All boolean operations are supported, such as `or` and `not`. This cannot be done in XQBE, which can only represent `and` relation, and this is why there are only `and` relations in their examples.

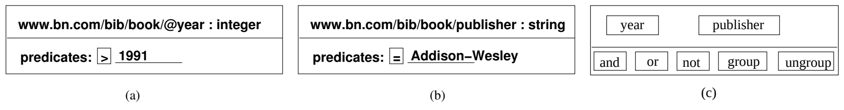

www.bn.com/bib/book/@year : integer
predicates: > 1991
(a)
www.bn.com/bib/book/publisher : string
predicates: = Addison−Wesley
(b)
year publisher
and or not group ungroup
(c)

Fig. 4. Pop-up windows for attributes and elements

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 shadowed frame for element `<book>`. Then users can drag `year` and `<title>` from the retrieval pane to the construction pane (left to right). 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. In the retrieval pane, elements `<bib>` and `<book>` at the outermost level are interpreted as "for each `<book>` element in `<bib>`". The predicates for `year` and `<publisher>` are interpreted to mean "the `year` has to be later than 1991 `and` the `<publisher>` has to equal `Addison-Wesley`". In the construction pane, elements `<bib>` and `<book>` are interpreted to mean "construct new nodes `<bib>` and `<book>`, with multiple `<book>` elements wrapped inside a single `<bib>` element." The attribute `year` and element `<title>` are copied from the retrieved results.

Query 2: *For each book in the bibliography, list the title and authors, grouped inside a `<result>` element.*

This query shows that we can construct new element with new names. This query is expressed in XQuery as follows:

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

Query 2 is represented by GXQL as in Figure 3 (b). In the left part, the frame for element `<book>` has a shadow, which means there are multiple instances of element `<book>` within a single pair of `<bib>` tags. We iterate over each instance of the `<book>` element in the input document, which corresponds to the `for` clause in XQuery. In the construction pane, dashed lines with arrowheads show how the elements are constructed. The shadowed frame of element `<result>` means multiple instances of `<result>` will be wrapped in one `<results>` tag. Each `<result>` instance will have a copy of the `<title>` and an `<author>` elements from each `<book>` in the input.

Query 3: *For each book, list only the title and the surnames of the authors (maintaining the books in the order of the original document).*

This query shows how to choose multiple descendants of arbitrary depth. This query is expressed in XQuery as follows:

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

Query 3 is represented by the GXQL given in Figure 3 (c). In the retrieval pane, the frame of `<book>` has shaded double lines, which means the query is looking for any `book` element in the document, corresponding to path `//book` in XQuery (frame options are chosen by right clicking on the frame). When this frame mode is selected, double clicking will cause zooming to operate "in place", allowing users to burrow down through the document tree.

This path cannot be represented clearly by XQBE. XQBE uses Figure 6 (a) to represent the descendant path. However, it is hard to tell whether this indicates `bib/book` or `bib//book`. For this reason, in the examples used by the XQBE paper, the "`//`" path specifier is not used except at the very beginning of the path. Even when XQBE example uses "`//`" at the beginning of a path, we feel the XQBE representation is confusing as in the second half of Figure 6 (a). When the users choose `<last>` in `<author>`, the `<author>` rectangle will expand, and users can drag the `<last>` element to the construction pane.

Query 4: *For each book found at both `bn.com` and `amazon.com`, list the title of the book and its price from each source.*

This example shows how to join two documents to produce a new document. The corresponding XQuery is as follows:

```
<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>
         { $a/title }
         <price-amazon> { $a/price/text() } </price-amazon>
         <price-bn>     { $b/price/text() } </price-bn>
         </book-with-prices> }
</books-with-prices>
```

Query 4 is represented by GXQL in Figure 3 (d). In the retrieval pane, users first right click on the "`<title>`" node in the first document, which will pop up a window. Then users need to drag the "`<title>`" node in the second document into the predicate, which will be converted into the path automatically. The rest of the construction pane is then built as previously discussed.

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 5 (a). 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.

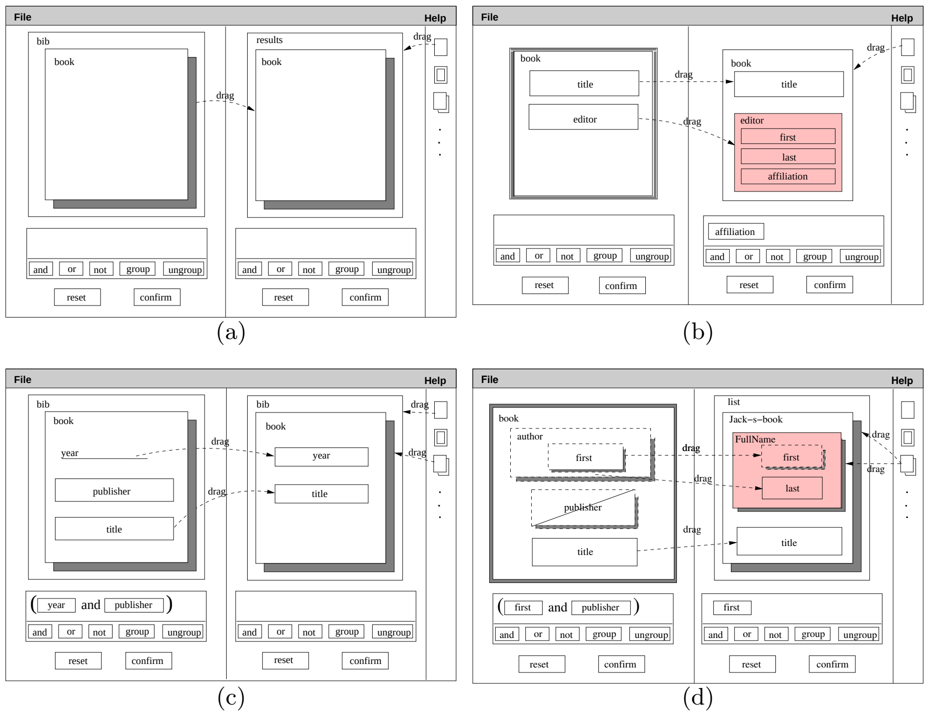


Fig. 5. GXQL expression of queries 5 through 8

Query 6: Make a list of all the books with their title, including the editors only if they are affiliated to CITI.

This query shows how to represent queries with predicates in the return clause. This query is expressed in XQuery as follows:

```
for $b in //book
return <book>
       { $b/title } { $b/editor[affiliation="CITI"] }
       </book>
```

Query 6 is represented by GXQL as in Figure 5 (b). The predicate is on the construction pane in this case. This is because the predicate must be in the return clause of the XQuery. Putting the predicate in the retrieval pane would filter off some results that we need. The pink color in the construction pane indicates that the predicate will affect all the elements that have the same color as the predicate element. In this case, element `<editor>` will be affected by the predicate. If, for example, only element `<first>` would be affected, then we would just dye the `<first>` rectangle pink.This relationship cannot be represented well in XQBE either. XQBE uses Figure 6 (b) to represent this relationship. The double border around `affiliation` indicates that it is an attribute. The $*$ indicates that all

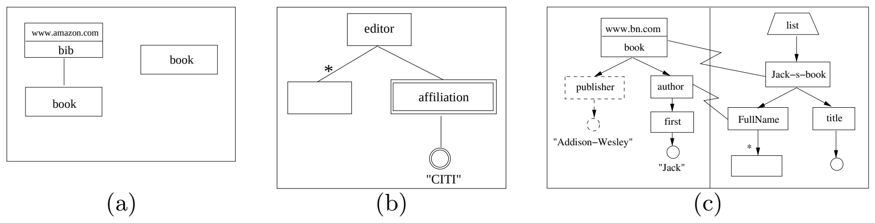


Fig. 6. XQBE examples. (a) Part of query 3 (b) Part of query 6 (c) XQBE expression of query 8

subelements of `editor` are selected. However, predicates in XQBE always affect only the parent element, so XQBE cannot express predicates when only `<first>` or `<last>` is affected.

Query 7: *List books published by Addison-Wesley after 1991, including their year and title, sorting the retrieved books in lexicographic order.*

This example shows how the "`order by`" clause can be invoked. The corresponding XQuery is given by

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

Query 7 is represented by GXQL as in Figure 5 (c). Setting up the sorting order of `<book>` works the same way as setting up predicates, except that it won't show up in the predicate panel.

Query 8: *List all the books not published by Addison-Wesley and with an author whose first name is "`Jack`". Rename each of these books in `<Jack-s-book>`, and only remain the title and the full name of the authors whose first name is `Jack`.*

The equivalent XQuery is given by the following:

```
<list>
{ for $b in document ("www.bn.com/bib.xml")//book
  where some $a in $b/author satisfies
         some $f in $a/first/text() satisfies
          ( $f = "Jack" and
          not ( some $p in $b/publisher/text() satisfies
           ($p = "Addison-Wesley" ) ) )
  return <Jack-s-book>
         { for $a in $b/author
```

```
          where some $f in $a/first/text() satisfies ( $f = "Jack")
          return <FullName> { $a/* } </FullName> }
        { $b/title }
        </Jack-s-book>
</list>
```

Query 8 is represented by GXQL as in Figure 5 (d). Remember if an element has a predicate and its frame border has a dashed line, it means we need only some of the elements to satisfy the predicates.

Users can read the query from the diagram as followings: In the retrieval pane, the `<book>` element has double-line frame, which means we are looking for all book elements in path "`//book`". The `<author>` element has both a dashed and a shadowed frame, which means multiple author elements are inside each book element, and we need only some of them to satisfy their predicates. The `<first>` rectangle is also shadowed, which means there are multiple `<first>` elements (to represent multiple given names) in each author element, and we need only some of them to satisfy the predicate. The `<publisher>` elements are set up similarly, except that we want the negation of a predicate, i.e. "not (some publisher elements satisfy the predicate)". In the construction pane, `<list>`, `<Jack-s-book>`, and `<FullName>` are new elements. The `<FullName>` element is pink, which means for the authors' name of each book retrieved from the document, we only want to list the names that have at least one first element that is equal to "`Jack`". In XQBE, they use the graph in Figure 6 (c) to represent the same query. Here the dashed line indicates negation. From the way the `<author>` and `<first>` nodes are drawn, it is really hard to tell whether it means all of them have to satisfy the predicates or only some of them have to.

5 Semantics

To implement a query in GXQL, we just have to translate a given GXQL diagram into a corresponding XQuery FLWOR expression. We are going to explain how each symbol in GXQL is translated into a clause in FLWOR, which also defines the semantics of GXQL. 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 frame unshaded works the same way as shaded double-line frame, except that it represents 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 complete FLWOR expressions based on the elements in 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. To keep the user interface clear, GXQL does use textual input for simple expressions. We think textual expressions are a much easier way to express simple computations.

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.

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.