Journal of King Saud University – Computer and Information Sciences (2011) **23**, 59–70

King Saud University

Journal of King Saud University – Computer and Information Sciences

www.ksu.edu.sa
www.sciencedirect.com

ORIGINAL ARTICLE

A survey of graphical query languages for XML data

Mourad Ykhlef *, Sarra Alqahtani

King Saud University, College of Computer and Information Sciences, Saudi Arabia

Received 3 November 2009; accepted 27 October 2010
Available online 8 May 2011

KEYWORDS

XML;
Graphical query language;
Semi-structured data model;
Querying and restructuring

Abstract XML is fast emerging as the dominant standard for information exchange on the World Wide Web. The ability to intelligently query XML data becomes increasingly important. Several query languages are proposed in the literature, some of them are textual query languages and some others are graphical query languages. In this article, we will survey some of graphical query languages for querying and restructuring XML data. Also, we will conduct a comparative study between these languages based on a set of requirements and features that an XML query language should have.

1. Introduction

eXtensible Markup Language (XML) was created to address limitations of the Hypertext Markup Language (HTML). XML has simple and flexible textual formats to meet the challenges of analyzing and proposing large-scale electronic data. It is used to store, present, and exchange a wide variety of data on the web. XML is now being used extensively in various applications, so query languages have become important tools for users from many different backgrounds. The XML query languages have developed in two kinds, textual and graphical. The W3C (World Wide Web Consortium) provides two textual languages to query XML data, XSLT (Kepser, 2002; Wide Web Consortium, 2001) and XQuery (World Wide Web Consortium, 2001). Many graphical query languages were proposed during the last years. A graphical query language can potentially be very helpful for users, with a graphical language, users do not have to remember the syntax of a textual language, all they need to do is to select options and draw diagrams. Each graphical language has its unique characteristics, strengths and weaknesses. In this article, we present the most popular graphical languages XML-GL (Ceri et al., 1999), GLASS (Ni et al., 2003), XQBE (Braga et al., 2003) and GQLX (Ykhlef and Alqahtani, 2009) through showing their data models, query languages, set of examples, their features and limitations.

The rest of the article is organized as follows. Section 2 discusses XML data modeling. Sections 3–6 provide overviews of XML-GL, GLASS, XQBE and GQLX languages, respectively. Section 7 presents some other graphical query languages. In Section 8 we will conduct a comparison between graphical languages based on a set of requirements and features that an XML query language should have.

* Corresponding author.
E-mail addresses: ykhlef@ksu.edu.sa (M. Ykhlef), Sarra.alqahtani@gmail.com (S. Alqahtani).

Peer review under responsibility of King Saud University.
doi:10.1016/j.jksuci.2011.05.002

Production and hosting by Elsevier

2. XML data modeling

Query languages for semi-structured and structured data rely on data models, which are abstract notations for representing the organization of data. XML document could be directly assumed as the data model for a query language or the language may rely on a special model. Here, we discuss four models for XML data.

2.1. XML-GDM data model

XML-GDM (Ceri et al., 1999) is denoted for (XML Graphical Data Model). It is used to express both the expected structure of XML documents (i.e., their DTDs) and actual documents. The data model has a graphical representation in which syntax translation produces graphical schemas of DTDs or of documents.

The XML-GDM data model consists of three concepts: *object*, *properties*, and *relationships*:

1. *Objects*: represented as rectangles, indicate abstract items without a directly representable value.
2. *Properties*: represented as circles connected to the object they refer to, indicate representable values (e.g., a character data); properties have a name and a type.
3. *Relationships*: represented as arcs between objects and indicate semantic associations (e.g., containment or reference). Relationships have a direction from a source object to a destination object (Ceri et al., 1999).

The correspondence between an XML DTD and an XML document and an XML-GDM graph is established by the following rules (Ceri et al., 1999).

1. Each non-terminal element E is mapped to an XML-GDM object with the same name as E.
2. Between any two non-terminal elements E1 and E2 such that element E1 is a sub-element of E2, a relationship is established from the object that represents E2 to the object that represents E1.
3. When a terminal element E1 is a sub-element of another element E2, it is mapped to a property of E2 named E1, with PCDATA type.
4. Element disjunction (|): if a non-terminal element E contains sub-elements E1 ... En that are in "exclusive or", i.e., only one of them can be present in E, then an arc is drawn which crosses the relationships between E and E1, ..., En labeled "xor".
5. Each printable attribute and object-identifier of an element E is mapped to a property of the object that represents E, with the same name and type of the XML attribute. For distinguishing XML attributes from nested elements, the small circle of the property is bold.
6. Element order: the actual or required order of appearance of sub-elements in a super-element is represented by ordering the arcs that represent the containment relationships counter-clockwise, starting from the arc corresponding to the first sub-element, which is marked by a small trait.

To illustrate these rules and the XML-GDM data model, consider the XML document in Fig. 1 (Ykhlef, 2007). A representation of this document in XML-GDM data model is shown in Fig. 2.

```
<bib>
        <book year="1988">
                <title>Principles of Database and Knowledge Base Systems</title>
                <author>
                        <name>Jeff Ullman</name>
                </author>
                <publisher>W.H. Freeman Company</publisher>
                <isbn>0716781581</isbn>
        </book>

        <book year="1995">
                <title>Foundations of Databases</title>
                <author>
                        <name>Serger Abiteboul</name>
                </author>
                <author>
                        <name>Rick Hull</name>
                </author>
                <author>
                        <name>Victor Vianu</name>
                </author>
                <publisher>Addison-Wesley</publisher>
                <isbn>0201537710</isbn>
        </book>

        <article>
                <title>Querying Semi-structured Data</title>
                <author>
                        <name>Serge Abiteboul</name>
                </author>
                <proceedings>ICDT</proceedings>
        </article>
</bib>
```

Figure 1 XML data of bibliography (source: Ykhlef, 2007).

2.2. ORA-SS data model

The ORA-SS (Object-Relationship-Attribute model for Semi-Structured data) is a rich semantic data model for semi-structured data (Gillian et al., 2009). Besides reflecting the nested structure, it also distinguishes between object classes, relationship types and attributes in ORA-SS. Furthermore, the ORA-SS specifies the participation constraints of object classes in relationship types and indicates whether an attribute belongs to an object class or a relationship type. Fig. 3 shows the ORA-SS diagram for the bibliographic data in Fig. 1.

2.3. XQBE data model

In the XQBE (XQuery By Example) data model (Braga et al., 2003), all the XML elements in the target document are depicted as labeled rectangles, their attributes are depicted as black circles with the attribute name on the arc between the rectangle and the circle, and their PCDATA content is always depicted as an empty circle. The black and empty circles are named value nodes. See Fig. 4.

2.4. G-XML data model

An XML document is modeled by an ordered labeled rooted graph called G-XML (Ykhlef and Alqahtani, 2009) where:

1. Each arc is labeled by an XML element or XML attribute.
2. The attribute arc is dashed while the child arc is directed.
3. Each leaf node is labeled with value.
4. G-XML has a distinguished node called the root.

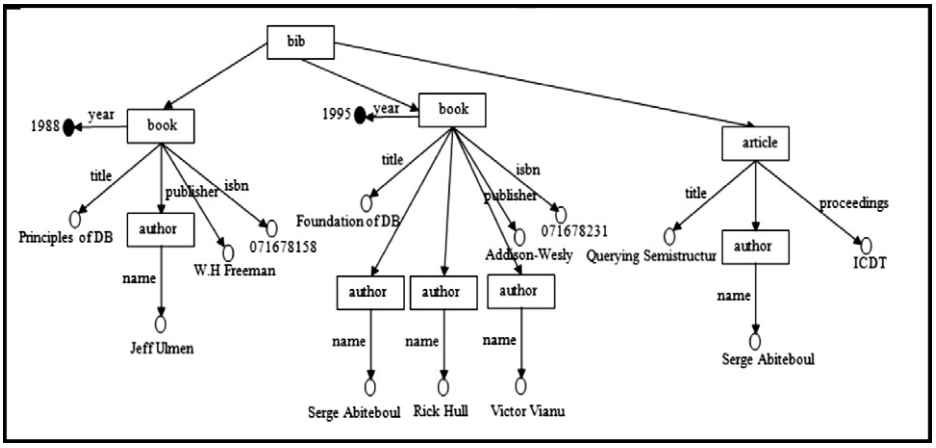

Figure 2 XML-GDM of bibliographic data.

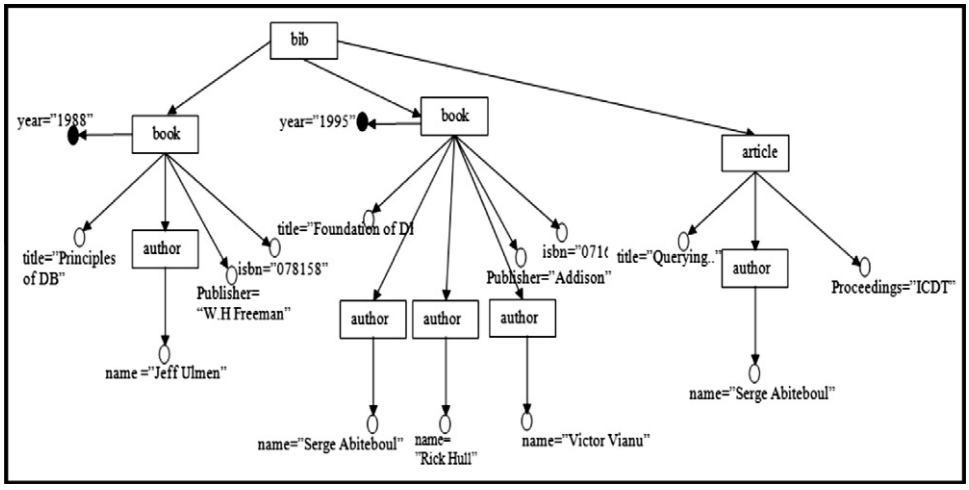

Figure 3 ORA-SS of bibliographic data.

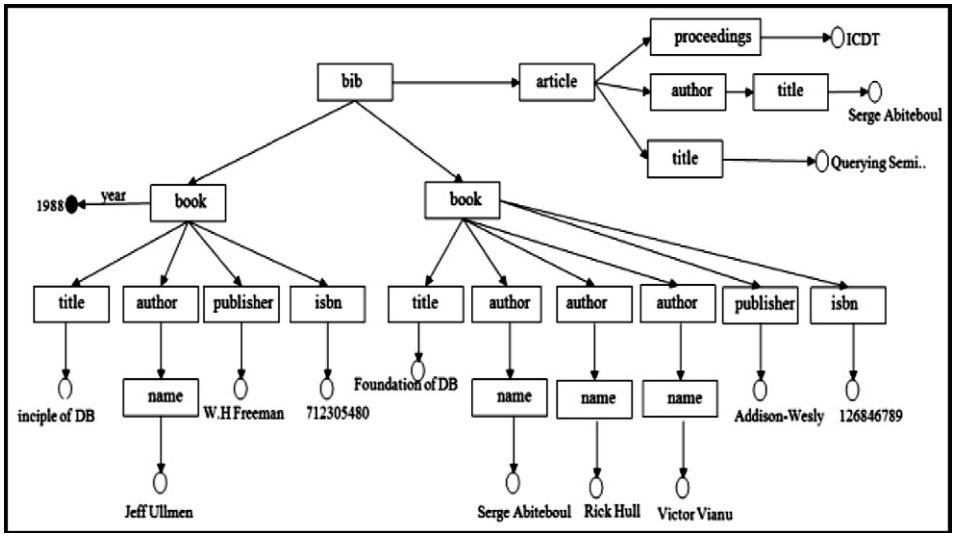

Figure 4 XQBE bibliographic data model.

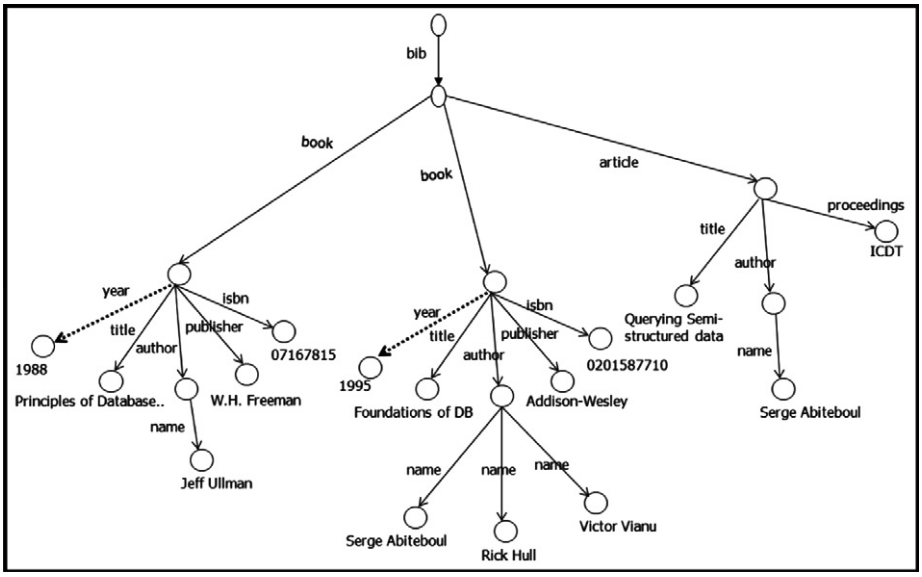

Figure 5 G-XML of bibliographic data.

For example, the XML document of Fig. 1, giving information on bibliographic data, is represented by the G-XML graph of Fig. 5. Remark that the arc labeled by title represents an element tag but the dashed arc labeled by year represents an attribute tag. Attributes are alternative ways to represent data.

3. Querying and restructuring XML data by XML-GL language

XML-GL is denoted for XML graphical language. The goal of designing XML-GL was to create a formal way to deal with both XML and XML queries graphically. It relies on XML-GDM data model. This language is able to express complex queries. All of its elements are displayed visually; therefore, XML-GL is suitable for supporting a user-friendly interface (Ceri et al., 1999).

3.1. XML-GL query

An XML-GL query can be applied either to a single XML document or to a set of documents. The query produces a new XML document as the result. Thus, the execution of a query results in a transformation of the source XML document(s) into a new XML document. Graphically, an XML-GL query is a pair of XML-GDM graphs, displayed side by side and separated by a vertical line; the left-side graph visually represents the extract and match parts, while the right-side graph conveys the clip and constructs parts (Ceri et al., 1999). The following sections present some examples of XML-GL queries.

3.2. Basic query operators

Example 1 (*Simple projection*). Based on bibliographic XML-GDM in Fig. 2 this query returns the title of all books (Fig. 6).

The left hand side (LHS) graph contains the extract part of the query which operates on the single target element *book* that has a sub-element *title*. The right hand side (RHS) graph expresses the clip part, which defines the DTD of the result document as an XML-GDM graph. In XML-GL, the correspondence between LHS and RHS objects is by name (as for

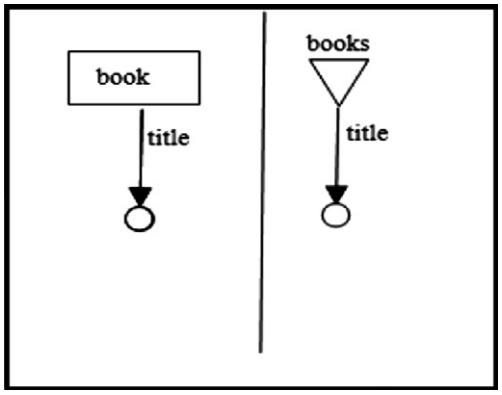

Figure 6 Projection in XML-GL.

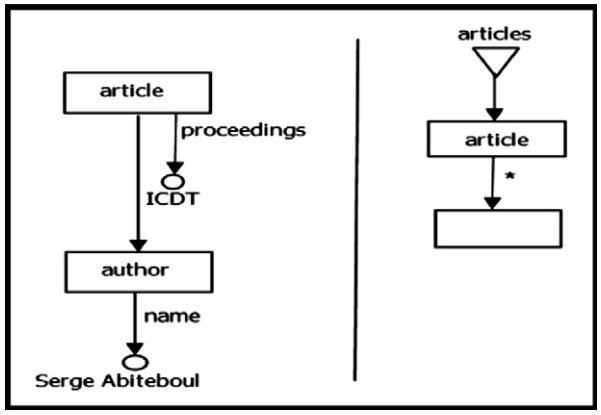

Figure 7 Selection in XML-GL.

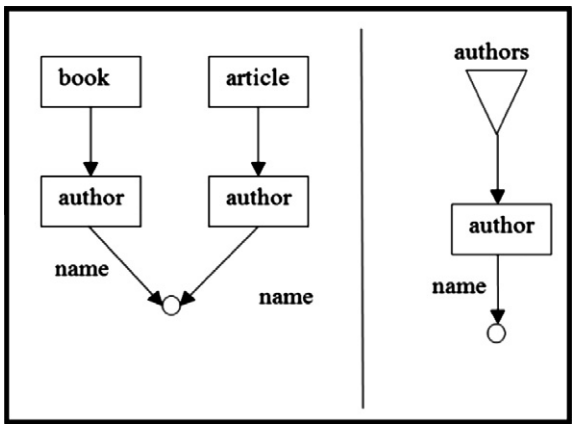

Figure 8 Complex selection in XML-GL.

object *title*). The triangle named of *books* used mainly in XML-GL to environ all titles inside one element tagged with *books*.

Example 2 (*Simple selection*). The query in Fig. 7 returns all articles written by Serge Abiteboul and appeared in ICDT proceedings.

In XML-GL, the match part extends the LHS of the query with logical conditions. Conditions of this query involve two sub-elements (*proceedings, name*) of the target element which is *article*.

Example 3 (*Complex selection*). The query in Fig. 8 returns authors who have written books and articles.

In this query, the extract part (LHS) has two target elements from the same target document *book* and *article*. Pointing to the same element (*name*) is using to express equality between these targets.

Example 4 (*Join*). Let us consider the XML-GDM graph of Fig. 2 and let us consider the following yellow XML data file giving addresses and phone numbers of people. Query in Fig. 9 returns the address and the phone number of each article author.

```
< persons >
  < person >
    < name > Serge Abiteboul < /name >
    < address > INRIA-France < /address >
    < phone > + 33 1 72 92 59 18 < /phone >
  < /person >
< /persons >
```

In this query, the join condition on name is expressed in the match part by expanding the graph of the *book* element to show the inner *author* element, and connecting the author's and person's names. In the result, one element *result* is constructed to environ the whole result of the extract-match part.

3.3. Extended query operators

Example 5 (*Grouping*). For each author, return the list of his/her publications (Fig. 10).

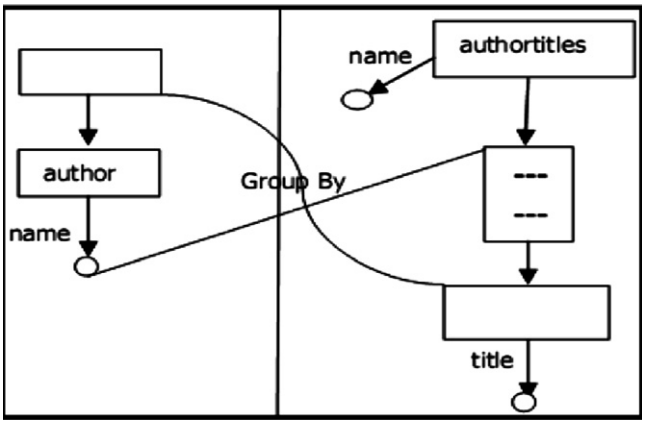

Figure 10 Grouping in XML-GL.

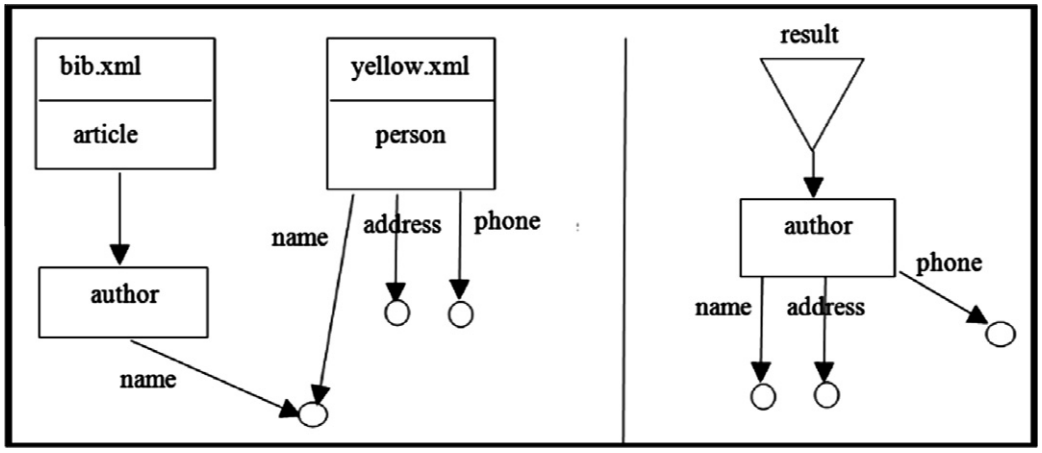

Figure 9 Join in XML-GL.

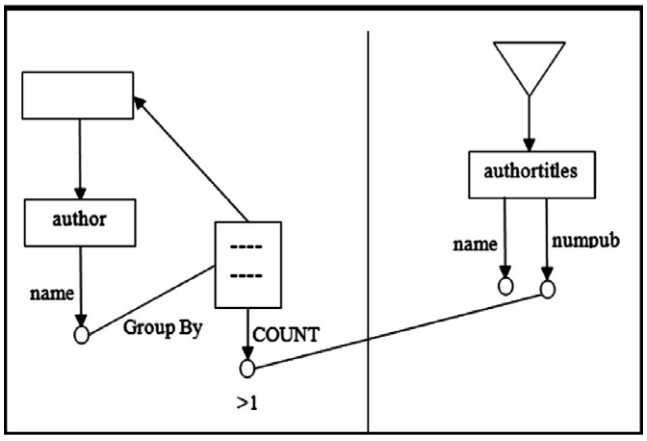

Figure 11 Aggregate function in XML-GL.

In XML-GL, the empty rectangle represents a path expression that mainly used here to express any publication regardless of its type is article or book. Note that the direct line is used to connect the two empty rectangles with each other to prevent ambiguity. Group by operator is represented in XML-GL with a box as in the above figure. This operator must have an edge that indicates a grouping criterion.

Example 6 (*Aggregate function*). The query in Fig. 11 returns authors with publication number above one.

The aggregate function (COUNT) is expressed by means of the same notation that used for specifying group by operator in the construct part (RHS). An aggregate value is represented as a property of a grouping list with the same name of the aggregate operator like COUNT here.

3.4. XML-GL limitations

Although XML-GL uses the tree representation to model XML documents, it violates one rule of tree structure in join operator by allowing two arcs to point to the same node. So join operators have more vertices and connections which always perform messy and unclear for end users. Also, XML-GL requires users to have an absolute knowledge of the XML source data schema, which a real limitation of XML-GL.

4. Querying and restructuring XML data by GLASS language

GLASS (Graphical Query Language for Semi-Structured Data) is developed as a graphical language for users to extract information from semi-structured data (Gillian et al., 2000). It supports aggregation, negation and other XQuery standards. Based on the ORA-SS model, most notations in GLASS are reused from the ORA-SS diagram.

4.1. GLASS query

A typical GLASS query consists of four parts (Ni et al., 2003):

1. *Left hand side graph (LHS graph)*: denotes the basic conditions of a query (which can be different from the structure of source schema).
2. *Right hand side graph (RHS graph)*: defines the output structure of the query result.
3. *Link set*: specifies the bindings between the RHS graph and LHS graph. When two graph entities are linked, they are visually connected by a line, which means the data type and value of the entity in the RHS graph are from the corresponding linked entity in the LHS graph.
4. *Condition logic window (CLW)*: it is an optional part where users write conditions and constructions that are difficult to draw, which includes logic expressions, mathematic expressions, comparison expressions and IF-THEN statements.

4.2. Basic query operators

GLASS answers Example 1 in the previous section as illustrated in Fig. 12.

The LHS is used to indicate the scope of query to every *book* which has a *title* as a sub-element. The RHS is used to define the structure of output. The solid line is used to connect two titles on both sides as a constraint that the titles in the result on the RHS are just the titles that available in the LHS.

Example 2 will be answered in GLASS as depicted in Fig. 13:

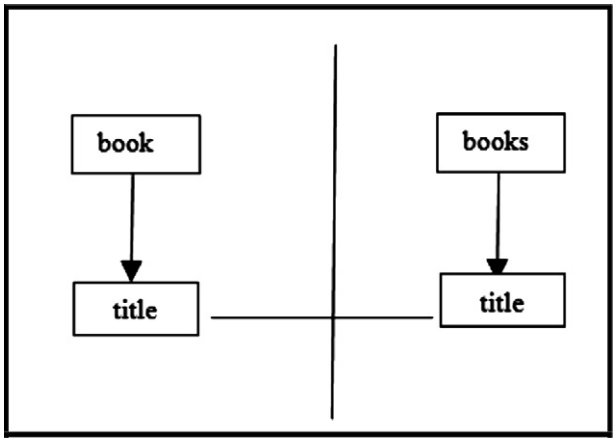

Figure 12 Projection in GLASS.

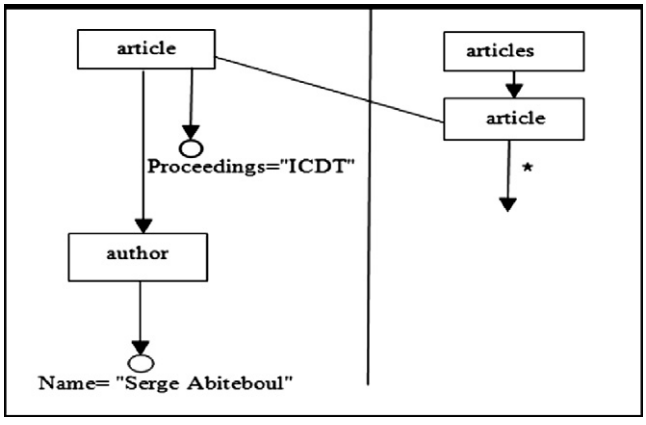

Figure 13 Selection in GLASS.

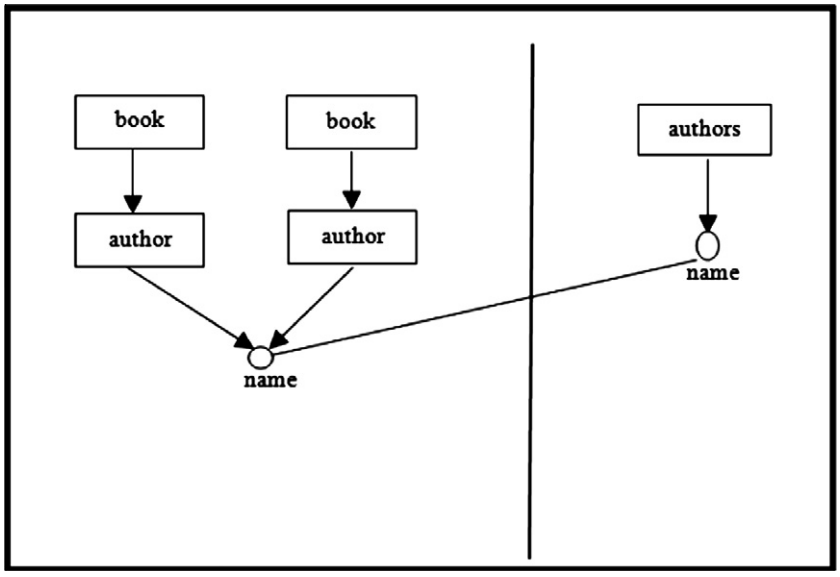

Figure 14 Complex selection in GLASS.

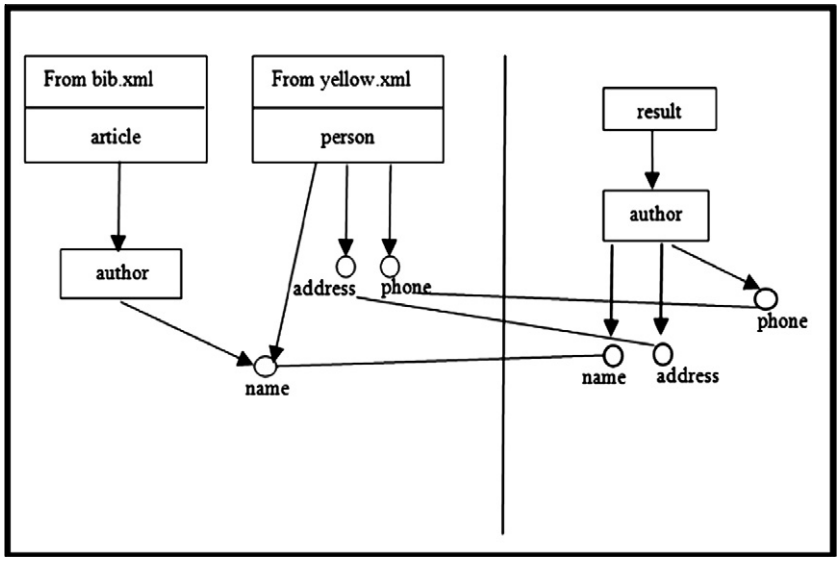

Figure 15 Join in GLASS.

GLASS uses * in the RHS to extract articles that written by "Serge Abiteboul" and proceedings in "ICDT" and all information at all levels under *article* elements.

In Fig. 14, GLASS accomplishes the complex selection example.

In this query, the LHS has two target elements from the same target document *book* and *article*. *name* element is the connector element between them. Pointing to the same element (*name*) is using to express equality between these targets in author name.

In Fig. 15, GLASS performs the join example in the previous section.

In this query, the join condition on name is expressed in the LHS by expanding the graph of the *article* element to show the inner *author* element, and connecting the author's and person's names. Direct lines used here to connect LHSs' elements with RHSs' elements. Without these lines, the RHSs' elements will be NULL.

4.3. Extended query operators

The grouping example answered by GLASS language is given in Fig. 16.

Here, you can note that the position between author and book has been changed from the original schema. Based on that, we call the GLASS query graphs as "view graphs because it is exactly a user defined view instead of tying him with the original data schema".

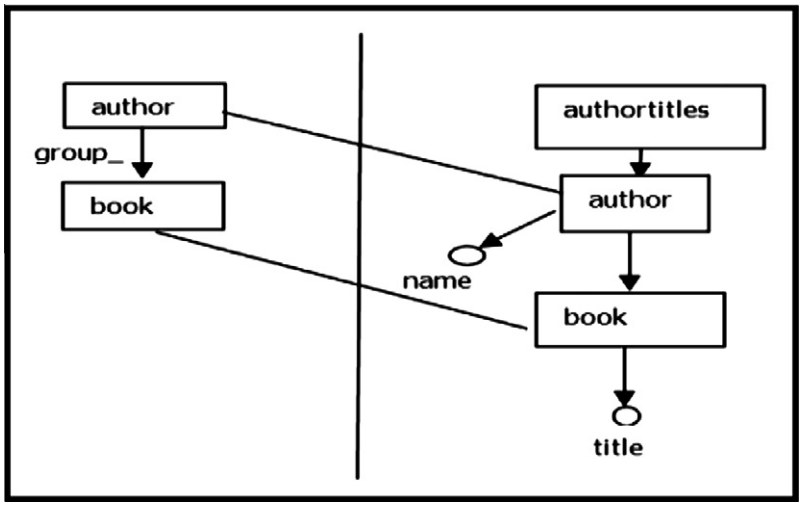

Figure 16 Grouping in GLASS.

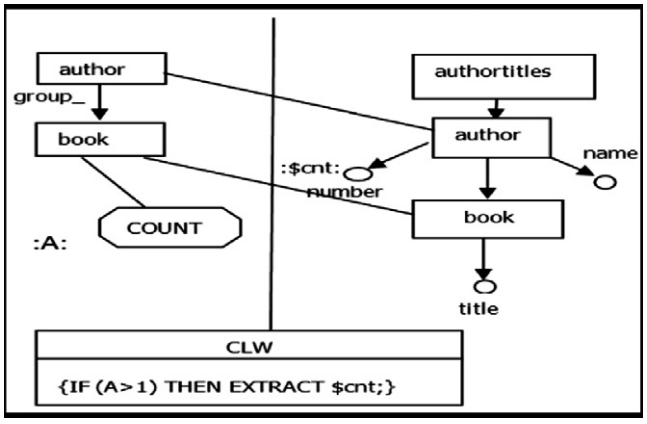

Figure 17 Aggregate function in GLASS.

GLASS answers the aggregate function example in Fig. 17.

GLASS query graph uses CLW to make sure that the output conditionally organized so that "if the number of author's publications more than one, display the result as the RHS's structure".

4.4. GLASS Limitations

GLASS does not support some important features like path expression. Also, it presents programming construct (If-Then clause). Although GLASS uses the tree representation to model XML documents, it violates one rule of tree structure in join operator by allowing two arcs to point to the same node which produce more ambiguity.

5. Querying and restructuring XML data by XQBE language

XQBE (XQuery By Example) (Braga et al., 2003) is an extension of XQuery inspired by the QBE (Query By Example) Zloof, 1977. It is initially proposed as an alternative to SQL. While QBE is a relational query language, based on the representation of tables, XQBE is based on the use of trees. XQBE was designed with the objectives of being intuitive and of being easy to map directly to XQuery, so as to be a GUI capable of running on top of any existing XQuery engine. However, the expressive power of XQBE is limited in comparison with that of XQuery (World Wide Web Consortium, 2001). XQBE does not support user defined functions, another limitation of XQBE concerns the support for disjunction.

5.1. XQBE query

An XQBE 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). 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) which newly generated XML items are to be inserted. The correspondence between the components of the two parts is expressed by explicit bindings across the vertical line and connects the nodes of the source part to the nodes that will take their place in the output document. New elements are always depicted as trapezia in XQBE. The following sections present some examples of XQBE queries.

5.2. Basic query operators

Example 1 (*Simple projection*). Based on XQBE bibliographic data model in Fig. 4 this query returns the title of all books (Fig. 18).

The binding edge between the *title* element in the LHS and *title* in the RHS causes the construction of a *title* in the result corresponding to title in the LHS. The trapezoidal *books* node above the *title* node means that all the generated titles are to be contained into a single *books* element. This node represents a newly generated element in XQBE.

Example 2 (*Simple selection*). This query returns all articles written by Serge Abiteboul and appeared in ICDT proceedings (Fig. 19).

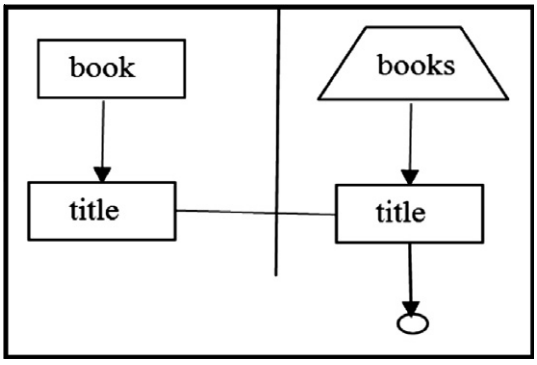

Figure 18 Projection in XQBE.

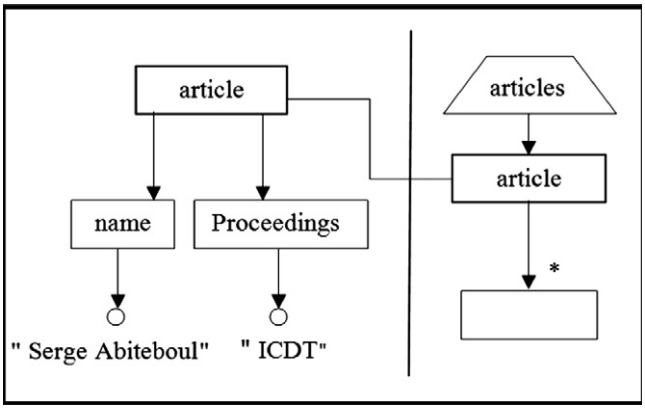

Figure 19 Selection in XQBE.

There are two logic conditions on *name* and *proceedings* which are filter articles to be result. The binding edge between the *article* nodes states that the query result shall contain as many *article* elements as those matched in the LHS part. In the RHS part, the path with * out of *article* element is mean that all information at all levels under the *article*.

Example 3 (*Complex selection*). This query returns authors who have written books and articles (Fig. 20).

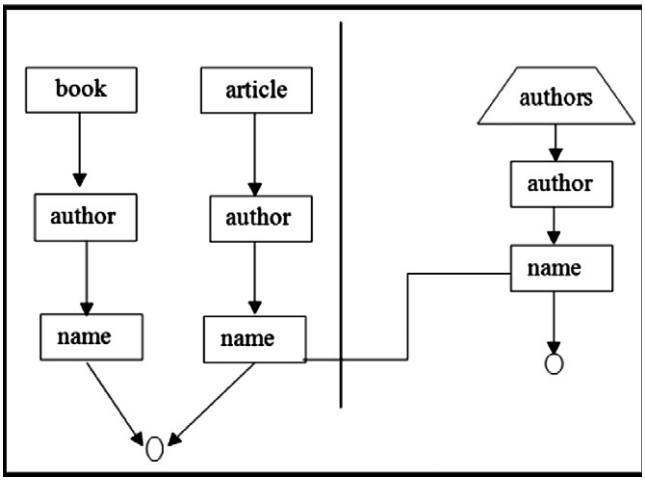

Figure 20 Complex selection in XQBE.

In this query, the LHS has two target elements from the same target document *book* and *article*. *name* element is the connector element between them. Pointing to the same element (*name*) is using to express equality between these targets in author name.

Example 4 (*Join*). Let us consider the XQBE data model of Fig. 4 and let us consider the yellow XML data file that giving addresses and phone numbers of people. Query in Fig. 21 returns the address and the phone number of each article author.

This query performs the join of authors and their addresses based on their names. In XQBE, the equality between the values is expressed by means of the confluence into a single value node, that represents the PCDATA contents of both the *name* elements.

5.3. Extended query operators

Example 5 (*Grouping*). For each author, return the list of his/her publications (Fig. 22).

Grouping in XQBE is done implicitly by using grouping node (trapezoidal). You can note that the path expression is expressed in the LHS graph by +. This symbol means any publication regardless of its type is book or article.

5.4. XQBE limitations

Although XQBE is a fairly robust and competent graphical interface for XQuery, it has a few shortcomings. First, XQBE requires users to have an absolute knowledge of the XML source data schema. XQBE does not display the entire source schema to users as a nested tree, so users need to know the names of all the elements and their hierarchical relations in the source schema. Second, XQBE defines many abstract symbols like two kinds of trapezoids, lozenges of two different colors and so on. With many symbols, it is difficult to remember which abstract symbol represents what concept.

6. Querying and restructuring XML data by GQLX language

GQLX (Ykhlef and Alqahtani, 2009) is denoted for Graphical Query Language for XML Data. The goal of designing GQLX was to query and restructure XML documents graphically. This language is able to express complex queries in an efficient manner. All of its elements are displayed visually; therefore, GQLX is suitable for supporting a user-friendly interface. It supports aggregation, aggregating and other standards.

6.1. GQLX query

GQLX is a graphical query language for G-XML data model. A GQLX query can be applied either to a single XML document or to a set of documents. Each query produces a new XML document as the result. GQLX query is a pair of G-XML graphs, displayed side by side and separated by a vertical line. With respect to SQL, the left hand side (LHS) graph visually represents from/where parts, while the right hand side

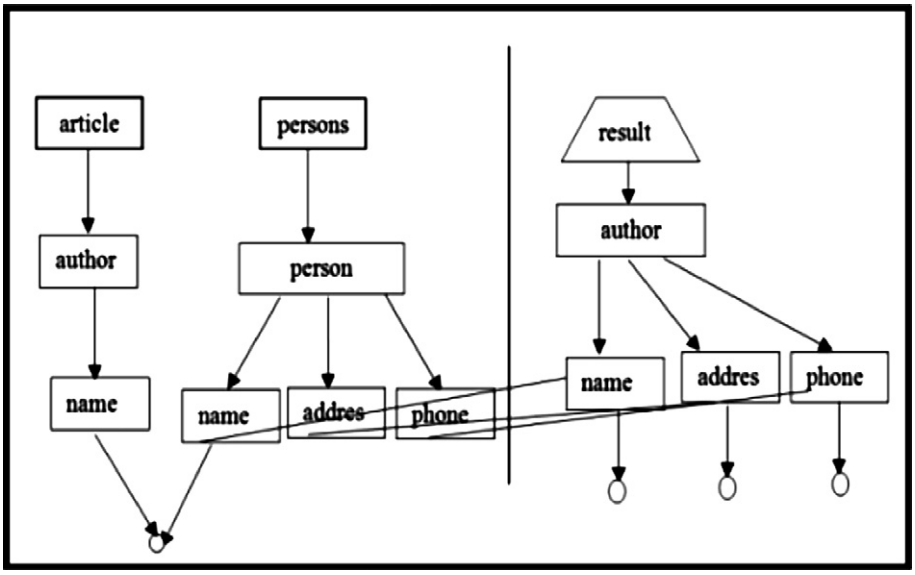

Figure 21 Join in XQBE.

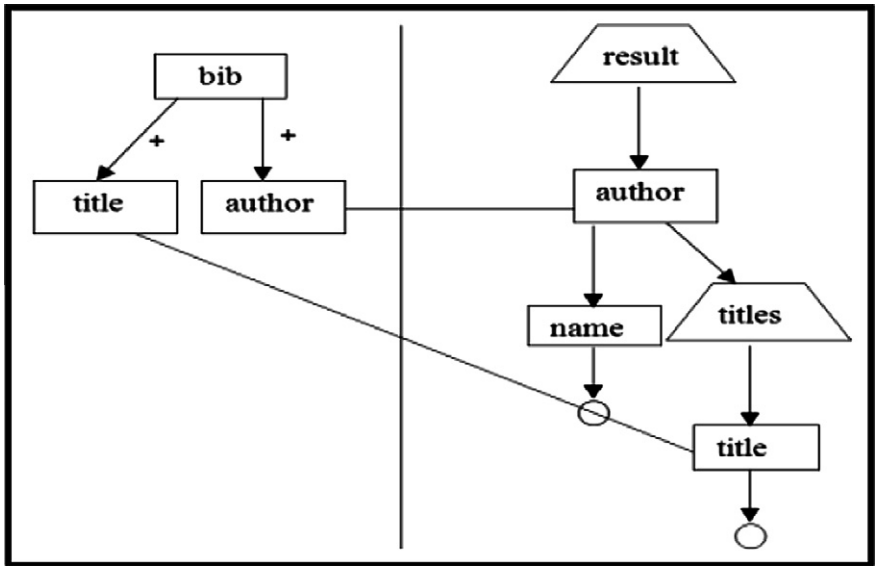

Figure 22 Grouping in XQBE.

(RHS) graph represents select/create view parts. A GQLX query's parts are described as the following:

The LHS part is mainly used to specify the scope of the query, by indicating both the target documents and the target elements inside these documents.

1. The LHS part can be used optionally to specify logical conditions.
2. The RHS part specifies the sub-elements of the extracted elements in the LHS part to be retained in the result.
3. The RHS part can be used optionally to create or construct new elements that should be included in the result document.
4. The condition box is an optional part in a GQLX query. It is placed to write logical conditions for complex queries rather than draw them in the graph.

6.2. Basic query operators

Example 1 (*Simple projection*). Based on bibliographic G-XML in Fig. 5 this query returns the title of all books (Fig. 23a). The LHS contains the indication of the target document(s) that should be used as input in order to evaluate the query like "**bib.xml**". In the rest queries of this article we will ignore the document indication to save the space. The RHS part defines the structure of the result document as a G-XML graph out of the structure of the target elements mentioned in the LHS. The corresponding between LHS and RHS elements is done explicitly by data variables like @x here. The double node that follows the **books** arc in the RHS part used here to environ the collections of **titles**. In GQLX, new elements are created directly in RHS if they have not data variables.

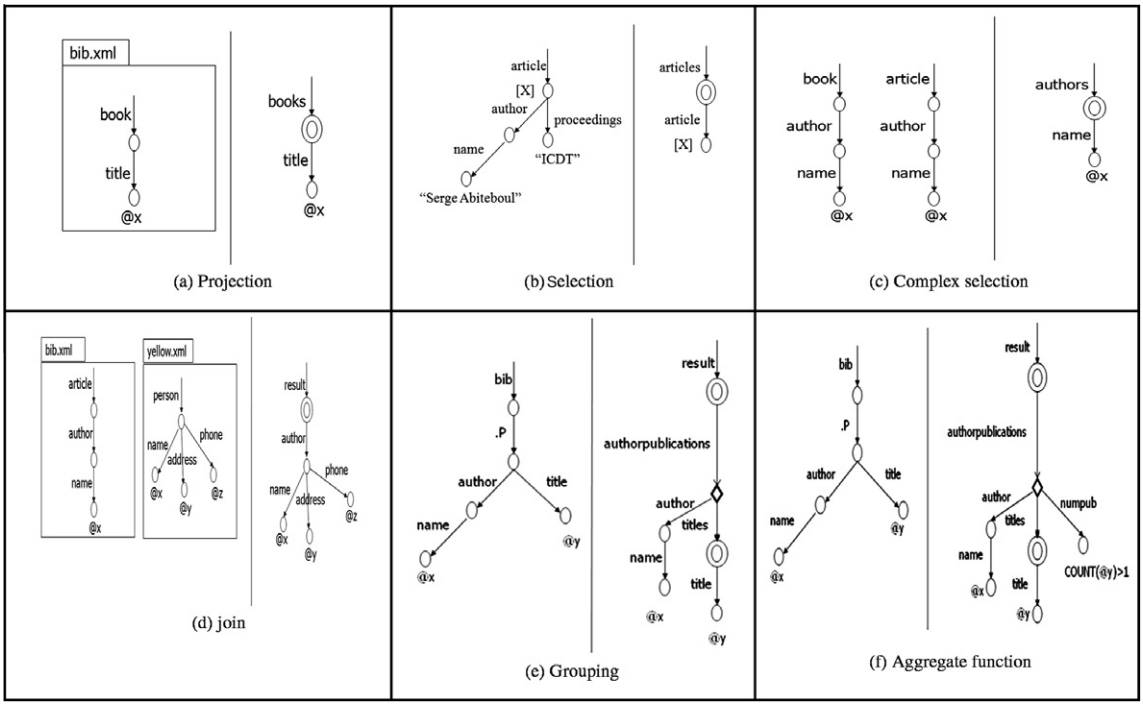

Figure 23 Queries of GQLX.

Example 2 (*Simple selection*). This query returns all articles written by Serge Abiteboul and appeared in ICDT proceedings (Fig. 23b). In this query, logical conditions are booleans and attached to specific nodes name and proceedings. The node that has a graph variable X beside it in LHS is used mainly to gather all sub-graphs starting with article node and match logical conditions. In the RHS, the graph variable X is used to include all its sub-elements in the result.

Example 3 (*Complex selection*). This query returns authors who have written books and articles (Fig. 23c). This query performs the ''inner join'' of book and article based on their author name. The equality between the values is expressed by using data variable @x.

Example 4 (*Join*). Let us consider the bibliographic G-XML of Fig. 5 and let us consider the following XML data file (yellow.xml) giving addresses and phone numbers of people:

```
< persons >
  < person >
    < name > Serge Abiteboul < /name >
    < address > INRIA-France < /address >
    < phone > + 33 1 72 92 59 18 < /phone >
  < /person >
< /persons >
```

Query in Fig. 23d returns the address and the phone number of each article author. The join condition on name is expressed in the LHS part by expanding the graph of the article and the person to the inner name elements. After that, the joining operation is done explicitly by data variable @x.

6.3. Extended query operators

Example 5 (*Grouping*). For each author, return the list of his/her publications (Fig. 23e). In GQLX, the grouping operation is directly done by using the grouping symbol (the diamond node). In LHS part, the path variable P used to find all occurrences of author and title elements available at any level of nesting, so P will be evaluated to the empty path ϵ or to a path having 1 to n arcs. We mainly use P here to specify any publication regardless of its type is a book or an article.

Example 6 (*Aggregate function*). Query in Fig. 23f returns authors with publication number above one. The last condition is achieved by (COUNT(@y) > 1). GQLX language uses the notion of ''Group by'' with ''Having'' in the same spirit of SQL language. Actually, this makes GQLX more expressive and more efficient. In general, aggregate functions are done in a condition box while a simple grouping can accomplish directly in RHS part without using of a condition box.

7. Other graphical query languages

Besides the presented four languages, there are many of graphical languages with different characteristics. Here, we provide brief descriptions about some languages. Xing (which is pronounced ''crossing'' and which is an acronym for XML in graphics) is based on a simple visualization of XML data, called the document metaphor. This document represents elements by boxes with the tag printed in bold face as a header above the box and the contents visualized inside the box. The same notation is used for attributes except that attribute names are not set in bold face (Erwig, 2003). A Xing query consists of two document patterns that are joined by a double

Table 1 Comparison between graphical query languages.

Features	XML-GL	GLASS	XQBE	GQLX
Data model	XML-GDM	ORA-SS	Object tree	G-XML
Path expression	Partially	N	Y	Y
Selection, projection and join	Y	Y	Y	Y
Query order	Y	Y	Y	Y
Grouping	Y	Y	Y	Y
Aggregate functions	Y	Y	Y	Y
Restructuring new element	Y	Y	Y	Y
Querying tags	N	N	N	Y

arrow: $p \Rightarrow q$. p is called the argument pattern and specifies structural and content constraints. Argument patterns are responsible for the selection of the desired data. q is called the result pattern and specifies how matched elements are to be presented. This means, separately from restructuring, result patterns mainly perform projections on sub-elements (Erwig, 2003). XQForms (XML Query Forms) is a generator of Web-based query forms and reports for XML data. It takes as input the XML Schema, a declarative specification of the logic of the query and a set of template libraries. The usage of these three different inputs allows a clear separation between data to be queried, query logic and presentation of the results (Papakonstantinou et al., 2001). QURSED (Querying and reporting semi-structured data) is a graphical language which allows the development of web-based query forms and reports (QFRs) for XML data. It produces XQuery-compliant queries. The QURSED Editor inputs the XML Schema that describes the structure of XML data and an HTML query form (Papakonstantinou et al., 2002). The editor displays the XML Schema and the HTML pages to the developer, who uses them to visually build the query set specification and the query/visual association that indicates how each parameter is associated to HTML form. Then a compiler generates Java Server Pages, which control the interaction with the end user. The last language is GXQL (Graphical XQuery Language using Nested Windows) which is a graphical query language using nested windows to visualize hierarchy. Representations in GXQL can be directly translated into corresponding ''FLWOR'' clauses which is an XQuery standard. It also supports predicates, different path patterns, and quantifiers. It is mainly developed to overcome the XQBEs' limitations like confusion of abstract symbol and quantifiers (Qin et al., 2004).

8. Conclusion

In this article, we presented some graphical languages by focusing on their features and their basic operators. Also, we examined their capabilities via some queries with increasingly complexity. By these queries' answers, we tried to explore their features and limitations. In Table 1 we compare these graphical query languages in term of data model, path expression, grouping, aggregate functions, restructuring and querying tags. XML-GL allows the path expression partially, however GLASS has no path expressions. The drawback of XQBE is it lack of restructuring. GQLX is better since it allows full path expressions and it allows the restructuring of new elements. GQLX preserves the tree structure of XML document and it has few symbols to express queries. Also, it does not include any programming constructs like IF-THEN used by GLASS.

References

Braga, D., Campi, A., Ceri, S., 2003. XQBE (XQuery By Example): a visual interface to the standard XML query language. ACM Transactions on Database Systems (TODS) 30 (2), 398–443.

Ceri, S., Comai, S., Damiani, E., Fraternali, P., Paraboschi, S., Tanca, L., 1999. XML-GL: a graphical language for querying and restructuring XML documents. Computer Networks 31 (11), 1171–1187 (17).

Erwig, M., 2003. Xing: a visual XML query language. Journal of Visual Languages and Computing 1 (14).

Gillian, D., Xiaoying, W., Ling, T.W., Mong Li Lee, 2000. ORA-SS: An Object-Relationship-Attribute Model for Semistructured Data, TR21/00, Technical Report, Department of Computer Science.

Kepser, S., 2002. A Proof of the Turing-Completeness of XSLT and XQuery. s.l.: EberhardKarls. University Tubingen, SFB441.

Papakonstantinou, Y., Vassalos, V., Petropoulos, M. 2002. Qursed: querying and reporting semistructured data. In: The ACM SIGMOD Int. Conf. on Management of Data.

Papakonstantinou, Y., Vassalos, V., Petropoulos, M. 2001. XML query forms (xqforms): Declarative specification of xml query interface. In: IW3C2: International World Wide Web Conference Committee.

Qin, Z., Yao, B.B., Liu, McCool, Y., 2004. A graphical XQuery language using nested windows. Lecture Notes in Computer Science, 681–687.

Ni, Wei, Ling, Tok Wang. 2003. GLASS: a graphical query language for semi-structured data. In: The Eighth International Conference on Database Systems for Advanced Applications, p. 363.

World Wide Web Consortium. 2001. XQuery 1.0: An XML Query Language W3C Working Draft. <http://www.w3.org/XML/Query>.

World Wide Web Consortium. 2001. Extensible Stylesheet Language (XSL) Version 1.0, W3C Recommendation. <http://www.w3.org/TR/xsl/>.

Ykhlef, M. 2007. Recursive SQL-Like query Language for XML. In: Jakarta: The 9th International Conference on Information Integration and Web-based Applications and Services, pp. 235–245.

Ykhlef, M., Alqahtani, S. 2009. GQLX: a new graphical query language for XML data. in: 11th International Conference on Information Integration and Web-based Applications and Services (IIWAS 2009), Malaysia.

Zloof, M.M., 1977. Query by example: a database language. IBM System journal, 324–343.