

XQBE: the Swiss Army Knife for Semi-structured Data

Daniele Braga¹, Alessandro Campi¹, Davide Martinenghi², Alessandro Raffio¹,
and Damiano Salvi¹

¹ Politecnico di Milano Dipartimento di Elettronica e Informazione piazza Leonardo da Vinci 32 I-20133 Milano Italy {braga,campi,raffio,salvi}@elet.polimi.it	² Roskilde University Computer Science Department P.O. Box 260 DK-4000 Roskilde Denmark dm@ruc.dk
---	---

Abstract. The growing importance of XML calls for easier access to data management technologies, in order to provide domain experts who are inexperienced in database technologies with the possibility to directly query and transform domain specific data. Intuitiveness and simplicity are gained with the use of a graphical representation. The former is obtained by depicting the hierarchical XML data model as tree structures; the latter consists in considering only elements, attributes, and un-typed textual data. In this regard, the graphical framework of XQBE is a suitable tool for querying, transforming, and updating XML data as well as for specifying integrity constraints.

1 Introduction

The diffusion of XML in most applicative fields poses a pressing need for providing a wide spectrum of professionals with the ability to handle XML data, including users with minimal programming skills. This paper describes a user friendly interface, based on an intuitive visual paradigm, that we developed for this purpose.

The success of the QBE paradigm (Query By Example [16]) demonstrated that, in order for a visual query language to be effective and intuitive, the basic graphical constructs of the language need to be close to a well known and well understood visual abstraction of the underlying data model. Accordingly, XQBE is based on annotated trees, so as to adhere to the hierarchical nature of the XML data model.

The W3C (World Wide Web Consortium) promotes two textual languages to express XML document transformations and to query XML data, XSLT [13] and XQuery [15] respectively. These languages, however, are far too complicated for occasional or unskilled users who might need to specify document mappings or transformations. Nevertheless, awareness of the basics of the XML data model and familiarity with the schema of the documents to be managed should be enough to allow such users to express their queries and transformations with

the core primitives of a simple manipulation language. Without this, XML will never step up to the status of a universally and successfully adopted data representation format.

XQBE (XQuery By Example [1]) was designed with the main objective of being easy to use; we also tried to achieve the highest possible expressiveness. Of course such goals cannot be fully achieved at the same time; *usability* is the most critical success factor, and therefore has been taken into consideration during the whole language design and GUI implementation process.

1.1 Related work

XQBE comes after a long stream of research on graph-based logical languages. The ideas in this field started years ago with the QBE paradigm [16].

Usually in graphical query languages, such as Graphlog [5], Good [11] or G-Log [12], graphs are patterns to be “matched” against a graph-based representation of the target data. G-Log 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 [4] to query internet pages and semi-structured data, by adding to G-Log some hypermedia features.

XML-GL [3] 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.

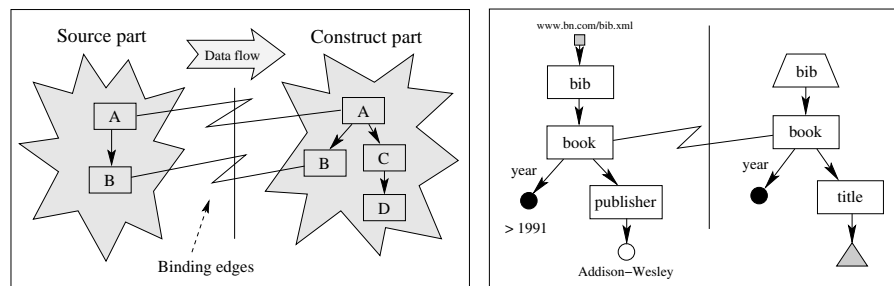
QSByE (Query Semi-structured data By Example [6]) is a graphical interface that represents data as nested tables and extends the QBE paradigm to deal with semi-structured data.

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: the only restructuring primitive is the introduction of new nodes for aggregates.

QURSED [10] allows the development of Web-based query forms and reports (QFRs) for XML data. QURSED is based on the QSS formalism, a capability-description language [7], and produces XQuery-compliant queries.

2 XQBE as a query and transformation language

The basic visual paradigm of XQBE is summarized in Figure 1(a). A vertical line divides the *source* part of the query (on the left) from the *construct* part (on the right). Thus, the query has a natural reading order from left to right. The source part describes the XML data to be matched against the set of input documents, while the construct part specifies which parts will be retained in the result, together with (optional) newly generated XML items. Both parts can be



(a) The visual paradigm **Fig. 1.** (b) An XQBE query

annotated to express selection predicates, and the correspondence between the components of the two parts is expressed by explicit binding edges across the vertical line, which connect the nodes of the source part to the nodes that will take their place in the output document.

Example 1.

```

<!ELEMENT bib      (book*)>
<!ELEMENT book    (title, (author+|editor+),
                  publisher, price)>
<ATTLIST book year CDATA #REQUIRED>
<!ELEMENT author (last, first)>
<!ELEMENT editor (last, first, affiliation)>
<!ELEMENT title   (#PCDATA)>
<!ELEMENT last   (#PCDATA)>
<!ELEMENT first  (#PCDATA)>
<!ELEMENT affiliation (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT price  (#PCDATA)>

```

2.1 A GUI for XQuery

The first query, q1 (named Q1 in [14], section 1.1.9) reads “*List books published by Addison-Wesley after 1991, including their year and title*”:

```

<bib>
{ for $b in doc("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>

```

In XQBE, all the XML *elements* in the target document are depicted as labelled 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*. Value nodes may be labelled, so as to express conditions on the values they represent.

The XQBE version of q1 is in Figure 1(b); the source part matches all the *book* elements with a *year* attribute whose value is greater than 1991 and a *publisher*

subelement whose PCDATA content equals “Addison-Wesley”. As shown by the grey square node above the `bib` node in the source part, URLs may be used as labels to locate the target XML documents. In the construct part, the paths that branch out of a bound node indicate which of its sub-items are to be retained, thus “projecting” the bound node. In `q1` only the title and publication year of the selected `books` are retained. The grey triangular node below the `title` node expresses the inclusion into the result of the entire `title` fragments, obtained by means of projection from each `book` element. This notation is always used to synthetically include entire fragments in the result. The binding edge between the `book` nodes states that the query result shall contain *as many* `book` elements *as* those matched in the source part. The trapezoidal `bib` node above the `book` node means that all the generated `books` are to be contained into one `bib` element. This node represents a newly generated element, and *new elements* are always represented as trapezia in XQBE.

XQBE allows one to specify more complex queries, with joins, negation, Cartesian products, aggregates and arithmetic computations. The XQuery generated by the automatic translator is compliant with a grammar describing a *normal* subset of XQuery. A complete reference is in [1], where a translation from this subset to XQBE is shown.

2.2 A GUI for XSL Transformations

XQBE, originally designed as a visual syntax for simple XQuery statements, lends itself well also to the specification of XSLT stylesheets, so as to perform XML-to-XML transformations; this scenario also covers XML-to-HTML transformations, as HTML can be regarded as one particular XML language.

As an example, consider a transformation that reads “*Build an HTML table with one book in each row, having one column for the title and one for the price*”. In XSLT:

```
<xsl:template match="/">
  <table>
    <xsl:for-each select="bib/book">
      <tr> <td> <xsl:value-of select="title"/> </td>
        <td> <xsl:value-of select="price"/> </td> </tr>
    </xsl:for-each>
  </table>
</xsl:template>
```

One row is constructed for each book, according to the interpretation of the binding edge that connects the `book` element in the source part with the `tr` element in the construct part. In other words, a “renaming” of the book elements is performed.

In each row two `td` elements are inserted, each one in turn containing a value projected from the matched books. This projection requires further explanation. The previous query (`q1`) projects the `book` elements “in breadth”, but dealing with trees also requires the ability to project them “in depth”, i.e., to take

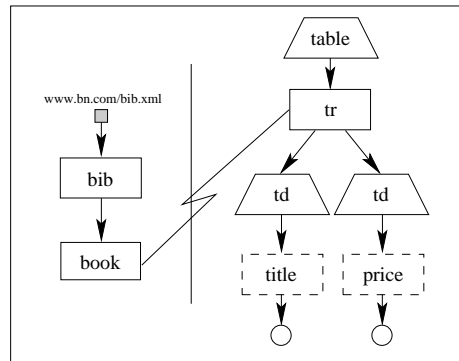


Fig. 2. Tables

far descendants of a given element and place them as direct subelements of that element, pruning the elements in the middle. Accordingly, dashed elements (such as `title` and `price` in the construct part) represent elements to be traversed but not included in the output document.

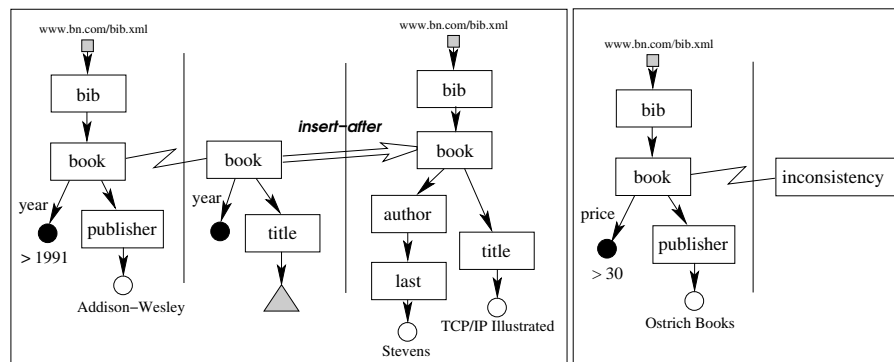
Also note that the presence of the `td` trapezoidal nodes does not prevent the projection of the `book` elements: trapezoidal nodes add new tags without “cutting” the context in the projection of already bound elements.

3 XQBE as an update language

XQBE was also adapted to support the specification of updates. Such extension required to add a third query area to the XQBE visual paradigm; such area (the *target* part of the update) is placed at the right of the construct part and locates the effect of the update. The source and construct parts were preserved because the update language is nothing but an extension of the query language. The use of the three parts is exemplified by the insertion depicted in Figure 3(a), which inserts the result of the query specified by the first two parts (namely the result of `q1`) after the `book` titled “TCP/IP Illustrated”, by W. Stevens.

4 XQBE as a constraint language

Semantic information in databases is typically represented as integrity constraints, i.e., properties that must always be satisfied for the data to be considered *consistent*. Besides simple forms of constraints such as primary and foreign keys, real-world applications may involve nontrivial integrity requirements that capture complex data dependencies and “business logic”. Maintaining compliance of data with respect to integrity constraints is a crucial database issue, since, if some data lack integrity, then query answers cannot be trusted. Trusted semantic properties can then be exploited to improve performance of information retrieval by means of so-called semantic query optimization. In this respect,



(a) An XQBE update

(b) An XQBE denial

Fig. 3.

XML is no exception; moreover, there is no standard means of specifying generic constraints over XML document collections. Although XML Schema offers a rich set of predefined constraints, it lacks full extensibility, since it is not possible to express general integrity requirements in the same way as, e.g., SQL assertions.

In a typical scenario, complex rules are best specified by domain experts, who, however, may lack knowledge in data definition and manipulation languages. A number of applications can be envisaged, e.g., in the medical domain, where health care professionals sharing information stored in clinical records may need to impose constraints, say, on compatibility between medicines and patients' profiles. A graphical framework such as XQBE would provide the domain specialist with a suitable and intuitive tool for the specification of integrity constraints.

Example 2. Suppose that in the collection of example 1 it is required that “Ostrich Books” (that only publishes paperback editions) only sell books cheaper than 30\$. Such requirement can be specified by means of an XQBE query (figure 3(b)) intended as a *denial*, i.e., a query whose answer is supposed to be empty. In particular, we use a binding edge between the topmost element in the hierarchy in the source part of the query and an `<inconsistency/>` element in the construct part. The XML document is consistent if and only if the query answer is empty.

Once structural and semantic constraints are specified on XML documents, these need to be verified and enforced on the data repositories each time new data are added or old data deleted or modified. In this regard, a database management system needs to provide means to automatically verify, in an *optimized* way, that updates do not introduce any violation of integrity. Data collections, however, are usually very large and quickly evolve over time. Indeed, verifying the whole database each time data are updated is unfeasible, especially when the underlying data management technology is as young and unripe as in XML.

In response to this, a large body of research, starting from [9], gave rise to a number of methods for incremental integrity checking within the framework of

deductive databases and with respect to the relational data model. A constraint optimization approach is currently being studied based on the conversion of the integrity constraints and database schema to their logical/relational counterpart. In this way, optimization methods that apply to the relational model, such as [8], could be applied to these cases. In particular, using the hypothesis that the database was consistent prior to the update, specialized versions of the original integrity constraints are produced for specific update patterns, which are also expressed with XQBE. For instance, the constraint of example 2 can be specialized as follows, with respect to an insertion of books such as that of Section 3: if the publisher is “Ostrich Books” and the price of the inserted book is greater than 30\$, then the update is illegal. The difference with the original constraint is remarkable, since the simplified version contains specific values coming from the concrete update statement which allow one to filter the values on which subsequent computations are applied (in this case the XML document does not even have to be accessed). Besides instantiation of constraints with specific constants, other typical improvements come from the elimination of “join” conditions in the optimized query. Guarantee of correctness and improved flexibility are evident benefits of automated approaches with respect to current techniques based on *ad-hoc* solutions, such as hand-coding of specific consistency tests at the application level.

Finally, we point out that XQBE may also allow the specification of strategies for integrity *maintenance* along the same lines as integrity checking. The construct part of a query can be used to indicate what corrective actions to apply to *repair* inconsistencies upon the occurrence of illegal updates.

5 Conclusion

In this paper we presented a general purpose framework for semi-structured data. We showed that XQBE is a suitable tool for expressing queries in an intuitive way that can be executed by any XQuery engine. We then demonstrated how XQBE can also be used to describe transformations from XML to possibly different target formats; as a particularly interesting and important case, XQBE can transform from XML to HTML. A further application of XQBE is obtained by extending it with the ability to specify update statements adding a new area devoted to the identification of the update target position. Finally, we showed how queries specified with XQBE can be intended as integrity constraints; this proves very useful for constraint specification and optimal integrity management for XML, which are innovative topics, yet building up after a large body of previous research on deductive databases.

As mentioned, there are several contexts in which the application of visual tools offers an important contribution. In particular we mention digital libraries and the medical domain. For the former, search and transformations are the most important aspects. The latter is a good candidate for the application of constraint optimization methods, not only because data integrity in this context

is a strong requirement, but also because the presence of large quantities of data makes optimization techniques indispensable.

We envision several new directions of research. The XQBE paradigm can be used to describe advanced interaction with XML data storage systems: visual mining tasks, active rules and fuzzy queries. The tool could also be extended to declare user-defined preference criteria that would render the specification of repair actions semi-automatic.

References

1. D. Braga, A. Campi, and S. Ceri. Xqbe (xquery by example): a visual interface to the standard xml query language. *To appear on ACM Transactions on Database Systems (TODS)*, June 2005.
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. Comai, E. Damiani, and P. Fraternali. Computing graphical queries over xml data. *ACM TOIS*, 19(4):371–430, 2001.
4. S. Comai, E. Damiani, R. Posenato, and L. Tanca. A schema based approach to modeling and querying www data. In *FQAS'98*, pages 110–125, May 1998.
5. M. P. Consens and A. O. Mendelzon. The g+/graphlog visual query system. In *Proc. of the 1990 ACM SIGMOD, Atlantic City, NJ, May 23-25*, page 388, 1990.
6. I. M. R. E. Filha, A. H. F. Laender, and A. S. da Silva. Querying semistructured data by example: The qsbye interface. In *Workshop on Information Integration on the Web*, pages 156–163, 2001.
7. A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. of the 22nd Int. Conf. on Very Large Databases (VLDB)*, 2002.
8. D. Martinenghi. Simplification of integrity constraints with aggregates and arithmetic built-ins. In *FQAS'04*, volume 3055 of *LNAI*, pages 348–361. Springer, 2004.
9. J.-M. Nicolas. Logic for improving integrity checking in relational data bases. *Acta Informatica*, 18:227–253, 1982.
10. Y. Papakonstantinou, M. Petropoulos, and V. Vassalos. Qursed: Querying and reporting semistructured data. In *Proc. of the ACM SIGMOD*, 2002.
11. J. Paredaens, J. V. den Bussche, M. Andries, M. Gemis, M. Gyssens, I. Thyssens, D. V. Gucht, V. Sarathy, and L. V. Saxton. An overview of good. *SIGMOD Record*, 21(1):25–31, 1992.
12. J. Paredaens, P. Peelman, and L. Tanca. G-log a declarative graph-based language. *IEEE Trans. on Knowledge and Data Eng.*, 1995.
13. W3C. Extensible Stylesheet Language (XSL). <http://www.w3c.org/TR/xsl/>, Oct. 2001.
14. W3C. XML Query Use Cases. <http://www.w3.org/TR/xmlquery-use-cases>, Nov 2003.
15. W3C. XQuery: An XML Query Language. <http://www.w3.org/XML/Query>, Nov 2003.
16. M. M. Zloof. Query-by-example: A data base language. *IBM Systems Journal*, 16(4):324–343, 1977.