A LIST OF FEATURES THAT A GRAPHICAL XML QUERY LANGUAGE SHOULD SUPPORT

K. Bekiropoulos¹, E. Keramopoulos ², O.Beza³, P.Mouratidis⁴

¹ Dpt of Information Technology, Alexander Technology Educational Institute (ATEI), Thessaloniki, Greece, E-mail: konmpek@it.teithe.gr
² Dpt Of Information Technology, Alexander Technology Educational Institute (ATEI), Thessaloniki, Greece, E-mail: euclid@it.teithe.gr
³ Dpt. Of Information Technology, Alexander Technology Educational Institute (ATEI), Thessaloniki, Greece, E-mail: olgabeza@yahoo.gr
⁴ Dpt. Of Information Technology, Alexander Technology Educational Institute (ATEI), Thessaloniki, Greece, E-mail: petrmour@it.teithe.gr

Abstract

EXtensible Markup Language (XML) tends to become a standard way of data interchanging on the Web. Therefore a need for one or more languages was created, that will have the ability to manage and extract data from XML sources. In addition, the data on the Web do not always have strict structure (semistructured data). As a result, query languages should be able to retrieve information from these data too. XQuery is the W3C standard to retrieve information from XML documents. Moreover, graphical user interfaces are user friendly enough for naive users to use them. Thus, a graphical query language for XML documents is a very interesting research field. In this paper we draw a list of features that a graphical query language for XML documents should support. Moreover, we examine how the most representative graphical query languages for XML data have presented those features. Finally, in order to compare the languages, we present a comparison table.

Keywords: XML Query Languages, graphical query languages, semistructured data, data interchange, XQuery

1. Introduction

XML (Extensible Markup Language) [1, 2] is a markup language developed by the World Wide Web Consortium (W3C) [3] to deliver structured content over the web and to provide a competitive way of storing data. XML query languages were mainly designed as a solution for retrieving information from XML documents. Traditional SQL applications can evolve to deal with XML data using extensions for XML to construct XML data from relational data, as well as store, query, and retrieve XML data. These extensions form the SQL/XML [4, 5] and were mostly used in DBMSs; however there was a need for purely expression languages to perform this kind of applications created that led to XML Query languages.

More than twenty different query languages for XML data were proposed. The first language introduced was a query language for semistructured data [6] called Lorel [7] that was later extended for XML data. Lorel was designed for Lore DataBase Management System. Another language originally developed for semistructured data is Tree Query Language (TQL) [8, 9] which comprehends an XML Document as a tree. XQL [10] is based on path expressions similar to XPath where XML-QL [11] uses a where-construct syntax. XML Matching and Structuring Language (XMAS) [12] deals with semistructured data and uses a construct-where syntax. CDUCE [13], which is an extension of XDuce [14], is a programming language based on matching patterns and uses a select-from-where syntax. A rather interesting language is CQL [15] for retrieve XML data, with a rather strict and complicated syntax. Tequyla-TX [16] is a
typed text retrieval query language for XML documents. DQL [17] is an extension of OQL for the manipulation of tree-like and forest-like data. Quilt [18] is an XML Language for heterogeneous data sources. In fact, is a functional language in which a query is represented as an expression. XML-SQL [19] is an XML Query Language based on SQL and path table. XJ [20] is a new language for searching XML data in a Java environment as well as LINQ [21]. A very interesting approach is presented in [22, 23, 24, 25], where a query is defined using XSemantics Nets on an XML Document Warehouse (XDW). World Wide Web Consortium created XQuery 1.0 [26], as the standard query language for XML data which is also used by a number of DBMS [4], and XSLT 2.0 [27], as the standard transformation language for XML data. Both languages embed XPath 2.0 [28], which use path expressions to retrieve data from XML Documents. XQuery 1.0 and XSLT 2.0 are superset of XPath 2.0 and all three languages share the same data model and have a common set of functions and operators. XML-GL [29, 30] is a graphical query language that depicts documents and their related DTDs with the use of graphical representations. Recently, a graphical query language called XQuery By Example (XQBE) [31] was created, based on XML-GL. In fact XQBE implements XQuery 1.0 queries with graphical representations. Xing [32] is a graphical query language as well as visXcerpt [33, 34, 35, 36], which is an extension of Xcerpt [34, 35].

On the other hand, the majority of computer users need only to learn how to complete simple work tasks, whereas the problems they have to solve are usually expressed in non-computing terms. Nowadays, the main type of user has changed from the skilled professional to the computer literate (unskilled or naive) user, and thus the user interfaces has to be simpler and friendlier. The initiation of graphical user interfaces, which utilize users cognitive skills and harness both advances in graphics technology and increased computing power, simplified and improved the way users interface with computers and made computer systems accessibly to an even larger number of users. Currently, graphical user interfaces have become an essential part of any computer system and system designers have come to accept that in order to improve users' productivity it is essential for a user interface to address users' skills [37].

The structure of this paper is as follows. Firstly, in section 2, we present the four most complete/representative graphical query languages for XML sources, i.e. XML-GL, XQBE, Xing, visXcerpt. Next, in section 3, we analyze a list of features that a query language for XML data should support. For every feature of the twenty-four we present/examine the way that the four graphical query languages (of section 2) support it. Moreover, we present a comparison table between the four languages based on the features list of our methodology. Finally, we conclude the paper by drawing the results of our analysis.

2. Presentation of four Graphical Query Language for XML documents

In this section, we present the basic characteristics of the four most representative graphical query languages for XML sources. In order to present the graphical query languages more effectively, we represent graphically the same query for the four languages. Thus, the query that we use is the following: “Find all books published in 1991 and having as publisher ‘Addison-Wesley’, containing ‘year’ and ‘title’.” More analytically we will present and compare the languages in the section 3 according to the twenty four features of our analysis methodology.

2.1 XML-GL

The XML-GL [29, 30] designed by the University of Milan. It represents graphically an XML document and a DTD by using XML nodes. A query in XML-GL has two parts, the right-hand side and the left-hand side. In the left-hand side of the graph the query domain is defined, i.e. the XML documents and the query selection. In the right-hand side of the graph is defined the query projection. The data model of XML-GL is called
XML-GDM and was introduced to represent XML documents and the respective DTD. The data model XML-GDM includes three concepts: (a) objects which are presented by a rectangle, (b) properties which are presented as circles connected to the corresponding object and (c) relationships which are presented as directed arcs between objects. In figure 1 is represented the query example in XML-GL.

![Figure 1: The query example in XML-GL](image)

2.2 XQBE

XQBE - XQuery By Example [31] is considered an extension of XML-GL. It is designed on top of XQuery and it uses trees as its main graphic elements. A XQBE query consists of two parts, (a) the source and (b) the construct part. In the first part, one or more trees are included. A tree can be connected to another tree representing join between different XML documents. In the second part, only one tree is used to define the projection. Moreover, in XQBE are used four different shapes to represent XQuery components, namely (a) rectangles represents node elements, (b) empty circles represents PCDATA nodes, (c) filled circles represents attribute nodes and (d) directed arcs represents the relationship between nodes. In figure 2 the query example is presented in XQBE.

![Figure 2: The query example in XQBE](image)
2.3 Xing

Xing (Xml In Graphics) [32] is a graphical query language for XML documents. It is based on a document metaphor and it uses a form-based interface for setting queries by using pattern rules. Patterns are defined as graphical sketches. The query designed on a view of the XML Document where a number of fields used. Fields includes a header, i.e. a short textual description, and a value, which is given by text or another structured part, for example, a collection of further fields. Elements are represented by boxes with the tag printed in bold face as a header above the box and the contents visualized inside the box. By that way a document is a whole level of nested boxes. Xing is not defined on a Data Model. In figure 3 the query example is presented in Xing.

![Figure 3: The query example in Xing](image)

2.4 visXcerpt

VisXcerpt [33, 34, 35, 36] is the graphical representation of Xcerpt. Xcerpt is a rule based deductive language for semi-structured data. visXcerpt has a rule-based philosophy, i.e. visXcerpt provides rules that are similar to SQL views. A rule contains a head, which is called construct pattern and a body that consisting of logically connected query patterns. It also provides Database patterns that are representations of XML documents.

In visXcerpt nested rectangular boxes with attached "tabs" are used in order to introduce database terms. The child-elements visualizations are nested inside of the box. XML Attributes are placed in a two-column table with the name in the left column and the value in the right one. Element namespace is placed on the top right corner. Every rule is surrounded by a thin black border. At the top there is a title bar containing the keyword rule and an optional comment about the rule. A folded rule is represented by its title bar.

In visXcerpt, a query consists of a set of rules. The rules are arranged as a list from top to bottom. A rule includes construction and query patterns. The construction pattern and the query patterns are arranged horizontally and separated by an arrow pointing from the query patterns (placed to the right) to the construction patterns (placed on the left). A query is displayed approximately the same way as a rule: a black bordered box with a title bar and the keyword query. In figure 4 the query example is presented in visXcerpt.
3. List of Features

In this section we draw a list of desirable features that an XML Graphical Query Language should support. We based our analysis methodology on [38, 39] and we also added some features from our survey. The survey is based on two axes: (a) all the characteristics that an XML query language has to support and (b) the human factors that a graphical query language has to adopt in order to be more effective with non-expert users.

3.1 Data Model

A data model for an XML Query Language [40], defines, on one hand, the input to the query processor and on the other the valid values of expressions in the language. In other words, input and output is an instance of the data model.

Only for the XML-GL’s has been defined a data model which is called XML-Graphical Data Model (XML-GDM) [41] and represents XML Documents and their related DTDs. This model consists of three concepts, objects (depicted by rectangles), properties (depicted as circles connected to the object they refer to) and relationships (depicted as arcs between objects, indicate semantic associations).

The other three languages have no specific data model. XQBE is based on the use of annotated trees, to adhere to the hierarchical nature of XML.

Xing visualizes an XML document with a simple design. Documents typically contain more or less portions of free text together with fields into which data can be entered. Fields consist of a header, that is, a short textual description, and a value, which is given by text or another structured part, for example, a collection of further fields. Elements are represented by boxes with the tag printed in bold face as a header above the box and the contents visualized inside the box.

In VisXcerpt database terms are represented through nested rectangular boxes with attached "tabs". In this graphical representation we have an underlying hierarchical structure. The child-elements visualizations are nested inside of the box. Sub-terms (i.e.
child nodes), are recursively visualized the same way and arranged vertically inside of the parent element's box.

3.2 Functions

In this paragraph we will examine built-in functions [42], the support of user-defined functions, aggregate functions and Skolem functions. A Skolem function generates a unique id for an element associated with it.

All languages have built-in functions for a variety of operations. In XML-GL, the built-in functions are implemented with graphical representations and XML-GL does not support \textit{avg} aggregate function. XQBE provides some of the XQuery's built-in functions and represents aggregate functions in rectangle nodes named by the name of each function. Xing supports Skolem and aggregate functions like \textit{count} but doesn't support user-defined functions. As Xcerpt is a functional query language, VisXcerpt supports a big number of built-in functions. All aggregate functions are supported and the user can create functions, with the use of rules. Skolem functions are not supported by visXcerpt.

Finally, neither XML-GL nor XQBE support user-defined nor Skolem functions.

3.3 Operators

There are three categories of operators: numeric, relational and Boolean [42].

XML-GL relational operators are \textit{"","<","="","<=","="}, and \textit{"<>"}; the Boolean operators are \textit{and} and \textit{or}. Numeric operators are \textit{+}, \textit{-}, \textit{*} and \textit{/}. XQBE has the following relational operators: \textit{"","<","=","<=","="}, and \textit{"<>"} and the Boolean operators \textit{"and"} and \textit{"or"}. Numeric operators are: \textit{+}, \textit{-}, \textit{*} and \textit{/}. As far as Xing is concerned, it supports Boolean operator \textit{"or"} with \textit{or-patterns} (is represented by the symbol \textit{"|"}). Xing doesn't have a predefined operator for Boolean \textit{"and"} but can achieve it by putting two or more patterns next to each other in a left-hand side of a rule (Cartesian product). Relational operators are \textit{"","<","=","<=","="}, and \textit{"<>"}. Finally, numeric operators are \textit{+}, \textit{-}, \textit{*} and \textit{/}.

All numeric and relational operators are supported by visXcerpt. As far as boolean operators is concerned, logical connectives \textit{"and"} and \textit{"or"} are displayed in different ways. For example \textit{"or"} is expressed through horizontal arrangement of its members while \textit{"and"} connected elements are arranged vertically. \textit{Or-connected} elements are arranged horizontally in a white box separated by the \textit{"or"} keyword.

3.4 Data types

In case of data types we examine built-in data types, that the graphical query language supports them as default, the null value and other data types that are defined in the XML Schema on which the query definition is based. All XML Query Languages offer a set of built-in data types. XML Schema data types and null value are not supported by any language. This is logical because no one of the four query languages of our analysis supports XML Schema (section 3.20).

3.5 Queries

In our analysis methodology, we will study the following types of query syntax: (a) general, (b) selection of elements, (c) data filtering, (d) nested queries, (e) construction of new elements on output, (f) binary queries, (g) recursive queries, (h) string search and (i) queries in document order.

Graphical languages XML-GL and XQBE have two parts in their queries, the left-one is the source part and the right-one is the construction part. Two parts are separated from a vertical line. Both languages do not support nested queries but support element construction. Union is succeeded placing more graphs in the left part. Intersection is indirectly succeeded with the use of graphical representations, whereas the difference is succeeded with discontinuous lines in graphs. Neither of two languages can perform
queries in document order. Only XML-GL can support recursive queries and string search in the document.

Xing represents elements (attributes) by boxes with the tag printed in bold face as a header (in attributes tags are not bold). For querying XML data uses **document patterns** and **document rules**. Xing succeeds selection of elements with document patterns. The desired data is described by patterns, that specify structural and contents constraints. A pattern consists of constants and variables; the constants specify objects that must appear in the data to qualify as a search result, that is, to match the pattern, and the variables are used to bind other data that is needed for further processing (and maybe for further constraints). A document can be instantly viewed as a document pattern and can be directly used as a query. Nested queries are supported by Xing. Data filtering and construction of new elements is achieved with **document rules**. Document rules can be used to restructure query results. A document rule consists of two document patterns that are joined by a double 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 the found elements are to be presented. This means, apart from doing restructuring, result patterns mainly perform projections on sub-elements. We can also assign an additional name to any found element, called an **alias**, which can be referred to by the result pattern. An alias is written as a tag followed by the regular expression in curly brackets. Finally, binary queries, recursive queries, string search and queries in document order are not supported in Xing.

In visXcerpt a Program consists of a set of rules (at least one rule). The rules are arranged as a list from top to bottom. Nested queries, construction of new elements on output, recursive queries, binary queries and queries in document order are expressed in visXcerpt. String search queries can not be expressed.

### 3.6 Quantifiers

There are two kinds of quantifiers: **existential**, in which at least one value satisfies a given condition and **universal** in which all values must satisfies a given condition. All XML Query Languages supports existential quantification. Universal quantifiers are not supported from XML-GL and XQBE. Xing offers two special-purpose patterns: **universal** and **existential** list patterns in support of quantifiers. VisXcerpt also supports this feature. The Xcerpt **All** construct is only used in the head. It is represented by a black bordered box with a translucent screening background. The contained pattern is prefixed by the word ALL written in italic font, because it is an Xcerpt keyword.

### 3.7 Order

Order includes the sorting of the result (either in ascending or in descending order) as well as with the preservation of order of the result. All languages can sort the result. In XML-GL one node can have an ASC or DESC label followed by the keyword ORDER. XQBE supports sorting of the result with a graph (depicted as rectangle) at the right-hand side of a query having as label the word SORT and an arrow with a label BY. In Xing sorting is performed on tags and, on equal tags, recursively by sub-elements. All languages can preserve the data order of Document.

### 3.8 Update Mechanism

XML Query languages offer an update mechanism to insert, delete, update and rename XML elements and attributes [43].

XML-GL supports insertion, deletion and update placing arrows with labels I, D and U respectively. All the other languages don’t have any specific mechanism for update.
3.9 Variables

Variables associated with elements or attributes of a Document and because of that they can be used in a query depicting the associated element. The graphical XML-GL, XQBE and Xing do not bind variables with data. VisXcerpt supports variables. Generally there are two different use cases for using variables in visXcerpt: variables placed somewhere inside a pattern and the so called as-construct, which restrict a variable to a sub-pattern. Variables are represented as black boxes with their name written in white at the top left corner inside its bounds.

3.10 Joins

A join in terms of XML is defined as a comparison between elements in the same document or different documents. All XML Query Languages support this feature. In Xing we can achieve Cartesian product and join operators by putting two or more patterns next to each other in a left-hand side of a rule. In visXcerpt conjunction and joins are supported. In order to query multiple resources at the same time, it is possible to put several queries in one rule body. Queries can be connected as disjunction or conjunction. If no conjunction is specified, a conjunction is built as the queries are arranged vertically. In a conjunction shared variables can be used to model joins.

3.11 Query Result

This feature has to do with the output of a query, which can be textual or graphical. All XML query languages return XML Documents; i.e. the query result is given in a textual format. visXcerpt is a rule-based query and transformation language for XML and because of that, it can return various types of documents in the output like HTML documents. Transformation is supported with the use of chaining. Chaining is the process of using the result of a rule evaluation as input for further rule application.

3.12 Path Expressions

Since XML Documents have a tree hierarchy, when the total structure is not known, it is very convenient to use a form of "navigational" query based on path expressions. Xing, XML-GL, XQBE and visXcerpt support path expressions with the use of regular expressions and wildcards. Furthermore, in Xing other possibilities are or-patterns and deep queries. In visXcerpt regular expressions are labeled by the Keyword regexp. The regular expression itself is underlined with a dotted line.

3.13 Conditional Expressions

Conditional expressions are if-then-else expressions. The languages do not support this feature.

3.14 Negation

Negation is the ability to query all elements that don’t have certain children elements or attributes. XML-GL and XQBE depict negation using discontinuous lines in the graphs. Xing doesn’t support this feature. The logical connective “not” is not currently covered in visXcerpt. Also the construct some of Xcerpt, that can be used to take some results out of pattern matches, is not implemented too.

3.15 Grouping Results

Another important feature of a query language is the grouping of results following specific criteria. XML-GL support group by. Xing supports group by operation with the use of flattering lists. In fact, grouping can be considered the inverse operation of flattening: a list of elements is transformed into a list of lists on the basis of common values for one (or more) sub-elements. In visXcerpt only all construct is implemented and can be used for creation of lists. XQBE do not support this feature.
3.16 Type coercion

When we want to compare elements or attributes that have different data types, it is important for a language to provide a mechanism that transforms the data types into same data types in order to be comparable. Xing, XML-GL, XQBE and visXcerpt do not support this feature.

3.17 Error handling

It is desirable for an XML Query Language to have a mechanism of detecting and handling errors during the execution of queries. XML-GL does not support this feature; it simply do not allow the same path to be accessed more than once. Xing, XQBE and visXcerpt do not have any error handling mechanism.

3.18 Namespaces

XML namespaces [44] provide a simple method for qualifying element and attribute names used in XML documents by associating them with namespaces identified by URI references. Xing, XML-GL and XQBE do not support XML Namespaces. Because visXcerpt is primarily intended to be used with XML, there is special handling namespaces. The namespace of an element is written on the top right corner in a smaller font and attribute namespaces are prefixed (also in smaller font) before the attribute name inside its table cell.

3.19 Distinct values

XML Documents can often contain a value more than once, so it is important for an XML query language to have the ability to eliminate duplicate values. The languages do not support this feature.

3.20 Support of DTD / XML Schema

It is also important for a language to support and manage an XML Schema or DTD. XML-GL and XQBE does not support this feature. Xing can exploit an XML Document from an available DTD. visXcerpt does not support directly XML Schema but it uses an XML type system similar to DTD for checking the validity of visXcerpt documents and editing operations.

3.21 IDREFs Management

An IDREF is a link by a designated key between elements in an XML document. The language or data model must be able to distinguish these IDREFs as unique values and not ordinary strings. If the query language does not support this, then the only way to link elements is if they have a descendant or containment relationship to each other. XML-GL and XQBE does not support this feature. Xing can use a graph-notation that is explicitly tied to ID and IDREF(S) attributes of elements: the fact that an attribute A references some element B is then represented as an arrow A → B. VisXcerpt uses hyperlinks in order to represent IDREFs.

3.22 Human Factors

The support of human factors, such as metaphors and color is very important in graphical user interfaces because they improve the ergonomics of the user interface [37]. In other words, in order to design a query language having as target (a) the usage of the language by naive users and/or (b) the improvement of performance of expert users we use human factors.

Only, VisXcerpt support human factors. It uses a certain number of colors (three to six) for better recognition of the element depth, i.e. every level has a different background color. Moreover, it uses a lightly beveled border line as a metaphor for nested content.
3.23 Implementation of the Language

It is obvious that a graphical query language has to be implemented. Moreover, the platform(s) used for the implementation of a language is a characteristic that can show the portability of a language. None of the four languages that we examine have been implemented. XQBE has a demo tool, and Xing a tool for testing queries. VisXcerpt supports a method for evaluating a query on a program with the use of HTTP post request.

3.24 Evaluation Method

Finally, in order to measure the quality of a language every language has to be evaluated by an evaluation method. There is no evaluation method supported by the graphical query languages.

Summarizing, we obtain the comparison table of Figure 5 that contains all the features we analyzed above. As we can notice the four languages supports a small portion of the twenty-four features that a XML graphical query language should support according to our analysis methodology. It is obvious that the four languages have been designed in order to support XML documents without having in mind HCI characteristics. Therefore, only visXcerpt uses color and metaphors but no one have been evaluated. Moreover, no one supports XML schema types, null values, conditional expressions, type coercion, error handling and distinct values. Finally, from all the above and noticing that no one of the four languages has been implemented, we believe that all the languages are in an early stage.

4. Conclusions

In this paper, we introduced and analyzed a list of features that a graphical query language for XML documents should support. Furthermore, we examined the most representative four graphical query languages for XML documents that we found in the literature. For each one of the languages we presented how they support every item of the feature list and we represented a comparative analysis table for those four query languages based on the feature list. It is obvious from our analysis that the proposed graphical query languages for XML documents are in a very early stage since they support a very small portion of the examined features that a XML query language should support. We are working now on the design of a new graphical query language for XML data that will have full functionality based on our analysis. We design the language using human factors in order to make the language user-friendly. Finally, we are implementing it in Java platform and we are planning to organize a number of experiments in order to evaluate the language and the metaphors that we will use.
<table>
<thead>
<tr>
<th>Features &amp; Languages</th>
<th>XML-GL</th>
<th>XQBE</th>
<th>Xing</th>
<th>visXcerpt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Model</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functions</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default functions.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>User defined functions</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Aggregate Functions</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Skolem Functions</td>
<td>Partially</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Operators</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arithmetic</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Relational</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Boolean</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Data Types</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XML Schema Types</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Null Values</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Queries</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Element Selection</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Nested Queries</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Constructing new Elements</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Binary Queries</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Recursive Queries</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>String Searching</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Queries in data order</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Quantifiers</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Existential</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Universal</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Order</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sorting Result</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Preserve data order</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Update Mechanism</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insertion</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deletion</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modification</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Renaming</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Variables</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Join</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Query Result</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XML</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>XML</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>XML</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>XML/HTML etc</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Path Expressions</td>
<td>Partially</td>
<td>Partially</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Conditional Expressions</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Negation</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Grouping Result</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Type Coercion</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error Handling</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Namespaces</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Distinct values</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Support of DTD / Schema</td>
<td>DTD</td>
<td></td>
<td></td>
<td>DTD</td>
</tr>
<tr>
<td>IDREFS Management</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Human Factors</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Implementation of the Language</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evaluation Method</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5: Comparative Table
AKNOWLEDGEMENTS

We would like to thank the anonymous referees for their extremely helpful comments.
This work has been supported by the Research Enhancement Program of Alexander Tecnological Education Institute.

REFERENCES

2. Extensible Markup Language (XML) 1.0 (Fourth Edition), W3C Recommendation, (29 September 2006), http://www.w3.org/TR/xml/
3. World Wide Web Consortium (W3C), http://www.w3.org/


26. XQuery 1.0: An XML Query Language, W3C Working Draft (23 January 2007), Website: http://www.w3.org/TR/XQuery


28. XPath Version 2.0, W3C Recommendation (23 January 2007), Website: http://www.w3.org/TR/xpath20/


40. XQuery 1.0 and XPath 2.0 Data Model (XDM), W3C Recommendation (23 January 2007), website: http://www.w3.org/TR/xpath-datamodel/


42. XQuery 1.0 and XPath 2.0 Functions and Operators, W3C Recommendation (23 January 2007), Website: http://www.w3.org/TR/xpath-functions/

43. XQuery Update Facility, W3C Working Draft, 11 July 2006, Website: http://www.w3.org/TR/xupdate/

44. Namespaces in XML 1.0 (Second Edition), W3C Recommendation, (16 August 2006), http://www.w3.org/TR/xml-names/