Kazienko P., Zgrzywa M.: *The evolution of Information Retrieval Languages for XML*. Information Systems Applications and Technology ISAT 2003 Seminar. Proceedings of the 24th International Scientific School. Wrocław University of Technology, 2003, pp. 92-99.

XML, information retrieval, XPath, XQuery, XML Schema, WWW

Przemysław KAZIENKO*, Michał ZGRZYWA*

THE EVOLUTION OF THE INFORMATION RETRIEVAL LANGUAGES FOR XML

XML Path Language (XPath) is nowadays the most important language for information retrieval in XML documents and its main features were described in the paper. XPath is now widespread used e.g. in XML databases, XSL Transformations, XML Schema, XLink etc. Main disadvantages and restrictions of XPath were pointed. Additionally, SQL was compared with this language.
Research on retrieval languages for XML are kept carrying out. The latest research results, incorporated into XPath 2.0 and XQuery 1.0, are: sequences, FLWOR statements and multi-document searching. The paper outlines similarities between XQuery and SQL. Also other approaches to XML information retrieval were mentioned.

1. INTRODUCTION

The very fast development of technologies related to text-based eXtended Markup Language (XML) has been observed for a few years [6]. The largest software suppliers e.g. Microsoft, Sun, IBM keep supporting these technologies, that additionally speeds up the research. One of the most important feature of the XML language is its flexibility, which facilitates creation and changing structure of documents. It also poses an important problem: how to retrieve data from such different structures in an easy and efficient way. The only solution is to create appropriate languages for querying XML documents.

2. XPATH LANGUAGE

2.1. XPATH BASIC FEATURES

The first step to create the unified standard of XML elements addressing was XML Path Language 1.0 (XPath) announced in 1999 [2]. Using XPath expressions it is possible to point at and retrieve parts of XML documents. These statements return values of four types: node-set, boolean, number and string. XPath expression can point at a node in the elements tree in an irrelative way (as a path from the root), or in a relative way – from the current element (relative location path).

XPath (and other) queries will be presented below using the following XML document, containing car spare parts data (line numbers are not the part of the document):

```
 1. <?xml version="1.0" encoding="UTF-8" ?>
 2. <Catalog Date="2003-04-01">
 3.  <CatName>Catalog of spare parts</CatName>
 4.  <Subcatalogs>
 5.   <Subcatalog Symbol="ENG24">
 6.    <SubcatName>Engines</SubcatName>
 7.    <Subcatalog Symbol="BE1">
 8.     <SubcatName>Benzine engines spare parts</SubcatName>
 9.     <Part Symbol="AD14">
10.      <PartName>Fuel separator</PartName>
11.      <Price Currency="PLN">75</Price>
12.      <Guarantee>0</Guarantee>
13.      <PartModel IdModel="Kad1.6" />
14.      <PartModel IdModel="Vec2.0" />
15.     </Part>
16.     <Part Symbol="AD15">
17.      <PartName>Fuel injector</PartName>
18.      <Price Currency="PLN">20</Price>
19.      <Guarantee>0</Guarantee>
20.      <PartModel IdModel="Vec2.0" />
21.     </Part>
22.    </Subcatalog>
23.    <Part Symbol="A111">
24.     <PartName>Benzine engine 1.6</PartName>
25.     <Price Currency="PLN">2400</Price>
26.     <Guarantee>24</Guarantee>
27.     <PartModel IdModel="Kad1.4" />
28.    </Part>
29.   </Subcatalog>
30.  </Subcatalogs>
31.  <Models>
32.   <Model Symbol="Kad1.4">
33.    <ModelName>Opel Kadet 1.4</ModelName>
34.    <Price>17000</Price>
35.   </Model>
36.   <Model Symbol="Kad1.6">
37.    <ModelName>Opel Kadet 1.6</ModelName>
38.    <Price>23000</Price>
39.   </Model>
40.   <Model Symbol="Vec2.0">
41.    <ModelName>Opel Vectra 2.0</ModelName>
42.    <Price>32000</Price>
43.   </Model>
44.  </Models>
45. </Catalog>
```

There are several axis indicating relative or irrelative position in a document tree, in XPath: root, `child`, `parent`, `self` (the current node), `attribute` and `descendant-or-self`, etc. It is possible to add to an axis further description of searching fragment, e.g. the XPath expression `/descendant-or-self::Part` will return the set of all nodes containing any part (lines from 9 to 21 and form 23 to 28), and `attribute::Date` will return string – the value of `Date` attribute of the current node (if the node is `Catalog` the result will be: `2003-04-01`).

There are couple of abbreviations for the most common location paths: `child::` is considered as the default and can be omitted, `attribute::` works just like `@`, `self` is `.` (dot), `parent` is `..` and instead of `descendant-or-self` — `//` may be used, e.g. `/Catalog`, `//Part` and `@Date`. One or more node tests, which contains criteria limiting returning node-set, can be added to the expression. All axis are allowed to be used in a node test. Additionally, logic operators as well as functions specific to different data types may be used. For example, expression: `//Part[Guarantee=0 and Price &lt; 100]` will return parts with symbols `AD14` and `AD15` (lines from 9 to 21). A set of specific functions is proposed for every data type [2, 7].

Above expressions are called "steps" and may be joined together to create so called "paths" to the node. Thus, it is easy to build complex queries, e.g.: `/Catalog/Subcatalogs//Subcatalog[@Symbol="BE1"]/Part[position()!= last() and PartModel/@IdModel="Kad1.6"]`. The return value is the set of parts designed for the model `Kad1.6` that are not the last in their subcatalogs, while the subcatalogs must have a symbol `BE1` (lines from 9 to 15). It is also possible to use multiple node tests simultaneously. Every next test filters the set returned by the previous test.

2.2. XPATH AND SQL

One of the most important application areas of retrieval languages are databases. In XML databases (also called XDBMS – XML Database Management Systems), dynamically developed for last four years, XML is the native language for storing and accessing data. Although there are many implementations of XML-based databases, still the most common architecture for commercial solutions is RDBMS (Relational DBMS) with its standard query language SQL. There are many differences between SQL and XPath (used in XDBMS). First of all, data models of relational and XML-based systems must be compared. Relational database consist of many different tables, which structure is strictly specified. The equivalent of the database in XDBMS model is a single XML document i.e. a hierarchy of elements. There are no relations between the XML data, although there are several linking techniques used instead: `ID` and `IDREF` attributes specified in DTD (Document Type Definition), `key` and `keyref` elements defined in XML Schema and links described using XML Linking Language [8]. None of them precisely correspond to relations. Besides, some of the relation information is encoded within hierarchy structure of XML elements.

Despite of the differences, both models are also similar in some aspects. This concerns their query languages: SQL and XPath, too. Let's consider the ability to embed SQL queries: `SELECT FROM (SELECT FROM ... ) WHERE (SELECT FROM ... ) = ...`. Such expressions correspond in XPath to multiple steps in the path. The result from the first step is filtered by the next one. Also the `WHERE` section is very similar to XPath node tests. Both languages uses them to filter the obtained results. However, in a SQL `WHERE` section joins conditions may also be included.

Different data models have stimulated the development of individual features of both languages. For example, SQL is working in the strictly described environment of relations. To find a row not only its key value have to be known, but also a data table (a relation) and a data column (an attribute) must be chosen. In XPath, the special `id()` function is defined, which does not need neither the name of the searching element (a relation equivalent) nor the name of the identifier attribute (a key column equivalent). Also the criteria for all of the relation's attributes cannot be defined using only one SQL expression (every data column name must be explicit defined). However, there is a * character in XPath. The query: `//*[contains(text(),'24') or contains(@*,'24')]` will return all elements (first star) that contain itself or in any of their attributes (second star) a substring "24", i.e.

the whole `Subcatalog` with the symbol `ENG24` (lines 5 - 29), an element `Price` (line 25) and an element `Guarantee` (line 26).

Lack of sort functions is another difference between the two languages. SQL has `ORDER BY` expression. While the results of XPath queries are always unordered, there are no functionality of grouping provided. In SQL, expression `GROUP BY ... USING ...` may be used. The only operation for manipulating sets of results in XPath is function `sum()` for aggregation of number values. Although it seems that XPath has poorer functionality than SQL, the differences between relational and hierarchical data models must not be forgotten. Many of the mentioned XPath drawbacks, e.g. sorting and grouping, are fulfilled in related languages, like XSLT or XQuery.

2.3. XPATH APPLICATIONS

Three main areas of XPath applications may be distinguished (fig. 1). The first one is the family of standards, which utilize the mechanism of pointing at different elements of XML documents: XSL Transformation, DOM (from Level 3), XML Schema. Languages that link XML documents and their parts – XLink and XPointer – are another area of XPath application. The XML Path Language also plays an important role in querying XML databases. Other manipulation operations on data (inserts, updates, deletes) are performed using DOM API (Document Object Model Application Programming Interface).

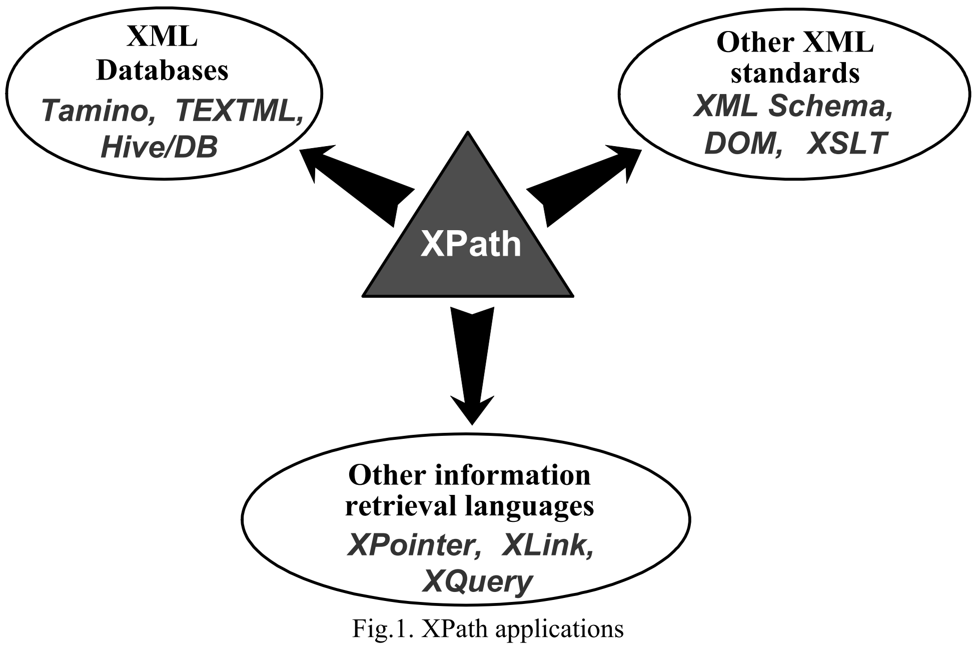

Fig.1. XPath applications

2.4. DISADVANTAGES OF XPATH

XPath is rather simple language, and its functionality is limited. The main problem of XPath is the dynamic development of XML related standards. Consequently, XPath is not compatible with the data types proposed in XML Schema in 2001. Instead of using only 4 types described above, it should recognize many others. The most important inconvenience is lack of data types for real numbers and date / time values. It is also not possible to search for texts that are spread on many elements or are parts of elements / elements names. The feature is especially important for documents with mixed elements.

Another problem is time and memory efficiency. Contemporary XPath query engines evaluate queries in time exponential in the size of input queries [5]. Works on improved versions of XPath are being carried out (e.g. algorithm OptMinContext). Also the limited variety of functions is another disadvantage. It would be useful to extend the function set with `abstractValue()`, `power()`, `to-upper()`, `to-lower()`and `replace()` (the XPath `translate()` function works only with single characters, not whole strings) [9]. In some cases, the functionality of missed operations can be implemented using combination of expressions provided. Unfortunately, such statements are usually long and hard to read.

3. XPATH 2.0 AND XQUERY 1.0

3.1. XPATH 2.0

Considering the disadvantages of XPath, works on new information retrieval languages for XML are kept carrying out. XPath 2.0 and XQuery 1.0 (XML Query Language) are currently being dynamically developed. Both are highly related to each other, they share data model, functions and operators [3]. XPath 2.0 is considered as simplified version of XQuery. It is also dedicated for broader use, among other it will be a part of XSLT 2.0 – the next version of the transformation language.

The most important innovation in XPath 2.0 is the adaptation of XML Schema data types [3, 10]. Values are easier to handle, e.g. dates may be compared and incremented, URI identifiers may be managed (parts of URLs may be obtained). Another innovation is replacing node-sets by sequences. The elements are ordered and repetitions are tolerated in sequences. The set of functions is significantly extended, so operations on data may be performed easier (e.g. `replace()` and `to-upper()` are added) [12]. The new qualifier – dereference (==>) has been also added. It makes it easier to use references between `ID` and `IDREF` elements. For example, to return the model names for the part `AD14` (lines 37 and 41), the XPath 1.0 statement is quite complex: `//Model[@Symbol = //Part[@Symbol=″AD14″]/PartModel/@IdModel]/ModelName`. If `@IdModel` is of type `IDREF`, and `Model/@Symbol` is of type `ID`, function `id()` may by used: `id(//Part[@Symbol=″AD14″]/PartModel/@IdModel)/ModelName`. The query is shorter, but using the function may be confusing. In XPath 2.0, dereference makes the statement more obvious and easier to read: `//Part[@Symbol=″AD14″]/PartModel/@IdModel==>Model/ModelName`.

3.2. XQUERY 1.0

XQuery, the next step of information retrieval languages evolution, is designed for creating new classes of queries [1, 13]. Its most powerful feature is ability to build and return new XML elements, attributes, etc. Thus, the result obtained from the query may be treated as a regular XML document. There are several ways to create an XML element. The most common is presented below – the query returns `PartPrice` elements, which are created from part names and prices:

```
for $part in input()//Part
return  <PartPrice>  {$part/Name}  {$part/Price}  </PartPrice>
```

Also the query itself may be expressed in an XML form. In this standard (called XQueryX) the question for `PartPrices` would look like:

```
<query xmlns="http://www.w3.org/2001/06/xqueryx">
  <flwr>
    <forAssignment variable="$part">
      <step axis="SLASHSLASH">
        <function name="input"/>      <identifier>Part</identifier>
      </step>
    </forAssignment>
    <return>
      <elementConstructor>
        <tagName>  <identifier>PartPrice</identifier>   </tagName>
        <step axis="CHILD">
          <variable>$part</variable>  <identifier>Name</identifier>
        </step>
        <step axis="CHILD">
          <variable>$part</variable>  <identifier>Price</identifier>
        </step>
      </elementConstructor>
    </return>
  </flwr>
</query>
```

Such expressions are much more complicated. Nevertheless, they are useful in terms of automatically generated questions. They can also be processed by any of XML tools, which makes it possible to work with both queries and answers in one environment.

The second important innovation is the introduction of FLWOR statements (`for, let, where, order, return`). These statements allow to retrieve information in more natural way than in XPath. In the `for` section, loop variables are declared, e.g. the query:

```
for $part in document("Catalog.xml")//Part
return <PartInfo> {$part/Name} </PartInfo>
```

will return three `PartInfo` elements, every containing one part name. In the `let` section, additional variables may be initiated with whole sequences. The query:

```
let $part := document("Catalog.xml")//Part
return <PartInfo> {$part/Name} </PartInfo>
```

will return one `PartInfo` element containing three part names. In `where` section, a condition concerning defined variables may be set. The query:

```
for $part in document("Catalog.xml")//Part
where $part/Price > 50
return <PartInfo> {$part/Name} </PartInfo>
```

will return only two `PartInfo` elements. The `order` keyword is responsible for sorting, e.g.: `order by Price`. In the `return` section, the result is constructed. When all five sections are prepared, the full FLWOR statement is constructed. Below example returns element `ModelParts` for `"Vec2.0"`, consisting of model symbol and names of all matching parts (lines 40, 10 and 17).

```
for $model in document("Catalog.xml")//Model
let $part := document("Catalog.xml")//Part
where $model/@Symbol = "Vec2.0"
return <ModelParts>  {$model/@Symbol} {$part/Name} </ModelParts>
order by $part/Name
```

Sequences for variables are defined by concatenation of the pointer to data source (XML document) and the XPath query. While every FLWOR statement may retrieve information from many sources, it is possible to perform joins between elements from different XML documents. Thus, the set of documents may be used as a database and queried with single expression. Below example lists all available parts using two documents: the catalog and information about parts in stock.

```
for $part in  document("Catalog.xml")//Part
let $entry :=  document("Stock.xml")//Entry
where $entry/Qantity > 0
return  <AvailablePart> {$part/Name} {$entry/Quantity} </AvailablePart>
```

There are several extensions of XQuery language, that allow to query other resources than XML: databases, html pages, active processes (C or C++ programs, Java classes) [14]. Thus, heterogeneous data sources may be treated as one consistent XML document.

Other new features of XQuery are: user-defined data types and functions, conditional statements (`if then else`) and nested queries.

3.3. XQUERY AND SQL

XQuery language is much more similar to SQL then XPath is. First of all, FLWOR statements looks like SQL queries. While SQL query builds result relation from the set of relations (tables) in database, FLWOR statement constructs new XML document from the set of provided documents. The `where` and `order by` sections in both languages are almost the same. Also joins between many documents are possible in XQuery – it is an equivalent of inner joins between database tables. Constructing the result for every element of `for` section sequence is very similar to building result by SQL cursor. Other mechanisms also have their equivalents in SQL, for example user defined functions strongly resemble stored procedures.

All mentioned similarities are not coincidental. It seems that the W3C Query Working Group, responsible for new language, wants it to look like SQL. The reason is obvious: most people working with computers know SQL very well, so it is easier for them to learn something similar. Of course XQuery has to work with hierarchical data, so it must have a few additional mechanisms – like XPath statements or declaring user data types.

4. OTHER INFORMATION RETRIEVAL LANGUAGES

Other approaches to XML information retrieval may also be considered. One of them is the conceptual querying [11], based on underlying abstract model of XML Schema. It utilizes the fact, that information is not only included in elements' data and hierarchy. It is also incorporated in the object oriented structure of data types. For example, when the task is to find all books in a XML library document, elements of all data types that derive from the `book` data type should be returned (e.g. `book`, `workbook` and `textbook` elements).

Another approach is to add completely new functionality to existing standards. For example, XIROL language [4] allows to give weights (real numbers form 0 to 1) to every single criteria included in the node test. Of course, a relevance calculating algorithm must be used during document search.

XPointer, another language developed by W3C, allows to search for strings, having all markups omitted, e.g. `xpointer(string-range(//,"200"))` will return two ranges: the first starts in line 18 and ends in line 19 and the second contains a fragment of line 42.

5. CONCLUSIONS

Dynamic development of the XML language and other derived standards affects constant enhancement of XML information retrieval mechanisms. The more complicated data is kept in XML format, the better mechanisms of querying are needed. It is especially important in an environment with evolving structure. XML databases have additional requirements, e.g. formal consistency of language or mechanisms of joins that are not necessary for a single XML document. Support for data manipulating tools is also eligible. Considering the heterogeneity of data sources that are used today, some kind of integration with other information retrieval languages (e.g. SQL) would be advisable. The development of information retrieval languages, which are the base of every data manipulating application, is also the fundament of development of all related domains.

REFERENCES

[1] BOAG S., CHAMBERLIN D., FERNANDEZ M.F., FLORESCU D., ROBIE J., SIMÉON J. (eds.), *XQuery 1.0: An XML Query Language. W3C Working Draft 15 November 2002.* WWW Consortium, 2002, http://www.w3.org/TR/xquery/

[2] CLARK J., DEROSE S. (eds.), *XML Path Language (XPath). Version 1.0. W3C Recommendation 16 November 1999,* WWW Consortium, 1999, http://www.w3.org/TR/xpath.

[3] FERNÁNDEZ M., MALHOTRA A., MARSH J., NAGY M., WALSH N. (eds.), *XQuery 1.0 and XPath 2.0 Data Model. W3C Working Draft 15 November 2002.* WWW Consortium, 2002, http://www.w3.org/TR/query-datamodel/

[4] FUHR N.*, XML Information Retrieval and Information Extraction,* University of Dortmund, 2003, http://www.is.informatik.uni-duisburg.de/bib/fulltext/ir/Fuhr:02a.pdf (to be published)

[5] GOTTLOB G., KOCH K., PICHLER R., *XPath Query Evaluation: Improving Time and Space Efficiency.* Accepted for 19th International Conference on Data Engineering, 2003 http://www.dbai.tuwien.ac.at/staff/koch/download/icde2003.pdf.

[6] KAZIENKO P., *The XML family,* Software 2.0 nr 6 (90) czerwiec 2002, pp. 21-27 (in Polish).

[7] KAZIENKO P., GWIAZDA K, *Taking XML seriously,* Helion, Gliwice, 2002, http://helion.pl/ksiazki/xmlnap.htm (in Polish)

[8] KAZIENKO P., GWIAZDA K, *XLink - the future of document linking, Information Systems Architecture and Technology,* ISAT 2001. Conference Proceedings, Wrocław University of Technology, 2001, pp. 132-139.

[9] KAZIENKO P., ZGRZYWA M., *XML Information Retrieval.* In: MiSSI 2002. Multimedialne i Sieciowe Systemy Informacyjne. Conference Proceedings (Cz. Daniłowicz, ed.). Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, 2002, pp. 391-400 (in Polish).

[10] LENZ E., *What's new in XPath 2.0,* XML.COM, 2002, http://www.xml.com/pub/a/2002/03/20/xpath2.html

[11] LUDASHER B., ALTINTAS I., GUPTA A., *Time to Leave the Trees: From Syntactic to Conceptual Querying of XML,* Proc. of Intl. Workshop on XML Data Management, Prague, Czech Republic, Lecture Notes in Computer Science 2490, Springer Verlag, 2002, pp. 148-168.

[12] MALHOTRA A., MELTON J., ROBIE J., WALSH N. (eds.), *XQuery 1.0 and XPath 2.0 Functions and Operators. W3C Working Draft 15 November 2002.* WWW Consortium, 2002, http://www.w3.org/TR/xquery-operators/

[13] MELTON J., EISENBERG A., *An Early Look at XQuery.* SIGMOD Record 31 (4) 2002, pp. 113-120.

[14] NACHOUKI G., QUAFAFOU M., *Extracting, Interconnecting, and Accessing Heterogeneous Data Sources: An XML Query Based Approach,* 2002 Université de Nantes, France