XPathMark
An XPath benchmark for XMark

Massimo Franceschet

Informatics Institute
University of Amsterdam, The Netherlands
Department of Sciences
University of Chieti and Pescara, Italy

Abstract

We propose XPathMark, an XPath benchmark for XMark. It consists of a set of queries which covers the main aspects of the language XPath 1.0. These queries have been designed for XML documents generated under the popular XMark benchmark.

1 Introduction

XMark [2] is a well-known benchmark for XML data management. It consists of a scalable document database modelling an Internet auction website and a concise and comprehensive set of XQuery queries which covers the major aspects of XML query processing.

XQuery [7] is much larger than XPath [3], and the list of queries provided in the XMark benchmark mostly focuses on XQuery features (joins, construction of complex results, grouping) and provides little insight about XPath characteristics. In particular, only child and descendant XPath axes are exploited. In this paper, we propose XPathMark [8], an XPath 1.0 benchmark for XMark. We have developed a set of XPath queries which covers the major aspects of the XPath language including different axes, node tests, predicates, Boolean operators, references, and functions. The queries are concise, easy to read and to understand. They have a natural interpretation with respect to the semantics of XMark generated XML documents. Moreover, we have thought most of the queries in such a way that the sizes of the intermediate and final results they compute, and hence the response times as well, increase as the size of the document grows. XMark comes with an XML generator that produces XML documents according to a numeric scaling factor.

The targets of XPathMark are:

- *functional completeness*, that is, the ability to support the features offered by XPath;
- *correctness*, that is, the ability to correctly implement the features offered by XPath;
- *efficiency*, that is, the ability to process the queries fast.
- *data scalability*, that is, the ability to process queries on documents of increasing sizes.

Since XPath is the core retrieval language for XSLT [4], XPointer [5] and XQuery [7], we think that the proposed benchmark can help vendors, developers, and users to evaluate these targets on XML engines implementing these technologies.

Our contribution is as follows. In Section 2, we take a close look to the XQuery benchmark proposed in XMark. We do so in order to: (i) investigate which of the queries of the XQuery benchmark are rewritable in equivalent XPath queries and which are not. This may be useful to understand the differences between XQuery and XPath. For some applications, XPath could be enough as a query language, and using the more powerful XQuery instead may not be a wise choice; (ii) show that the XPath fragment used in the XQuery benchmark is not large enough to be considered a benchmark itself. In Section 3 we carefully describe the proposed XPath benchmark for XMark. Finally, in Section 4, we suggest how to evaluate the XPath benchmark on a given XML engine. We list some future work in Section 5.

2 XMark: an XML benchmark

In this section we focus on the XQuery benchmark proposed in XMark. We proceed as follows: we first write the XQuery query in natural language, then, if it has a correspondent in XPath, we write first the XQuery version and then the corresponding XPath one, otherwise we explain why the query in not expressible in XPath.

Q1 *The name of the item with ID 'item20748' registered in North America*

```
FOR $b IN document("auction.xml")/site/regions/namerica
          /item[@id="item20748"]
RETURN $b/name/text()

/site/regions/namerica/item[@id="item20748"]/name/text()
```

Q2 *The initial increases of all open auctions*

```
FOR $b IN document("auction.xml")/site/open_auctions/open_auction
RETURN <increase> $b/bidder[1]/increase/text() </increase>

/site/open_auctions/open_auction/bidder[1]/increase/text()
```

Q3 *The first and current increases of all open auctions whose current increase is at least twice as high as the initial increase*

This query has no XPath correspondent since its result is a set of pairs of nodes, and not a set of nodes. For the same reason, queries Q8, Q9, Q11, Q12, Q13, and Q19 cannot be expressed in XPath.

Q4 *List the reserves of those open auctions where a certain person issued a bid before another person*

```
FOR $b IN document("auction.xml")/site/open_auctions/open_auction
WHERE $b/bidder/personref[@person="person18829"] BEFORE
      $b/bidder/personref[@person="person10487"]
RETURN <history> $b/reserve/text() </history>
```

```
/site/open_auctions/open_auction[bidder[personref/@person="person18829"]
/following-sibling::bidder[personref/@person="person10487"]]/reserve/text()
```

Q5 *How many sold items cost more than 40?*

```
COUNT(FOR $i IN document("auction.xml")/site/closed_auctions/closed_auction
      WHERE $i/price/text() >= 40
      RETURN $i/price)
```

```
count(/site/closed_auctions/closed_auction[price>= 40]/price)
```

Q6 *How many items are listed on all continents?*

```
FOR $b IN document("auction.xml")/site/regions
RETURN COUNT ($b//item)
```

```
count(/site/regions//item)
```

Q7 *How many pieces of prose are in our database?*

```
FOR $p IN document("auction.xml")/site
RETURN count($p//description) + count($p//annotation) + count($p//email)
```

```
count(/site//description | /site//annotation | /site//email)
```

Q10 *List all persons according to their interest; use French markup in the result*

Query Q10 cannot be expressed in XPath since XPath is not able to group and reconstruct information. For the same reason, query Q20 is not expressible in XPath.

Q14 *The names of all items whose description contains the word 'gold'*

```
FOR $i IN document("auction.xml")/site//item
WHERE CONTAINS ($i/description,"gold")
RETURN $i/name/text()
```

```
/site/regions/*/item[contains(description,'gold')]/name/text()
```

Q15 *The keywords in emphasis in annotations of closed auctions*

```
FOR $a IN document("auction.xml")/site/closed_auctions/closed_auction
          /annotation/description/parlist/listitem/parlist/listitem
          /text/keyword/emph/text()
RETURN <text> $a <text>
```

```
/site/closed_auctions/closed_auction/annotation/description
/parlist/listitem/parlist/listitem/text/keyword/emph/text()
```

Q16 *The identifiers of the sellers of those auctions that have one or more keywords in emphasis*

```
FOR $a IN document("auction.xml")/site/closed_auctions/closed_auction
WHERE NOT EMPTY ($a/annotation/description/parlist/listitem/parlist
                listitem/text/keyword/emph/text())
RETURN <person id=$a/seller/@person />
```

```
/site/closed_auctions/closed_auction[annotation/description/parlist
/listitem/parlist/listitem/text/keyword/emph/text()]/seller/@person
```

Q17 *Which persons don't have a homepage?*

```
FOR $p IN document("auction.xml")/site/people/person
WHERE EMPTY($p/homepage/text())
RETURN <person name=$p/name/text()/>
```

```
/site/people/person[not(homepage/text())]/name/text()
```

Q18 *Convert the currency of the reserve of all open auctions to another currency*

This query is not expressible in XPath since there are no user defined functions in XPath.

It turns out that 13 over 20 queries of the XQuery benchmark can be expressed in XPath.

3 XPathMark: an XPath benchmark for XMark

In this section we propose an XPath benchmark for XMark. In Section 3.1 we describe the methodology of the benchmark, while in Section 3.2 we describe

```
<!DOCTYPE benchmark [

<!ELEMENT benchmark     (document*,query*)>
<!ELEMENT document      (#PCDATA)>
<!ELEMENT query         (type,description,syntax,answer)>
<!ELEMENT type          (#PCDATA)>
<!ELEMENT description   (#PCDATA)>
<!ELEMENT syntax        (#PCDATA)>
<!ELEMENT answer        (#PCDATA)>

<!ATTLIST benchmark targets   CDATA #REQUIRED
                    language  CDATA #REQUIRED
                    authors   CDATA #REQUIRED>
<!ATTLIST document  id        ID #REQUIRED>
<!ATTLIST query     id        ID #REQUIRED
                    against   IDREF #REQUIRED>
]>
```

Figure 1: The benchmark DTD

the benchmark itself.

3.1 The methodology

The benchmark has been designed in XML (see [8] for the XML version). This solution has a number of advantages, including:

- the benchmark is easy to download, ship, read, and modify;
- the benchmark itself can be queried with XML query languages (XPath itself, for instance);
- it is handy to write a *benchmark checker*, that is an application that automatically tests the benchmark against a given XML engine and computes some performance indexes.

Figure 1 contains the Document Type Definition (DTD) for the XML document containing the benchmark. The root element is named `benchmark` and has the attributes `targets` (the targets of the benchmark, for instance, functional completeness), `language` (the language for which the benchmark has been written, for instance XPath 1.0), and `authors` (the authors of the benchmark). The benchmark element is composed of a sequence of `document` and a sequence of `query` elements. Each `document` element is identified by an attribute called `id` of type ID and it contains, enclosed into a Character Data (CDATA) section, a possible target XML document for the benchmark queries. Each `query` element is identified by an attribute called `id` of type ID and it has an attribute called

`against` of type IDREF that points to the document against which the query has been evaluated. Moreover, each `query` element contains the following child elements:

- `type`, containing the category of the query;
- `description`, containing a description of the query in English;
- `syntax`, containing the query formula in the XPath syntax; and
- `answer`, containing the result of the evaluation of the query against the pointed document, enclosed within a CDATA section. The result is a sequence of XML elements with no separator between two consecutive elements (not even a whitespace).

3.2 The XPath benchmark

We have two benchmark documents. The first document corresponds to the XMark document generated with a scaling factor of 0.0005. The XMark document type definition is included in the document. The set of queries that have been evaluated on this document are divided into the following 5 categories: axes, node tests, Boolean operators, references, and functions.

- **Axes**

  These queries focus on the navigational features of XPath, that is on the different kinds of axes that may be exploited to browse the XML document tree. XPath contains 13 axes: child, descendant, descendant-or-self, parent, ancestor, ancestor-or-self, following-sibling, preceding-sibling, following, preceding, attribute, and self.

  - **Child**

    One short query (Q1) with a possibly large answer set, and a deeper one (Q2) with a smaller result. Only the child axis is exploited in both the queries.

    Q1 *All the items*

    ```
    /site/regions/*/item
    ```

    Q2 *The keywords in annotations of closed auctions*

    ```
    /site/closed_auctions/closed_auction
    /annotation/description/parlist/listitem
    /text/keyword
    ```

  - **Descendant axes**

    The tag `keyword` may be arbitrarily nested in the document tree and hence the following queries can not be rewritten in terms of child axis.

Notice that list items may be nested in the document. During the processing of query Q4 an XPath processor should avoid to search the same subtree twice.

Q3 *All the keywords*

```
//keyword
```

Q4 *The keywords in a paragraph item*

```
/descendant-or-self::listitem/descendant-or-self::keyword
```

– **Parent**

Items are children of the world region they belong to. Since XPath does not allow disjunction at axis step level, one way to retrieve all the items belonging to either North or South America is to combine the parent axis with disjunction at filter level (another more costly solution is to use disjunction at query level).

Q5 *The (either North or South) American items*

```
/site/regions/*/item[parent::namerica or parent::samerica]
```

– **Ancestor axes**

Keyword elements may be arbitrarily deep in the document tree hence the ancestor operator in the following query may have to ascend the tree of an arbitrarily number of levels.

Q6 *The paragraph items containing a keyword*

```
//keyword/ancestor::listitem
```

Q7 *The mails containing a keyword*

```
//keyword/ancestor-or-self::mail
```

– **Sibling axes**

Bidders of a given open auction are siblings, and the XPath sibling axes may be exploited to explore them. As for query Q4 above, during the processing of query Q9, the XPath processor should take care to visit each bidder only once.

Q8 *The open auctions in which a certain person issued a bid before another person*

```
/site/open_auctions/open_auction
[bidder[personref/@person='person0']
/following-sibling::bidder[personref/@person='person1']]
```

Q9 *The past bidders of a given open auction*

```
/site/open_auctions/open_auction[@id='open_auction0']
/bidder/preceding-sibling::bidder
```

– **Following and Preceding**

Following and preceding are powerful axes since they may potentially traverse all the document in document or reverse document order. In particular, following and preceding explore more than following-sibling and preceding sibling. Compare query Q8 with query Q11: while in Q8 only sibling bidders are searched, in Q11 also bidders of different auctions are accessed.

Q10 *The items that follow, in document order, a given item*

```
/site/regions/*/item[@id='item0']/following::item
```

Q11 *The bids issued by a certain person that precedes, in document order, the last bid in document order of another person*

```
/site/open_auctions/open_auction
/bidder[personref/@person='person1']
/preceding::bidder[personref/@person='person0']
```

– **Attribute**

We have already used the attribute axis in combination with other axis. The following two queries focus on this axis in isolation.

Q12 *All the featured items*

```
//item[@featured='yes']
```

Q13 *The elements having declared an attribute* id

```
//*[@id]
```

– **Namespace**

XMark generated documents do not declare namespaces except the implicit one with prefix xml, which is declared at the document root and hence is valid for the entire document.

Q14 *Person elements in scope of some a namespace with prefix* xml

```
//person[namespace::xml]
```

– **Self**

The self axis is useful to filter elements according to their string values, like in the next query.

Q15 *The increases grater than 20*

```
//increase[. > 20]
```

- **Node tests**

  There are 8 node tests in XPath: `name`, `*`, `prefix:*`, `node()`, `comment()`, `processing-instruction()`, `processing-instruction('target')`, and `text()`. The first two node tests have been already demonstrated above (see, e.g., query Q1). The next six queries focus on the remaining node tests. Notice that XMark documents do not contain comments and processing instructions and do not define namespaces.

  Q16 *The elements whose namespace URIs are the same as the namespace URI to which the prefix* xml *is mapped*

  ```
  //xml:*
  ```

  Q17 *Any children node of the root*

  ```
  /node()
  ```

  Q18 *The children nodes of the root that are comments*

  ```
  /comment()
  ```

  Q19 *The children nodes of the root that are processing instructions*

  ```
  /processing-instruction()
  ```

  Q20 *The children nodes of the root that are processing instructions with target robots*

  ```
  /processing-instruction('robots')
  ```

  Q21 *The text nodes that are contained in the keywords of the description element of a given item*

  ```
  /site/regions/*/item[@id='item0']
  /description//keyword/text()
  ```

- **Boolean operators**

  Queries may be disjuncted with the | operator, while filters may be arbitrarily combined with conjunction, disjunction, and negation. This calls for the implementation of intersection, union, and set difference on context sets. These operations might be expensive if the XPath engine does not maintain (document) sorted the context sets.

  Q22 *The (either North or South) American items*

  ```
  /site/regions/namerica/item | /site/regions/samerica/item
  ```

  Q23 *People having an address and either a phone or a homepage*

```
/site/people/person[address and (phone or homepage)]
```

Q24 *People having no homepage*

```
/site/people/person[not(homepage)]
```

- **References**

  References break down the typical tree symmetry of XML documents. A reference may potentially point to any node in the document having an attribute of type ID. Chasing a reference implies the ability of coping with arbitrary jumps in the document tree. However, references are crucial to avoid redundancy in the XML database and to implement joins in the query language. In summary, references provide data and query flexibility and they pose new challenges to the query processors.

  Reference chasing is implemented in XPath with the function `id()` and may be static, like in query Q25, or dynamic, like in queries Q26-Q29. The `id()` function may be nested (like in query Q27) and its result may be filtered (like in query Q28). The `id()` function may also be used inside filters (like in query Q29).

  Q25 *The name of a given person*

  ```
  id('person0')/name
  ```

  Q26 *The open auctions that a given person is watching*

  ```
  id(/site/people/person[@id='person1']/watches/watch/@open_auction)
  ```

  Q27 *The sellers of the open auctions that a given person is watching*

  ```
  id(id(/site/people/person[@id='person1']/watches/watch/@open_auction)
  /seller/@person)
  ```

  Q28 *The American items bought by a given person*

  ```
  id(/site/closed_auctions/closed_auction[buyer/@person='person4']
  /itemref/@item)[parent::namerica or parent::samerica]
  ```

  Q29 *The items sold by Alassane Hogan*

  ```
  id(/site/closed_auctions/closed_auction
  [id(seller/@person)/name='Alassane Hogan']/itemref/@item)
  ```

- **Functions**

  XPath defines 27 built-in functions for use in XPath expressions. We have categorized them as follows:

– **Node-set functions**
These are the following functions: `position()`, `last()`, `count()`, `id()`, `local-name()`, `name()`, `namespace-uri()`, and `lang()`. The function `id()` has been already demonstrated above.

Q30 *The initial and last bidder of all open auctions*

```
/site/open_auctions/open_auction
/bidder[position()=1 and position()=last()]
```

Q31 *The open auctions having more than 5 bidders*

```
/site/open_auctions/open_auction[count(bidder)>5]
```

Q32 *The elements with local name* item

```
//*[local-name()='item']
```

Q33 *The elements with qualified prefixed name* svg:item

```
//*[name()='svg:item']
```

Q34 *The elements with a defined namespace URI*

```
//*[boolean(namespace-uri())]
```

Q35 *The elements written in Italian language*

```
//*[lang('it')]
```

– **String functions**
This section contains examples about the following functions: `concat()`, `contains()`, `normalize-space()`, `starts-with()`, `string()`, `string-length()`, `substring()`, `substring-after()`, `substring-before()`, and `translate()`.

Q36 *The items whose description contains the word 'gold'*

```
/site/regions/*/item[contains(description,'gold')]
```

Q37 *People having a name starting with 'Ed'*

```
/site/people/person[starts-with(name,'Ed')]
```

Q38 *Mails sent on the 10th*

```
/site/regions/*/item/mailbox/mail[substring-before(date,'/')='10']
```

Q39 *Mails sent in September*

```
/site/regions/*/item/mailbox/mail
[substring-before(substring-after(date,'/'),'/')='09']
```

Q40 *Mails sent in 1998*

```
/site/regions/*/item/mailbox/mail
[substring-after(substring-after(date,'/'),'/')='1998']
```

Q41 *Mails sent in the 21st century*

```
/site/regions/*/item/mailbox/mail[substring(date,7,2)='20']
```

Q42 *Items with a space-normalized description longer than 1000 characters*

```
/site/regions/*/item
[string-length(normalize-space(string(description))) > 1000]
```

Q43 *People with an address longer than 30 characters*

```
/site/people/person[string-length(translate(concat(
address/street,address/city,address/country,address/zipcode),
" ",""")) > 30]
```

– **Number functions**

This section contains examples about the following functions: `ceiling()`, `floor()`, `number()`, `round()`, and `sum()`.

Q44 *Open auctions with a total increase greater or equal to 70*

```
/site/open_auctions/open_auction
[floor(sum(bidder/increase)) >= 70]
```

Q45 *Open auctions with a total increase less or equal to 70*

```
/site/open_auctions/open_auction
[ceiling(sum(bidder/increase)) <= 70]
```

Q46 *Open auctions with an average increase greater than 8*

```
/site/open_auctions/open_auction
[round((number(current) - number(initial)) div count(bidder))
> 8]
```

– **Boolean functions**

This section contains examples about the following functions: `boolean()`, `true()`, `false()`, and `not()`.

Q47 *People with both an email address and a homepage*

```
/site/people/person
[boolean(emailaddress) = true() and not(boolean(homepage))
= false()]
```

XMark documents do not contain any comment or processing instruction. Moreover, they do no declare namespaces and language attributes. Although we have used these features above, the corresponding queries do not give interesting insights when evaluated on XMark documents, since their answer sets are trivial. Therefore, we have included in the benchmark a second document and a different set of queries in order to test these features only. The document has been adapted from Example 4-4 in [1]. The benchmark queries are the following:

A1 *The children nodes of the root that are comments*

```
/comment()
```

A2 *The children nodes of the root that are processing instructions with target* robots

```
/processing-instruction('robots')
```

A3 *The elements in scope of a namespace with prefix* xlink

```
//*[namespace::xlink]
```

A4 *The elements in scope of an XLink namespace (a namespace bound to the URI http://www.w3.org/1999/xlink)*

```
//*[namespace::* = 'http://www.w3.org/1999/xlink']
```

A5 *The elements whose namespace URIs are the same as the namespace URI to which the prefix* xlink *is mapped*

```
//xlink:*
```

A6 *The elements in scope of a namespace with prefix* svg

```
//*[namespace::svg]
```

A7 *The elements in scope of a Scalable Vector Graphics namespace (a namespace bound to the URI http://www.w3.org/2000/svg)*

```
//*[namespace::* = 'http://www.w3.org/2000/svg']
```

A8 *The elements whose namespace URIs are the same as the namespace URI to which the prefix* svg *is mapped*

```
//svg:*
```

A9 *The elements with a defined namespace URI*

```
//*[boolean(namespace-uri())]
```

A10 *The elements with local name* ellipse

```
//*[local-name()='ellipse']
```

A11 *The elements with qualified prefixed name* svg:ellipse

```
//*[name()='svg:ellipse']
```

A12 *The elements written in Italian language*

```
//*[lang('it')]
```

4 Evaluation of XML engines

Having the benchmark, it is possible to run the benchmark against a given XML processor and draw conclusions about the performance of the XML processor. In this section, we suggest a methodology for this evaluation.

We first describe a set of performance indexes that might help the evaluation and the comparison of XML processors with respect the the proposed benchmark. We say that a query is *supported* by an engine if the engine processes the query without giving an error. We say that a query is *correct* with respect to an engine if it is supported and the engine returns the correct answer for the query. We define the *completeness index* (*com*, for short) as the number of supported queries divided by the number of benchmark queries, and the *correctness index* (*cor*) as the number of correct queries divided by the number of benchmark queries. Notice that $0 \leq \text{cor} \leq \text{com} \leq 1$. Ideally, both completeness and correctness should be one. An acceptable situation in the one in which completeness is less than one, correctness is equal to completeness, and all the unsupported features are documented in the engine documentation.

XMark benchmark offers a document generator that generates XML documents of different sizes according to a numeric scaling factor. The scaling factor grows linearly with respect to the document size. For instance, factor 0.01 corresponds to a document of (about) 1,16 MB and factor 0.1 corresponds to a document of (about) 11,6 MB. Given an XMark document and a benchmark query, we can measure the time that the XML engine takes to evaluate the queries on the document. Usually, the query is ran a number of times, and the average is computed. The *query response time* (*qrt*) is the time taken by the engine to give the answer for the query, including parsing of the document, parsing, optimization, and processing of the query, and serialization of the results. We define the *query response speed* (*qrs*) as the size of the document divided by the query response time. The measure unit is, for instance, MB/sec. If we run the query against a documents series of documents of increasing size, we can generate a *speed sequence* for the query. The *average query response speed* (*aqrs*) is the average of the query response speeds.

Moving from one document (size) to another, the engine may show either a positive or a negative *acceleration* in its response speed, or the speed may remain constant. The concept of speed acceleration is intimately connected to that of *data scalability*. Indeed, consider two documents with sizes s_1 and s_2. Let t_1 and t_2 be the response times of the engine for a given query evaluated on the documents with sizes s_1 and s_2, respectively. The *data scalability factor* is defined as:

$$\frac{t_2 \cdot s_1}{t_1 \cdot s_2}$$

Notice that the scalability factor is always bigger than 0. If it is lower than

1, then we say that the scalability is *sub-linear*. If it is equal to 1, then we say that the scalability is *linear*. Finally, if it is grater than 1, then we say that the scalability is *super-linear*. A sub-linear scalability corresponds to a negative acceleration in the engine speed (a deceleration), while a super-linear scalability corresponds to a positive acceleration in the engine speed. If the scalability is linear, then the engine speed remains constant. If we run the query against a document series of documents of increasing size, we can generate a *data scalability sequence* and compute the *average data scalability* (*ads*) for the considered query.

All these indexes can be computed for a single query or for an arbitrary subset of the benchmark (taking the average of the query times). Of particular interest is the case when the whole benchmark is considered. Moreover, instead of checking the query response time, it might be interesting to evaluate the *query processing time*, which is the fraction of the query response time that the engine takes to process the query only, excluding the parsing of the document and the serialization of the results.

We can summarize the most relevant benchmark performance indexes we have discussed as follows:

- The **completeness** index (com). This gives an indication of how much of the benchmark is supported by the engine.
- The **correctness index** (cor). This indicates how much of the benchmark is correctly implemented by the engine.
- The **average query response speed** (aqrs). If we compute this index for the entire benchmark, it indicates how fast the engine processes the benchmark. If the benchmark and the document series on which the benchmark is ran are wide enough, then this index gives a precise indication of the efficiency of the XML engine with respect to the benchmark language (XPath 1.0 in our case).
- The **average data scalability** (ads). If we compute this index for the entire benchmark, it indicates how well the XML engine scales up with respect to the data size. An engine scales up well with respect to the data size if its data scalability is linear or better if it is sub-linear.

If we normalize these four indexes in the interval $[0, 1]$ (in fact, com and cor already belong to this interval) and possibly attach to them different weights w_i such that $w_1 + w_2 + w_3 + w_4 = 1$, we can aggregate them obtaining the following unique numerical index:

$$\iota = w_1 \cdot \mathrm{com} + w_2 \cdot \mathrm{cor} + w_3 \cdot \mathrm{aqrs} + w_4 \cdot \mathrm{ads}$$

that tells us how good is the XML engine with respect to our benchmark and our evaluation parameters (of course, it tells us nothing about what is not included in the evaluation).

The outcomes of the evaluation for a specific XML engine should be formatted in XML. In Figure 2 we suggest a DTD for this purpose. The document root is named `benchmark`. The engine under evaluation and its version are specified as attributes of the element `benchmark`. The benchmark element has an `index` and a number of `query` children.

The `index` element contains the performance indexes of the engine and has attributes describing the testing environment (like the CPU and the amount of memory of the used machine). The performance indexes are the following: completeness, correctness, a series of query response times (varying the document size, computed for the whole benchmark), a series of query response speeds (varying the document size, computed for the whole benchmark), a series of scalability factors (varying the document size, computed for the whole benchmark), the average query response speed (over the chosen document series, computed for the whole benchmark), and finally the average data scalability (over the chosen document series, computed for the whole benchmark). Notice that the query response time (`qrt`) and the query response speed (`qrs`) elements have an attribute called `factor` indicating the factor of the XMark document for which the time and speed have been computed. The scalability factor element (`scala`) has two attributes called `factor1` and `factor2` indication the factors of the two XMark documents for which the scalability factor has been computed.

Each `query` element contains information about the single query and is identified by an attribute called `id` of type ID. In particular, it includes the type of the query, a description in English, the XPath syntax, whether or not the query is supported by the benchmarked engine, the possible error message (only if the query is not supported), whether or not the query is correctly implemented in the benchmarked engine, the given and expected query answers (only if the query is not correct), a series of query response times (varying the document size, computed for the considered query), a series of query response speeds (varying the document size, computed for the considered query), a series of scalability factors (varying the document size, computed for the considered query), the average query response speed (over the chosen document series, computed for the considered query), and finally the average data scalability (over the chosen document series, computed for the considered query).

The solution of composing the results in XML format has a number of advantages. First, the outcomes are easier to extend with different evaluation parameters. More importantly, the outcomes can be queried to extract relevant information and to compute performance indexes. For instance, the following XPath query retrieves the benchmark queries that are supported but not correctly implemented:

```
/benchmark/query[supported="yes" and correct="no"]/syntax
```

Moreover, the following XQuery computes the completeness and correctness indexes:

```
let $x := doc("outcome_engine.xml")/benchmark/query
let $y := $x[supported="yes"]
```

```
let $z := $x[correct="yes"]
return <indexes>
         <completeness> {count($y) div count($x)} </completeness>
         <correctness> {count($z) div count($x)} </correctness>
       </indexes>
```

Finally, the following XQuery computes the average query response time of queries in the axes category when evaluated on the XMark document with scaling factor 0.1:

```
let $x := doc("outcome_engine.xml")/benchmark/query[type="axes"]
let $y := sum($x/times/qrt[@factor="0.1"])
let $z := count($x)
return <average_axes> {$y div $z} </average_axes>
```

More generally, the major advantage of the XML format of the benchmark and of its evaluation results is that is it easier to program an application that automatically checks the benchmark on a given XML engine, computes some performance indexes, and rates the XML engine.

5 Future work

We have a lot of ideas for the future:

1. To develop a benchmark to test *query scalability*. Query scalability is the ability of an XML engine to process queries of increasing lengths. Human-composed queries tend to be short, but computer-generated ones might be very long. This benchmark should include "crash-me queries", which are queries that seriously challenge current XML engines but that eventually might be solved efficiently.

2. To devise a larger set of *performance indexes* for the evaluation of XML engines.

3. To program a *benchmark checker* to automatically check the benchmark on a particular XML engine. The application should compute some performance indexes for the given engine and possibly compare them with other performances.

4. All this for XPath 2.0 [6]. The expressive power of XPath 2.0 lies between XPath 1.0 and XQuery.

References

[1] E. R. Harold and W. S. Means. *XML in a Nutshell*. O'Reilly, 2nd edition, 2002.

[2] A. R. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, and R. Busse. XMark: A Benchmark for XML Data Management. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 974–985, 2002. URL: `http://monetdb.cwi.nl/xml/`.

[3] World Wide Web Consortium. XML path language (XPath) version 1.0. URL: `http://www.w3.org/TR/xpath`, 1999.

[4] World Wide Web Consortium. XSL Transformations (XSLT). URL: `http://www.w3.org/TR/xslt`, 1999.

[5] World Wide Web Consortium. XML Pointer Language (XPointer). URL: `http://www.w3.org/TR/xptr`, 2002.

[6] World Wide Web Consortium. XML path language (XPath) version 2.0. URL: `http://www.w3.org/TR/xpath20`, 2005.

[7] World Wide Web Consortium. XQuery 1.0: An XML Query Language. URL: `http://www.w3.org/TR/xquery`, 2005.

[8] XPathMark: An XPath benchmark for XMark. `http://www.science.uva.nl/~francesc/xpathmark`.

```
<!ELEMENT benchmark        (indexes,query*)>

<!ELEMENT indexes          (completeness,correctness,
                            times?,speeds?,scalas?,aqrs?,ads?)>
<!ELEMENT completeness     (#PCDATA)>
<!ELEMENT correctness      (#PCDATA)>
<!ELEMENT times            (qrt+)>
<!ELEMENT speeds           (qrs+)>
<!ELEMENT qrt              (#PCDATA)>
<!ELEMENT qrs              (#PCDATA)>
<!ELEMENT scalas           (scala+)>
<!ELEMENT scala            (PCDATA)>
<!ELEMENT aqrs             (#PCDATA)>
<!ELEMENT ads              (#PCDATA)>

<!ELEMENT query            (type,description,syntax,supported,error?,
                            correct,given_answer?,expected_answer?,
                            times?,speeds?,scalas?,aqrs?,ads?)>
<!ELEMENT type             (#PCDATA)>
<!ELEMENT description      (#PCDATA)>
<!ELEMENT syntax           (#PCDATA)>
<!ELEMENT supported        EMPTY>
<!ELEMENT error            (#PCDATA)>
<!ELEMENT correct          EMPTY>
<!ELEMENT given_answer     (#PCDATA)>
<!ELEMENT expected_answer  (#PCDATA)>

<!ATTLIST benchmark engine    CDATA #REQUIRED
                    version   CDATA #REQUIRED>
<!ATTLIST query     id        ID #REQUIRED>
<!ATTLIST indexes   cpu       CDATA #IMPLIED
                    memory    CDATA #IMPLIED
                    os        CDATA #IMPLIED
                    time_unit (msec | csec | dsec | sec)  #IMPLIED
                    time_type (elapsed | cpu) #IMPLIED>
<!ATTLIST supported value     (yes | no) #REQUIRED>
<!ATTLIST correct   value     (yes | no) #REQUIRED>
<!ATTLIST qrt       factor    CDATA #REQUIRED>
<!ATTLIST qrs       factor    CDATA #REQUIRED>
<!ATTLIST scala     factor1   CDATA #REQUIRED
                    factor2   CDATA #REQUIRED>
```

Figure 2: The DTD for a benchmark outcome