EMBEDDING XPATH QUERIES INTO SPARQL QUERIES

Matthias Droop[1], Markus Flarer[1], Jinghua Groppe[2], Sven Groppe[2], Volker Linnemann[2]
Jakob Pinggera[1], Florian Santner[1], Michael Schier[1], Felix Schöpf[1]
Hannes Staffler[1] and Stefan Zugal[1]
[1] University of Innsbruck, Technikerstrasse 21a, A-6020 Innsbruck, Austria

[2] Institute of Information Systems, University of Lübeck, Ratzeburger Allee 160, D-23538 Lübeck, Germany

Keywords: XML, XPath, Semantic Web, SPARQL, RDF.

Abstract: While XPath is an established query language developed by the W3C for XML, SPARQL is a new query language developed by the W3C for RDF data. Comparisons between the data models of XML and RDF and between the query languages XPath and SPARQL are missing. Since XML and XPath are earlier recommendations of the W3C than RDF and SPARQL, currently more XML data and XPath queries are used in applications. However, recently available SPARQL query evaluators do not deal with XML data and XPath queries. We have developed a prototype for translating XML data into RDF data and embedding XPath queries into SPARQL queries for the following two reasons: 1) We want to compare the XPath and XQuery data model with the RDF data model and the XPath query language with the SPARQL query language in order to show similarities and differences. 2) We want to enable SPARQL query evaluators to deal with XML data and XPath queries in order to support XPath processing and SPARQL processing in parallel. We have developed a prototype for the source-to-source translations from XML data into RDF data and from XPath queries into SPARQL queries. We have run experiments to measure the execution times of the translations, of XPath queries and of their translated SPARQL queries.

1 INTRODUCTION

XML is a data format for exchanging data on the web, between databases and elsewhere. Furthermore, XML has become a widely used native data format. The W3C has developed XPath (W3C, 2007) as a query language for XML data. XPath is embedded in many other languages like the XQuery query language and the XSLT language for transforming XML data. The name of XPath derives from its basic concept, the path expression, with which the user can hierarchically address the nodes of the XML data. The user of XPath may not only use simple relationships like parent-child, but also more complex relationships like the descendant relationship, which is the transitive closure of the parent-child relationship. Furthermore, complex filter expressions are allowed in XPath queries.

The Resource Description Framework (RDF) (Carroll and Klyne, 2004) is a language for representing information about resources in the World Wide Web. SPARQL (Prud'hommeaux and Seaborne, 2008) is a query language for formulating queries against RDF graphs. SPARQL supports querying by triple patterns, conjunctions, disjunctions, and optional patterns, and constraining queries by a source RDF graph and extensible value testing. Results of SPARQL queries can be ordered, limited and offset in number. There are plans to embed SPARQL in forthcoming languages similar to XPath that is embedded in XQuery and XSLT.

In recent years, RDF storage systems, which support or plan to support SPARQL, have occurred like Jena (Wilkinson et al., 2003), Sesame (Broekstra et al., 2003), rdfDB (Guha, 2006), Redland (Beckett, 2002), Kowari (Northrop Grumman Corporation, 2006), RDF Suite (Alexaki et al., 2001) and Allegro (Franz Inc., 2006). These RDF storage systems do not support XML data and XPath queries, which are currently widely used in applications. Integration of XML data into RDF data and embedding of XPath queries into SPARQL queries can make XML data and XPath available for these products. Furthermore, the proposed embedding enables users to work in parallel with XML data and RDF data, and with XPath queries and SPARQL queries, i.e. XML data is integrated in RDF data and the result of XPath subqueries can be used in SPARQL for further processing. As many SPARQL tools do not allow calling an external XPath evaluator from SPARQL, we propose to translate embedded XPath queries into SPARQL subexpressions.

Tree-based queries can be easier expressed in the tree query language XPath in comparison to the graph query language SPARQL. For example, SPARQL does not allow computing all descendant nodes of a node like XPath does. Furthermore, the formulation of joins in graphs is easier in SPARQL than in XPath. An embedding of XPath into SPARQL allows the easy formulation of tree queries and graph queries in one query. Thus, the host language SPARQL benefits from the embedded language XPath and the embedded language XPath benefits from its host language SPARQL.

In this paper, we first compare the different data models of XML and RDF and the different query languages XPath and SPARQL. Based on the comparison, we propose a translation scheme for XML data into RDF data and XPath queries into SPARQL queries. Furthermore, we present the results of an experimental evaluation of a prototype, which shows that various different XPath queries can be embedded into SPARQL.

2 FURTHER RELATED WORK

There are many contributions (see (Florescu and Kossmann, 1999), (Georgiadis and Vassalos, 2006), (Grust et al., 2004), (Tatarinov et al., 2002), (Manolescu et al., 2001), (Shanmugasundaram et al., 1999), (Subramanyam, and Kumar, 2005), (Fan et al., 2005) and (Yoshikawa et al., 2001)) to source-to-source translations for evaluating XPath expressions on relational database management systems. Many techniques described there can be adapted for evaluating XPath expressions on SPARQL evaluators like using a numbering scheme for the XML data to support all XPath axes (Grust et al., 2004), but some other techniques cannot be adapted like the evaluation of positional predicates (Tatarinov et al., 2002) as SQL supports more language constructs than SPARQL like the rank clause.

(Bettentrupp et al., 2006), (Fokoue et al., 2005), (Klein et al., 2005) and (Lechner et al., 2001) focus on the translations between XSLT and XQuery (and vice versa), which embed the XPath language, where the XSLT and XQuery languages are based on the same data model.

Furthermore, some contributions deal with bridging the gap between SPARQL/RDF and the relational world (see (Chong et al., 2005), (Dokulil, 2006), (Harris and Shadbolt, 2005) and (de Laborda and Conrad, 2006)).

(Groppe et al., 2008) describes a translation scheme from SPARQL to XQuery/XSLT.

(Droop et al., 2007) deals with a translation from XML data into RDF data and from XPath queries into SPARQL queries, but it neither contains the general translation algorithm nor an experimental evaluation. In this work, we suggest a translation algorithm and present the results of an experimental evaluation, which demonstrates which kind of XPath queries can be translated into SPARQL and which kind of embedded XPath queries are fast processed when using SPARQL evaluators. Furthermore, (Droop et al., 2007) does not deal with an *embedding* of XPath queries into SPARQL queries.

3 COMPARISON OF XML/RDF AND XPATH/SPARQL

We describe the XML and RDF data models and introduce the XPath and SPARQL query languages in the following subsections (see Section 3.1 and Section 3.2). Furthermore, we describe the similarities of the data models and query languages in Section 3.3 and the differences in Section 3.4.

3.1 XPath and XQuery Data Model and XPath Query Language

The XPath and XQuery data model is defined as follows:

Definition 1 (*Data Model of XPath and XQuery*). An XML document is a tree of nodes. The kinds of nodes are *document*, *element*, *attribute*, *text*, *namespace*, *processing-instruction*, and *comment*. Every node has a unique node identity that distinguishes it from other nodes. In addition to nodes, the data model allows atomic values, which are single values that correspond to the simple types defined in (W3C, 2001), such as *strings*, *Booleans*, *decimal*, *integers*, *floats*, *doubles*, and *dates*. The first node in any document is the document node, which contains the entire document. Element nodes, comment nodes, and processing instruction nodes occur in the order in which they are found in the textual representation of the XML document.

Element nodes occur before their children – the element nodes, text nodes, comment nodes, and processing instructions, which they contain. Attribute nodes and namespace nodes are not considered as children of an element.

See Figure 1 for an example of a textual XML document representing a bookstore containing the two books “Harry Potter” from J. K. Rowling and “Learning XML” from Erik T. Ray. Figure 2 is a graphical representation of the XML document of Figure 1.

```
<bookstore>
  <book category="CHILDREN">
    <title>Harry Potter</title>
    <author>J. K. Rowling</author></book>
  <book category="WEB">
    <title>Learning XML</title>
    <author>Erik T. Ray</author></book>
</bookstore>
```

Figure 1: An example XML document representing a bookstore.

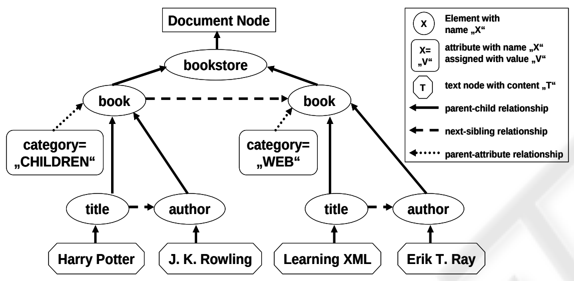

Figure 2: Graphical representation of the XML document of Figure 1.

The W3C developed the XPath language as a simple query language to describe node sets of XML documents. The basic concept of XPath expressions are location steps separated by a slash ("/"). Each location step of the form a::nt[P1]...[Pn] contains

- an axis a, which can be one of child, attribute, self, parent, descendant, descendant-or-self, ancestor, ancestor-or-self, following, following-sibling, preceding and preceding-sibling.
- a node test nt. Among the possible node tests are a name node test for a specific name A (declared by A itself), for an arbitrary name (declared by the wildcard *), for a text node (declared by text()), and a node test node() for all node types.
- an arbitrary number of predicates P1 to Pn. A predicate is enclosed by the brackets [and]. A predicate contains a Boolean expression, e.g. a comparison with a constant string or number.

Starting with the root node of an XML document, each location step from left to right describes, which XML nodes must be considered for the next location step by following the axis for the XML nodes of the previous location step, checking the node test and the predicates. The whole XPath expression describes the resultant XML nodes of the last location step. For example, the resultant nodes of the XPath query of Figure 3 with input XML document of Figure 1 are presented in Figure 4.

```
/bookstore/parent::node()/descendant::title/text()
```

Figure 3: An example XPath query.

```
Harry Potter
Learning XML
```

Figure 4: Resultant text nodes when applying the XPath query of Figure 3 to the input XML document of Figure 1.

3.2 RDF Data Model and SPARQL

In comparison to the XPath and XQuery data model, the data model of RDF is defined as follows:

Definition 2 (*Data Model of RDF*). The underlying structure of any expression in RDF is a collection of triples, each consisting of a subject (a RDF URI reference or a blank node), a predicate (a RDF URI reference) and an object (a RDF URI reference, a blank node or a literal, which can be plain literals having optionally a language tag, or which can be a typed literal having additionally a datatype URI being a RDF URI reference). A set of such triples is called an RDF graph. The nodes of an RDF graph are its subjects and objects. The predicate holds a directed relationship from a subject to an object.

There are different ways to represent RDF data, e.g. RDF Triplets or RDF/XML documents, which use XML to encode RDF data**.** Figure 5 contains an example RDF/XML document, which actually represents the generated RDF graph of the data translation module of our prototype when using the XML data of Figure 1 as input. Figure 6 is a graphical representation of the RDF data of Figure 5.

SPARQL (see (Prud’hommeaux and Seaborne, 2008)) is a query language for retrieving information from RDF graphs stored in semantic storage systems. The outline query model is graph patterns expressed by simple triple patterns. It does not use rules and is not path based.

We briefly introduce SPARQL by a simple example. Figure 7 presents a SPARQL query, which actually is a query translated by our prototype with the input XPath query of Figure 3. The query consists of three parts, the PREFIX declarations, the SELECT clause and the WHERE clause. The PREFIX declarations specify prefixes for short name usages (e.g. here rel is declared as short name for <http://uibk.ac.at/informatic/comdesign/relations:>). The

SELECT clause identifies the variables to appear in the query results (here ?v9). The WHERE clause contains triple patterns like "?v0 rel:child ?v1.". The first position (?v0) in the triple pattern represents the constraints or bindings to variables for the subjects in the RDF data. The second position (here it is the relation rel:child) contains the constraints or bindings to variables for predicates of the triples of the RDF data, and the third (here ?v1) contains the constraints or bindings to variables for the objects of the triples of the RDF data. A join between the first triple pattern (?v0 rel:child ?v1.) and the second triple pattern (?v1 rel:type "1".) of the query is expressed by using the same variable ?v1 in both triple patterns. Furthermore, SPARQL queries may contain filter expressions (here e.g. FILTER (xsd:long(?v6) > xsd:long(?v4)).), which contain boolean expressions to constrain the input data (here the considered filter expression contains a greater-than comparison (">") between the two variables ?v6 and ?v4, which are first castet to the XML Schema datatype **long**).

```
<rdf:RDF xmlns:rel=
  "http://uibk.ac.at/informatic/comdesign/relations:"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
  <rdf:Description rdf:nodeID="A0">
    <rel:end>10</rel:end><rel:child rdf:nodeID="A1"/>
    <rel:start>7</rel:start><rel:type>1</rel:type>
    <rel:name>author</rel:name>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A2">
    <rel:type>2</rel:type><rel:value>WEB</rel:value>
    <rel:name>category</rel:name>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A3">
    <rel:end>6</rel:end><rel:child rdf:nodeID="A4"/>
    <rel:start>3</rel:start><rel:type>1</rel:type>
    <rel:name>title</rel:name>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A5">
    <rel:end>20</rel:end><rel:child rdf:nodeID="A6"/>
    <rel:start>17</rel:start><rel:type>1</rel:type>
    <rel:name>author</rel:name>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A1">
    <rel:end>9</rel:end><rel:type>3</rel:type>
    <rel:value>J. K. Rowling</rel:value>
    <rel:start>8</rel:start>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A7">
    <rel:end>16</rel:end><rel:child rdf:nodeID="A8"/>
    <rel:start>13</rel:start><rel:type>1</rel:type>
    <rel:name>title</rel:name>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A9">
    <rel:type>2</rel:type><rel:value>CHILDREN</rel:value>
    <rel:name>category</rel:name>
  </rdf:Description>
  <rdf:Description rdf:about="file:///C:/bookstore.xml">
    <rel:end>23</rel:end><rel:child rdf:nodeID="A10"/>
    <rel:type>9</rel:type><rel:start>0</rel:start>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A8">
    <rel:end>15</rel:end><rel:type>3</rel:type>
    <rel:value>Learning XML</rel:value>
    <rel:start>14</rel:start>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A11">
    <rel:end>11</rel:end><rel:child rdf:nodeID="A0"/>
    <rel:child rdf:nodeID="A3"/>
    <rel:attribute rdf:nodeID="A9"/>
    <rel:start>2</rel:start><rel:type>1</rel:type>
    <rel:name>book</rel:name>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A4">
    <rel:end>5</rel:end><rel:type>3</rel:type>
    <rel:value>Harry Potter</rel:value>
    <rel:start>4</rel:start>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A10">
    <rel:end>22</rel:end><rel:child rdf:nodeID="A12"/>
    <rel:child rdf:nodeID="A11"/>
    <rel:start>1</rel:start><rel:type>1</rel:type>
    <rel:name>bookstore</rel:name>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A12">
    <rel:end>21</rel:end><rel:child rdf:nodeID="A5"/>
    <rel:child rdf:nodeID="A7"/>
    <rel:attribute rdf:nodeID="A2"/>
    <rel:start>12</rel:start><rel:type>1</rel:type>
    <rel:name>book</rel:name>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A6">
    <rel:end>19</rel:end><rel:type>3</rel:type>
    <rel:value>Erik T. Ray</rel:value>
    <rel:start>18</rel:start>
  </rdf:Description>
</rdf:RDF>
```

Figure 5: RDF/XML document representing the generated RDF graph of the data translation module of our prototype when using the XML data of Figure 1 as input.

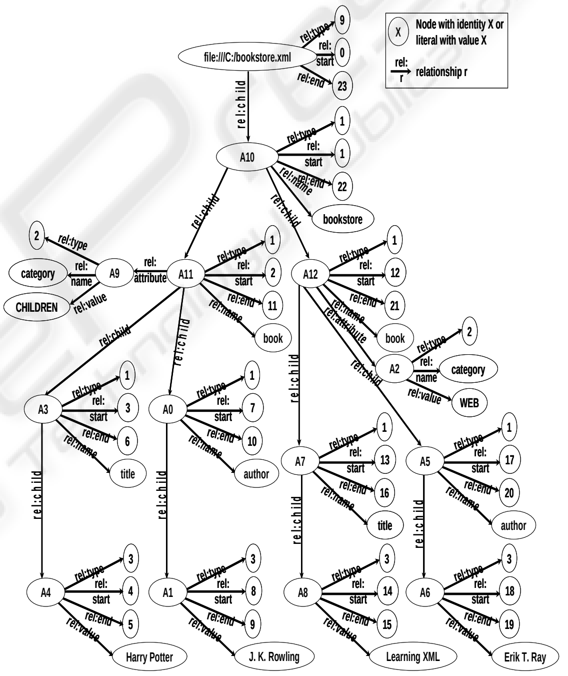

Figure 6: Graphical representation of the RDF data of Figure 5.

Figure 8 presents the XML representation of the resultant bindings of the SPARQL query of Figure 7 applied to the RDF data of Figure 5.

There are further constructs to e.g. use built-in functions and set operations like the UNION operator. We refer the interested reader to (Prud'hommeaux and Seaborne, 2008) for a complete list and description of the SPARQL features.

```
PREFIX rel: <http://uibk.ac.at/relations/>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
SELECT ?v9 WHERE { ?v0 rel:type "9".
              ?v0 rel:child ?v1.
              ?v1 rel:name "bookstore".
              ?v2 rel:child ?v1.
              ?v7 rel:start ?v3.
```

```
            ?v2 rel:start ?v5.
                ?v7 rel:end ?v4.
                ?v2 rel:end ?v6.
                ?v7 rel:name "title".
                ?v7 rel:child ?v8.
                ?v8 rel:type "3".
                ?v8 rel:value ?v9.
                FILTER(xsd:long(?v6)>xsd:long(?v4)).
                FILTER(xsd:long(?v5)<xsd:long(?v3)).}
```

Figure 7: Translated SPARQL query when using the XPath query of Figure 3 as input of our prototype.

```
<?xml version="1.0"?>
<sparql
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xs="http://www.w3.org/2001/XMLSchema#"
    xmlns="http://www.w3.org/2005/sparql-results#" >
  <head><variable name="v10"/></head>
  <results ordered="false" distinct="false">
   <result><binding name="v10">
            <literal>Harry Potter</literal>
          </binding></result>
   <result><binding name="v10">
            <literal>Learning XML</literal>
          </binding></result>
  </results>
</sparql>
```

Figure 8: Result of the SPARQL query of Figure 7 when we use the RDF graph of Figure 5 as input.

3.3 Similarities

Both query languages, XPath and SPARQL, support complex queries and support the usage of variables, constraining the result of queries (see predicates **[…]** in XPath and **FILTER** expressions in SPARQL) and joins using variables. XPath and SPARQL support iterating through an input data set (see **for** clauses in XPath and triple patterns in SPARQL). Furthermore, XPath and SPARQL have a rich set of built-in functions, some of which are equivalent (see e.g. function **fn:matches** in XPath and the equivalent function **regex** in SPARQL). Both, XPath and SPARQL, do not support user-defined functions. Both languages support conditional results (see e.g. **if-then-else** expressions in XPath and **OPTIONAL** patterns in SPARQL) and support nesting of their expressions and statements. Our translation scheme shows that many language constructs of XPath can be embedded into SPARQL expressions in a straightforward way. The supported datatypes in XPath queries are the datatypes of XML Schema, which are supported in SPARQL, too.

XML data represents a tree of information. RDF data consists of triple data, which expresses a graph. Nevertheless, XML data can contain information to represent graphs and RDF data can express trees. For our translation from XML to RDF, we translate the tree structure into triples, such that no information is lost of the content of each node and the relationship between the nodes of the XML tree. Furthermore, because of some differences between the XPath and SPARQL languages (see Section 3.4), we have to add a numbering scheme for the representation of the XML data in RDF.

3.4 Differences

XPath supports mechanisms to determine transitive closures by built-in mechanisms (e.g. using the **descendant** axis), but SPARQL does not support the determination of transitive closures. Furthermore, XPath supports nesting of expressions with full generality, but SPARQL does not support, e.g. querying the result of a SPARQL subquery formulated in one SPARQL query.

Both languages, XPath and SPARQL, define built-in functions, which do not have a corresponding built-in function in the other language (see e.g. **fn:replace** and simple aggregates like **fn:count** and **fn:max** in XPath, and **isIRI** and **isBound** in SPARQL). Some of these built-in functions of XPath can be expressed by SPARQL language constructs (e.g. **id** in XPath); some other might only be expressed in an external function formulated in another programming language. Cast operations in XPath queries of data of a not castable datatype lead to an error, which stops the evaluation of the XPath query, while casting in SPARQL queries within filter expressions constraints the input data.

XPath expressions return a single or a sequence of atomic values, which are single values that correspond to the simple types defined in (W3C, 2001), such as strings, Booleans, decimal, integers, floats, doubles, and dates, or a non-nesting, un-typed sequence of nodes, which are ordered according to the document order. The evaluation of SPARQL queries returns a set of bindings of variables. Thus, we have to provide a mechanism to transform the set of bindings of SPARQL results into typical XPath results in the case that SPARQL results are returned back from the SPARQL query.

The implicit type of each node of the XML tree has to be stored explicitly by a special relationship for translating XML data into RDF data. All implicit relationships of nodes of the XML tree have to be added as explicit relationships in the RDF data. This includes parent-child relationships, attribute and namespace relationships, and next-sibling relationships. As SPARQL does not support the determination of transitive closures, we have to add a numbering scheme (see Section 4.1) to the RDF data in order to support XPath axes, which require the determination of the transitive closure of basic relationships, like **descendant**, **ancestor**, **following** and **preceding**.

4 TRANSLATION OF XPATH QUERIES

We propose to embed XPath subqueries in SPARQL queries by binding a SPARQL variable to the result of an XPath subquery by using a `BIND(S, E)` construct in the WHERE clause of the host SPARQL query, which assigns the resultant nodes of an embedded XPath expression `E` to a SPARQL variable `S`. For example, we present a SPARQL query with an embedded XPath subquery in Figure 9, where the titles of all available books in a collection of books from a bookstore, the data of which is stored in the input XML document, are retrieved and represented by the SPARQL variable `?XPath` (see lines (5) to (7)). Furthermore, the retrieved result of the embedded XPath subquery is compared with the titles in the book collection of the user (see line (8)), which are stored in the input RDF document, and the current places of the books are determined (see line (9)). The title of the books available in both, in the bookstore and in the book collection of the user, is returned together with the current place of the book (see line (4)).

```
(1) PREFIX myCollection:
(2)       <http://uibk.ac.at/informatic/myCollection>
(3) PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
(4) SELECT ?XPath ?place WHERE
(5) { BIND(?XPath, /child::bookstore/
(6)                parent::node()/
(7)                descendant::title/child::text()).
(8)   ?x myCollection:title ?XPath.
(9)   ?x myCollection:place ?place. }
```

Figure 9: Host SPARQL query with embedded XPath query.

We first explain how to translate an XPath query into an equivalent SPARQL query. Afterwards, we explain how to integrate the translation of an embedded XPath subquery into its host SPARQL query.

The translation process consists of three steps (see Figure 10): (i) the translation of the input data from XML into RDF, (ii) the source-to-source translation from the XPath query into the translated SPARQL query, and (iii) in the case that the host SPARQL query returns the result of an embedded XPath query the translation of the result from the translated SPARQL query into the result according to the XPath and XQuery data model, which is equivalent to the result of the XPath query. We explain each translation step in more detail in Section 4.1 to Section 4.3.

4.1 Translation of Data

We translate the XML data into RDF data by a depth-first traversal of the XML tree and annotate each translated node of the XML data with the relationships **`rel:type`**, **`rel:child`**, **`rel:attribute`**, **`rel:namespace`**, **`rel:name`**, **`rel:value`**, **`rel:start`** and **`rel:end`**.

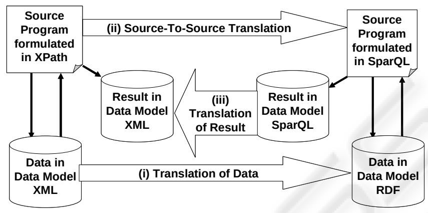
Source Program formulated in XPath
(ii) Source-To-Source Translation
Source Program formulated in SparQL
Result in Data Model XML
(iii) Translation of Result
Result in Data Model SparQL
Data in Data Model XML
(i) Translation of Data
Data in Data Model RDF

Figure 10: The translation process consists of three steps: (i) the translation of data from XML into RDF, (ii) the source-to-source translation from XPath into SPARQL, and (iii) the translation of the result, which is returned by the SPARQL query, from an embedded XPath query into the result in the data model XML, which is equivalent to the result of the XPath query.

As an example, see Figure 1 for the original XML data and see Figure 2 for its graphical representation, and see Figure 5 for the translated RDF data and see Figure 6 for its graphical representation.

We use a relationship **`rel:type`** in the RDF data in order to explicitly annotate the type of an XML node. The explicit relationship **`rel:child`** in the translated RDF data expresses a parent-child relationship in the XML tree, **`rel:attribute`** an attribute relationship, and **`rel:namespace`** a namespace relationship. The value associated over the relationship **`rel:name`** contains the name of the XML node, **`rel:value`** contains the value of the node. The relationships **`rel:start`** and **`rel:end`** are computed by a numbering scheme for XML data and their purpose are for the determination of the descendant relationships in SPARQL queries, as SPARQL does not support the determination of transitive closures.

We use a numbering scheme based on the region encoding on elements of the XML tree, which we adapt from (Grust et al., 2004): For each element, the values of **`rel:start`** and **`rel:end`** can be assigned by a depth-first traversal through the XML tree. The value of **`rel:start`** of the document tree is **`1`**. The value **`rel:end`** of a node **`v`** with start value **`n`** can be computed by **`n+count(subtree(v))+1`**, where the function **`count`** returns the number of nodes of the subtree rooted at **`v`**. The value of **`rel:start`** of the first child of a node **`v`** is **`v.start+1`**. The value of

`rel:start` of a non-first child is the value of **`rel:end`** of the previous sibling plus **`1`**.

Between any two elements **`a`** and **`b`** of the XML tree, **`a`** is a descendant node of **`b`**, if **`a.start>b.start`** and **`a.end<b.end`**. Analogously, **`a`** is an ancestor node of **`b`**, if **`a.start<b.start`** and **`a.end>b.end`**. **`a`** is a following-sibling (a preceding-sibling respectively) of **`b`**, if **`a.start>b.start`** (**`a.start<b.start`** respectively) and there exists a subject **`p`** such that **`p rel:child a`** and **`p rel:child b`** holds.

With these relationships, we can support all XPath axes in our translation scheme, as we can determine the nodes according to the basic relationships. Note that the XPath location step **`following::n`** is equivalent to **`ancestor-or-self::node()/following-sibling::node()/descendant-or-self::n`**, and the XPath location step **`preceding::n`** is equivalent to **`ancestor-or-self::node()/preceding-sibling::node()/descendant-or-self::n`**.

4.2 Translation of Queries

We translate an XPath query into an equivalent SPARQL query in the following way:

First, we determine the syntax tree of the XPath query. See Figure 11, which contains the syntax tree of the XPath query of Figure 3. This can be done by using standard compiler techniques, the input of which is the XPath grammar.

Second, we evaluate the attribute grammar, which we do not present here due to space limitations, on the syntax tree. This attribute grammar defines computation rules for each possible situation in the syntax tree. The computation rules compute attributes of the nodes of the syntax tree. Depending on the dependencies between the attributes in the computation rules, a tree walking algorithm defines the traversal of the syntax tree and the evaluation order of the attributes. After applying the tree walking algorithm, a special attribute **`SPARQL`** of the root node **`XPATH`** of the syntax tree contains the translated SPARQL query.

As an example of the query translation, see Figure 3 for the original XPath query, Figure 11 for its syntax tree with computed attributes according to our attribute grammar and Figure 7 for the translated SPARQL query.

The translation of embedded XPath subqueries and the integration of their translations into the host SPARQL query require that the resultant SPARQL query containing the result of the XPath subquery is renamed according to the bound variable of the extended SPARQL expression for embedding XPath queries. Furthermore, the declared prefixes must be added to the declared prefixes of the host SPARQL query and the WHERE clause of the translated XPath query has to be added to the WHERE clause of the host SPARQL query.

4.3 Translation of Result

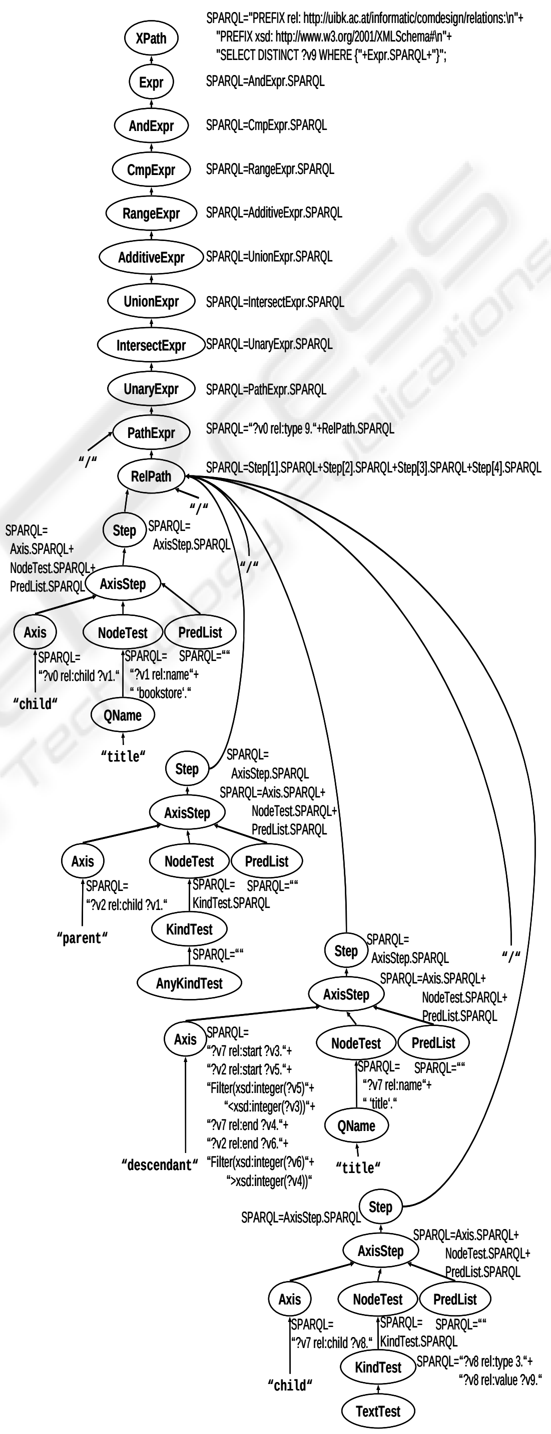
XPath
SPARQL="PREFIX rel: http://uibk.ac.at/informatic/comdesign/relations:\n"+ "PREFIX xsd: http://www.w3.org/2001/XMLSchema#\n"+ "SELECT DISTINCT ?v9 WHERE {"+Expr.SPARQL+"}";
Expr
SPARQL=AndExpr.SPARQL
AndExpr
SPARQL=CmpExpr.SPARQL
CmpExpr
SPARQL=RangeExpr.SPARQL
RangeExpr
SPARQL=AdditiveExpr.SPARQL
AdditiveExpr
SPARQL=UnionExpr.SPARQL
UnionExpr
SPARQL=IntersectExpr.SPARQL
IntersectExpr
SPARQL=UnaryExpr.SPARQL
UnaryExpr
SPARQL=PathExpr.SPARQL
PathExpr
SPARQL="?v0 rel:type 9."+RelPath.SPARQL
"/"
RelPath
SPARQL=Step[1].SPARQL+Step[2].SPARQL+Step[3].SPARQL+Step[4].SPARQL
"/"
Step
SPARQL= AxisStep.SPARQL
SPARQL= Axis.SPARQL+ NodeTest.SPARQL+ PredList.SPARQL
AxisStep
"/"
Axis
NodeTest
PredList
SPARQL= "?v0 rel:child ?v1."
SPARQL= "?v1 rel:name"+ " 'bookstore'."
SPARQL=""
"child"
QName
"title"
Step
SPARQL= AxisStep.SPARQL
SPARQL=Axis.SPARQL+ NodeTest.SPARQL+ PredList.SPARQL
AxisStep
Axis
NodeTest
PredList
SPARQL= "?v2 rel:child ?v1."
SPARQL= KindTest.SPARQL
SPARQL=""
"parent"
KindTest
SPARQL=""
AnyKindTest
Step
SPARQL= AxisStep.SPARQL
"/"
AxisStep
SPARQL=Axis.SPARQL+ NodeTest.SPARQL+ PredList.SPARQL
Axis
SPARQL= "?v7 rel:start ?v3."+ "?v2 rel:start ?v5."+ "Filter(xsd:integer(?v5)"+ "<xsd:integer(?v3))"+ "?v7 rel:end ?v4."+ "?v2 rel:end ?v6."+ "Filter(xsd:integer(?v6)"+ ">xsd:integer(?v4))"
NodeTest
PredList
SPARQL= "?v7 rel:name"+ " 'title'."
SPARQL=""
QName
"descendant"
"title"
Step
SPARQL=AxisStep.SPARQL
AxisStep
SPARQL=Axis.SPARQL+ NodeTest.SPARQL+ PredList.SPARQL
Axis
NodeTest
PredList
SPARQL= "?v7 rel:child ?v8."
SPARQL= KindTest.SPARQL
SPARQL=""
KindTest
SPARQL="?v8 rel:type 3."+ "?v8 rel:value ?v9."
"child"
TextTest

Figure 11: Syntax tree of the XPath query of Figure 3 and computed attributes according to our attribute grammar.

The result of an SPARQL query is a set of bindings of variables in the SELECT clause of an SPARQL query. We determine the translated XPath query in such a way that the retrieved bindings of the XPath query represent the resultant XML nodes of the original query. In the module of the translation of result and in the case that the result of an embedded XPath query is returned by the SPARQL query, we now rebuild the subtrees of these resultant XML nodes by considering the information of the original XML tree in the RDF data (especially the rel:type, rel:child, rel:attribute, rel:namespace, rel:name, rel:value, rel:start and rel:end relationships). Furthermore, we sort the resultant XML trees according to the document order of the original XML tree, as the XPath language specifies the result of an XPath query to be in document order of the queried XML document. Note that the less-than order relation of the computed values of rel:start correspond to the document order relation, i.e. if a and b are the values of the rel:start relationship of two nodes of the translated RDF data and a<b, then the corresponding node of a occurs before the corresponding node of b in document order in the original XML document.

5 PERFORMANCE ANALYSIS

The test system for all experiments is a 2.66 Gigahertz Intel Pentium 4 processor with 1 Gigabytes main memory. The test system runs Windows XP Professional Version 2002 Service Pack 2 and Java version 1.5. We have used the Java 1.5 internal XPath evaluator. Furthermore, we have used the XQuery evaluators Saxon (Kay, 2006), as Saxon is widely used, and Qizx (Axyana software, 2006), as Qizx is a fast evaluator, to process the original XPath queries. We have used Jena (Hewlett-Packard Labs, 2003) to process the translated SPARQL expressions, as Jena supports the current version of SPARQL and as Jena is the most widely used Semantic Web reasoning engine (see (Cardoso, 2007)). We present the average of 10 execution times of evaluating the original XPath queries, of the data translation, of the query translation, of processing the translated SPARQL queries and of the result translation.

For the original queries, we have used the queries for the performance test of the XPathMark (Franceschet, 2005) benchmark. The data is generated by the data generation tool of the benchmark, which allows scaling the size of the input data.

We have excluded those queries of the XPathMark benchmark, which are not supported by our prototype for the following reasons:

- The queries contain an XPath built-in function, which does not correspond to any built-in function of SPARQL. Our prototype supports the **`not`**, **`round`**, **`abs`**, **`floor`**, **`ceiling`** and **`substring`** function. One possible way to support other built-in functions is to use external functions, which are especially implemented to provide the same functionality as the corresponding XPath built-in function. We did not implement these external functions, as external functions are not standardized and thus depend on the used SPARQL engine.

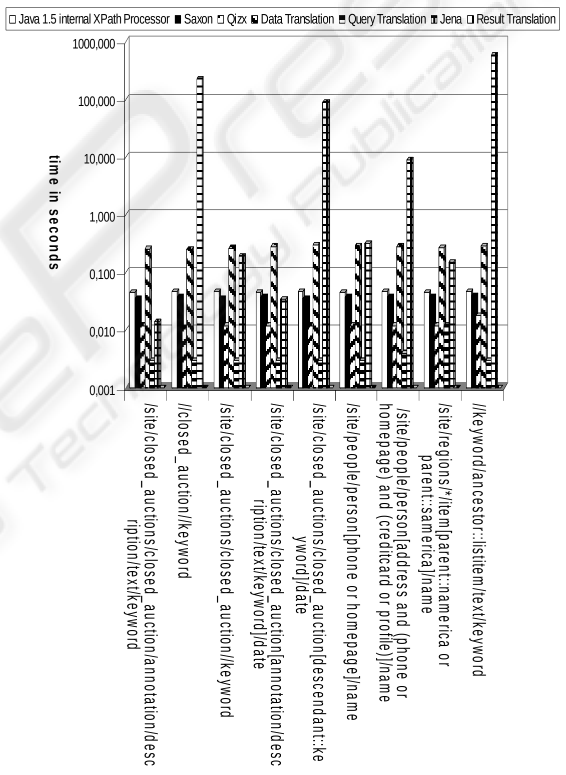

Figure 12: Execution time of the first subset of original XPath queries of the XPathMark Benchmark, of the data translation, of the query translation, of the translated SPARQL query and of the result translation when using an input XML document of size 56.55 Kilobytes.

- The queries contain predicates of the form **`[x]`**, where **`x`** is a number, which restrict the current node set of a location step to its **`x`**-th element. We are currently not aware of a simple, but efficient way to access the **`x`**-th element of a dynamically determined set in SPARQL queries. Note that techniques developed for SQL as described in

(Tatarinov et al., 2002) cannot be adapted to SPARQL, as a **RANK** clause as in SQL or an equivalent language construct is missing in the SPARQL language.

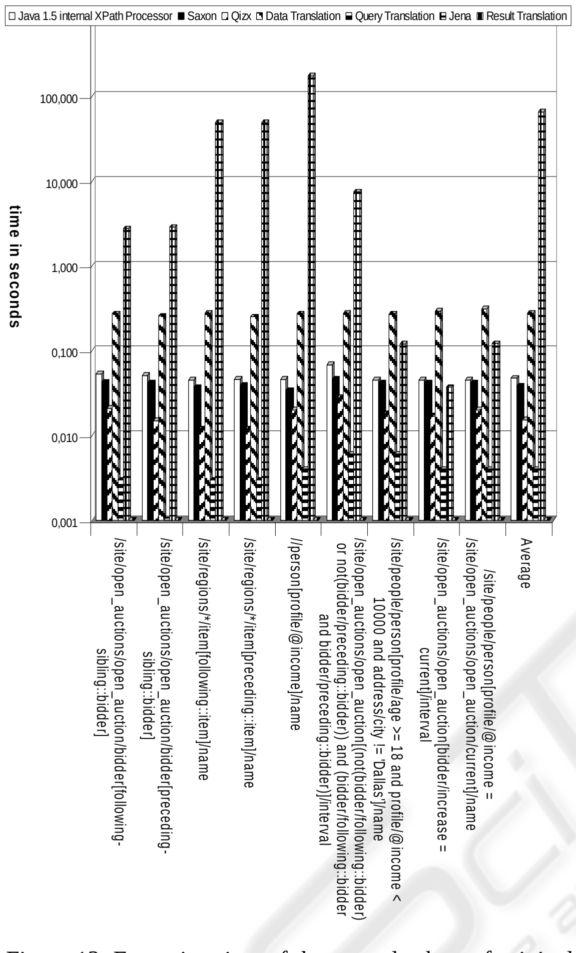

Figure 13: Execution time of the second subset of original XPath queries of the XPathMark Benchmark, of the data translation, of the query translation, of the translated SPARQL query and of the result translation when using an input XML document of size 56.55 Kilobytes.

Figure 12 and Figure 13 present the execution times of the original XPath queries of the XPathMark benchmark, of the data translation, of the query translation, of the translated SPARQL query and of the result translation when using an input XML document of size 56.55 Kilobytes. Furthermore, Figure 13 presents the average execution times of all these 18 queries of the XPathMark benchmark (most right column). Figure 14 presents the average execution times when varying the size of the input file. Furthermore, it shows that the average execution times for processing the translated SPARQL queries is dominated by the execution time of those translated queries, which are translated from XPath queries containing a recursive axis like descendant, descendant-or-self, ancestor, ancestor-or-self, following, preceding, following-sibling and preceding-sibling. The translated SPARQL queries of XPath queries containing a recursive axes have filter expressions like FILTER (xsd:long(?v6) > xsd:long(?v4)), the processing of which is time consuming. Future versions of Jena or other SPARQL engines may optimize these kinds of filter expressions, such that the translations for recursive axes are faster processed.

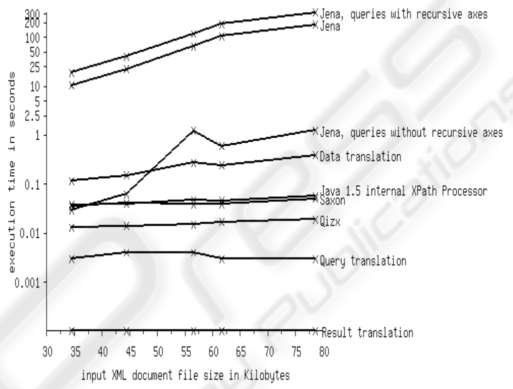

Figure 14: Average execution times of the original queries of the XPathMark benchmark, of data translation, query translation, result translation and the execution times of the translated SPARQL queries (all, only those containing recursive axes and those, which contain only non-recursive queries) using Jena.

6 CONCLUSIONS

In this paper, we first compare the RDF and XPath and XQuery data model and the XPath and SPARQL query languages. Then we propose translations from XML into RDF, from XPath into SPARQL, and from the result of the translated SPARQL query into the XPath and XQuery data model, in order to integrate XML data into RDF data and embed XPath subqueries into SPARQL queries. A translation from XML into RDF and an embedding from XPath into SPARQL enable SPARQL query evaluators to deal with XML data and to process XPath queries as subqueries.

We have developed a prototype to verify our translations and to show the practical usability of such a source-to-source translator. We have done a performance analysis to measure the execution times of the translations and the evaluations of the XPath query and the translated SPARQL query. The evaluation of translated SPARQL queries from XPath queries not containing a recursive axis is efficient and not significantly slower than processing the original XPath queries. The translated SPARQL queries of XPath queries containing a recursive axes contain filter expressions like FILTER (xsd:long(?v6) > xsd:long(?v4)), the processing of which is time consuming. Future versions of Jena or other SPARQL engines may optimize these kinds of filter expressions, such that the translations for recursive axes are faster processed.

REFERENCES

Alexaki, S., Christophides, V., Karvounarakis, G., Plexousakis, D., and Tolle, K.. 2001, The rdfsuite: Managing voluminous rdf description bases. *SemWeb'01 in conjunction with WWW*, Hongkong.

Axyana software, 2006, Qizx/open version 1.1, http://www.axyana.com/qizxopen.

Beckett, D., 2002. The design and implementation of the Redland RDF application framework. *Computer Networks*, 39(5):577-588.

Bettentrupp, R., Groppe, S., Groppe, J., Böttcher, S., and Gruenwald, L., 2006. A Prototype for Translating XSLT into XQuery, *ICEIS 2006*, Paphos, Cyprus.

Broekstra, J., Kampman, A., van Harmelen, 2002. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. *ISWC*, Sardinia.

Cardoso, J., 2007, The Semantic Web Vision: Where are We?, *IEEE Intelligent Systems*, pp.22-26.

Carroll J. J., Klyne G., 2004, Resource Description Framework: Concepts and Abstract Syntax, *W3C Recommendation*, 10th February 2004.

Chong E. I., Das S., Eadon G., Srinivasan J., 2005, An Efficient SQL-based RDF Querying Scheme, *VLDB*, Trondheim, Norway.

Dokulil, J., 2006, Evaluation of SPARQL Queries Using Relational Databases. *ISWC*, Athens, GA, U.S.A..

Droop, M., Flarer, M., Groppe, J., Groppe, S., Linnemann, V., Pinggera, J., Santner, F., Schier, M., Schöpf, F., Staffler, H., and Zugal, S., 2007, Translating XPath Queries into SPARQL Queries, *ODBASE 2007*, Vilamoura, Algarve, Portugal.

Florescu, D., Kossmann, D., 1999, Storing and Querying XML Data Using an RDBMS. *IEEE Data Engineering Bulletin* 22 (1999) 27–34

Fokoue, A., Rose, K., Siméon, J., and Villard, L., 2005, Compiling XSLT 2.0 into XQuery 1.0, *WWW 2005*, Chiba, Japan.

Franceschet, M., 2005, XPathMark – An XPath Benchmark for the XMark Generated Data. *XSym 2005*, Trondheim, Norway.

Franz Inc., 2006. AllegroGraph 64-bit RDFStore, http://www.franz.com/products/allegrograph.

Georgiadis, H., and Vassalos, V., 2006, Improving the Efficiency of XPath Execution on Relational Systems, *EDBT*, Vol. 3896, pp. 570-587, Springer.

Groppe, S., Groppe, J. Linnemann, V., Kukulenz, D., Höller, N., and Reinke, C., 2008, Embedding SPARQL into XQuery / XSLT, ACM SAC 2008, Fortaleza, Ceara, Brasilien

Grust, T., van Keulen, M., and Teubner, J., 2004, Accelerating XPath evaluation in any RDBMS, *ACM Trans. Database Syst*, Vol. 29, pp. 91-131.

Guha, R., 2006. rdfDB: An RDF Database, http://www.guha.com/rdfdb.

Harris, S., and Shadbolt, N., 2005, SPARQL Query Processing with Conventional Relational Database Systems. *WISE Workshops 2005*.

Hewlett-Packard Labs, 2003, The Jena SemanticWeb Toolkit. *Technical report*, Hewlett-Packard Labs, http://jena.sourceforge.net/.

Tatarinov, I., Viglas, S., Beyer, K. S., Shanmugasundaram, J., Shekita, E. J., Zhang, C., 2002, Storing and querying ordered XML using a relational database system. *SIGMOD Conference 2002*, Madison, Wisconsin, U.S.A..

Kay, M. H., 2006, Saxon - The XSLT and XQuery Processor, http://saxon.sourceforge.net.

Klein, N., Groppe, S., Böttcher, S., and Gruenwald, L. , 2005, A Prototype for Translating XQuery Expressions into XSLT Stylesheets, *ADBIS*, Talinn.

de Laborda, C. P., Conrad, S., 2006, Bringing Relational Data into the SemanticWeb using SPARQL and Relational.OWL. *SWDB'06 in conjunction with ICDE 2006*, Atlanta, Georgia, U.S.A..

Lechner, S., Preuner, G., and Schrefl, M., 2001, Translating XQuery into XSLT, In *ER 2001 Workshops*, Yokohama, Japan.

Manolescu, I., Florescu, D., Kossmann, D., 2001, Pushing XML Queries inside Relational Databases. INRIA, *Rapport de recherche* 4112 (2001).

Northrop Grumman Corporation, 2006. Kowari, http://www.kowari.org.

Prud'hommeaux E., Seaborne A., 2008, SPARQL Query Language for RDF, *W3C Recommendation*.

Shanmugasundaram, J., Tufte, K., Zhang, C., He, G., DeWitt, D.J., Naughton, J.F., 1999, Relational Databases for Querying XML Documents: Limitations and Opportunities. *VLDB 1999*, Edinburgh, Scotland.

Subramanyam, G. V., and Kumar, P. S., 2005, Efficient Handling of Sibling Axis in XPath, *COMAD 2005*, Goa, India.

Fan, W., Yu, J. X., Lu, H., Lu, J., and Rastogi, R., 2005, Query Translation from XPath to SQL in the presence of recursive DTDs, *VLDB*, Trondheim, Norway.

Wilkinson, K., Sayers, C., Kuno, H. A., Reynolds, D., 2003. Efficient RDF Storage and Retrieval in Jena2. *SWDB'03 co-located with VLDB 2003*, Berlin.

W3C, 2001, XML Schema Part 2: Datatypes, *W3C Recommendation*, 2001.

W3C, 2007, XPath Version 2.0, *W3C Recommendation*.

Yoshikawa, M., Amagasa, T., Shimura, T., and Uemura, S., 2001, XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases. *ACM TOIT*, 1 (2001) 110–141.