RDF Databases for Querying XML. A Model-mapping Approach

Rubén Tous and Jaime Delgado

Universitat Pompeu Fabra (UPF), Dpt. de Tecnologia, Pg. Circumval.lació, 8. E-08003 Barcelona, Spain,
{ruben.tous, jaime.delgado}@upf.edu

Abstract. Some recent research works face the challenge to map XML documents to RDF triples. Ontologies are used to establish semantic connections among XML applications, and some mechanisms have been defined to query them with natural XML query languages like XPath and XML Query. Generally *structure-mapping* approaches define a simple translation between trivial XPath expressions and a target RDF query language; however some XPath constructs (those depending on node order) cannot be covered in a *structure-mapping* strategy. In contrast, our work takes the *model-mapping* approach, respectful with the node order, that allows a complete mapping of any XPath axis to a single RDQL query. The obtained XPath processor can be fed with an unlimited set of XML schemas and/or RDFS/OWL ontologies. The queries are resolved taking in consideration the structural and semantic connections descrived in the schemas and ontologies. Such behaviour, schema-awareness and semantic integration, can be useful for exploiting schema and ontology hierarchies in XPath queries.

1 Introduction

1.1 Motivation

Translating XML documents to RDF allows to take profit from the powerful descriptive tools of Description Logics (materialised in RDFS and OWL constructs) allowing XML documents interoperate at the semantic level [3]. This allows e.g. to declaratively defining the semantic connections between two different XML schemas or to make XPath queries truly schema-aware. A key element of this approach is providing some mechanisms that allow to keep using natural XML query languages (XPath or XQuery) over the RDF representation obtained. We improve other approaches demonstrating that an XPath processor, respectful with the node order, can be implemented on top of RDF. The resulting processor has some interesting properties, not present in conventional implementations, like schema-awareness and IDREF-awareness. Of course schema-awareness could also be obtained from existing XPath processors by rewriting the XPath queries with all the possible names from the inheritance hierarchy. For queries involving two or more names we should cover all the possible combinations, and all of that should be performed dinamically if we want to consider future changes on the schemas. Such solution is clearly inefficient, complex to implement, difficult to maintain and lets to the client applications the task of interact with the schemas and ontologies and their specific APIs.

1.2 Related work. Model-mapping vs. Structure-mapping

In 2001, [17] defined the terms *structure-mapping* and *model-mapping* to differentiate between works that map the structure of some XML schema to a set of relational tables ([2] [15]) and works that map the XML model to a general relational schema ([17] [18] [4]) respectively. The first ones simply translate XML Schemas or DTDs into a set of database relations (generally one for each complex-type). However soon some researchers realised that having relations with a structure so coupled with the XML structure was not very flexible, because changes in the XML schema propagate to the database schema. Furthermore, some information of the original XML data, the node order, was lost during the translation (the data structure of XML documents is modeled by an ordered tree).

To overcome the limitations of *structure-mapping* strategies [17] [18] [4] suggested to map the constructs of the XML Information Set [6] into a fixed database schema. This schema provided structures to store the different types of nodes of XML according to [6] (document, element, attribute, text, namespace, etc.) and their position in the tree. These approaches, classified as *model-mapping* strategies according to [17], are schema-less, that means they're independent of DTDs or XML schemas and their changes.

An analogous situation can be found today in the RDF field. Some works have appeared focusing on the mapping between XML and RDF [3], but most part of them (all those we have revised) take the *structure-mapping* approach according to the terminology previously mentioned. However, despite most part of these works claim to achieve a simple mapping between XPath and some RDF query language, the reality is that all XPath axis cannot be mapped using this strategy, because of the ordered nature of XML trees differs from the graph nature of RDF. Some XPath axis like *following-sibling* (suspiciously omitted in all these works) cannot be translated with the *structure-mapping* approach.

[9] takes a *structure-mapping* approach and defines a direct way to map XML documents to RDF triples ([3] classifies this approach as *Direct Translation*). [5], [3], and [1] take also a *structure-mapping* approach but focusing on defining semantic mappings between different XML schemas ([3] classifies their own approach as *High-level Mediator*). They also describe some simple mapping mechanisms to cover just a subset of XPath constructs. Other authors like [10] or [11] take a slightly different strategy (though within the *structure-mapping* trend) and focus on integrating XML and RDF to incorporate to XML the inferencing rules of RDF (strategies classified by [3] as *Encoding Semantics*). Finally it's worth mention the RPath initiative [14], that tries to define an analogous language to XPath but for natural (not derived from XML) RDF data (this last work doesn't pursue interoperability between models or schemas).

2 An OWL ontology for the XML model (XML/RDF Syntax)

Our approach takes a strategy more similar to the *model-mapping* approach. We tried to represent the XML Infoset [6] using RDFS and OWL axioms. This allows us to represent any XML document without any restriction and without losing information about node-order. A simplified description of the ontology in Description Logics syntax ($\mathcal{SHIQ}$-like style [21]) would be:

$$
\begin{aligned}
Document &\sqsubseteq Node \\
Element &\sqsubseteq Node \\
TextNode &\sqsubseteq Node \\
childOf &\sqsubseteq descendant \\
parentOf &\sqsubseteq ancestor \\
childOf &= parentOf^{-} \\
&\quad \mathcal{T}rans(ancestor) \\
ancestor &\sqsubseteq ancestorOrSelf \\
self &\sqsubseteq descendantOrSelf \\
self &\sqsubseteq ancestorOrSelf \\
self &= sameAs \\
immediatePrecedingSibling &\sqsubseteq precedingSiblinng \\
immediateFollowingSibling &\sqsubseteq followingSibling \\
immediatePrecedingSibling &= immediateFollowingSibling^{-} \\
&\quad \mathcal{T}rans(followingSibling)
\end{aligned}
$$

Fig. 2 shows graphically how the example of fig. 1 will be represented using the classes and properties defined with OWL.

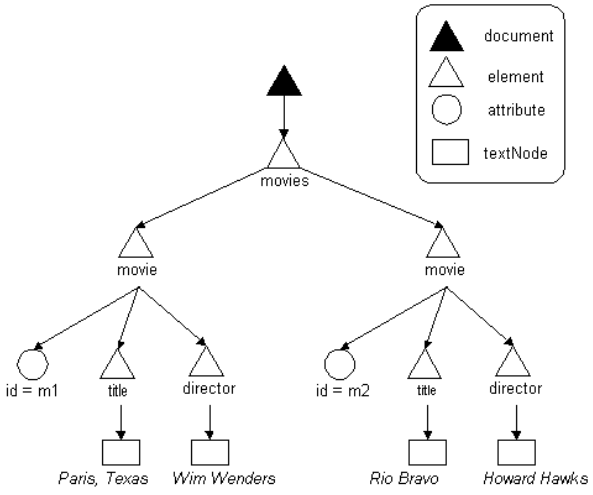

Fig. 1. XML simple example describing two movies

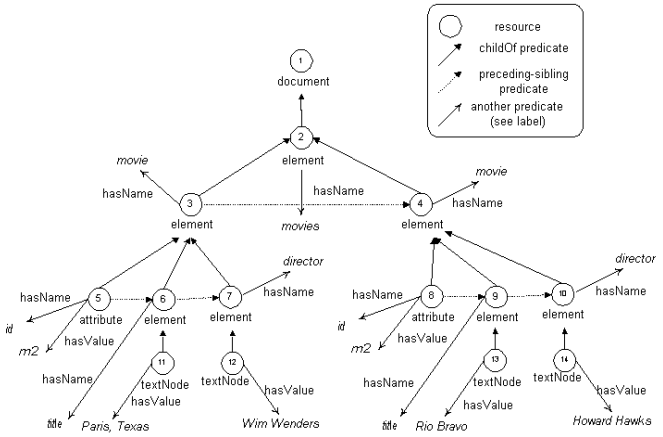

Fig. 2. RDF graph for movies example

3 An inference-based XPath processor

3.1 Overview

Figure 3 outlines how the processor works. The key issue is the XML-to-RDF mapping, already present in other works, but that we face from the *model-mapping* approach. We use the same approach with XSD, obtaining an RDF representation of the schemas. Incorporating alternative OWL or RDFS ontologies is straightforward, because they

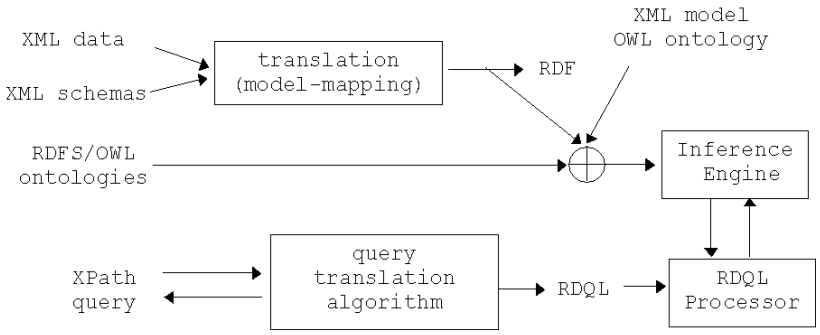

Fig. 3. Semantic XPath processor architecture overview

are already compatible with the inference engine. In the figure we can see also that an OWL representation of the XML model is necessary. This ontology allows the inference engine to correctly process the different XPath axis and understand how the XML elements relate to the different XSD constructs.

3.2 XPath Formal semantics

XPath is a language for addressing parts of an XML document. The language can be formally defined by describing the operations on this data model. It is not a coincidence that some of the axioms are already present in the XML/RDF ontology, because they map directly to XML primitives (e.g. *child*).

First we must define the function E, corresponding to the *XPathExpr* rule from the EBNF grammar [16].

$$E : Path \rightarrow Node \rightarrow sequence(Node)$$

$$\begin{aligned} E[[e_1/e_2]]_x &= \{x_2 \mid x_1 \in E[[e1]]_x \wedge x_2 \in E[[e2]]_{x1}\} \\ E[[a :: t]]_x &= \{x_1 \mid x_1 \in A_a(x) \wedge T_t(x_1)\} \\ E[[e[p]]]_x &= \{x_1 \mid x_1 \in E[[e]]_x \wedge P[[p]]_{x_1}\} \end{aligned}$$

The function A_a describes both the *ForwardAxis* and the *ReverseAxis* rules from the grammar.

$$A_a :\rightarrow Node \rightarrow sequence(Node)$$

$$\begin{aligned} A_{child}(x) &= \{x_1 \mid childOf(x_1, x)\} \\ A_{descendant}(x) &= \{x_1 \mid childOf(x_1, x) \vee \\ &\quad (childOf(x_2, x) \\ &\quad \wedge x_1 \in A_{descendant}(x_2))\} \\ A_{descendant-or-self}(x) &= \{x\} \cup \{x_1 \mid x_1 \in A_{descendant}(x)\} \\ A_{parent}(x) &= \{x_1 \mid childOf(x, x_1)\} \\ A_{ancestor}(x) &= \{x_1 \mid childOf(x, x_1) \vee \\ &\quad (childOf(x, x_2) \\ &\quad \wedge x_1 \in A_{ancestor}(x_2))\} \end{aligned}$$

$$\begin{aligned} A_{ancestor-or-self}(x) &= \{x\} \cup \{x_1 \mid x_1 \in A_{ancestor}(x)\} \\ A_{preceding-sibling}(x) &= \{x_1 \mid precedingSibling(x_1, x)\} \\ A_{preceding}(x) &= \{x_1 \mid x_1 \in A_{descendant-or-self}(x_2) \\ &\quad \wedge x_2 \in A_{preceding-sibling}(x_3)\} \\ &\quad \wedge x_3 \in A_{ancestor-or-self}(x)\} \\ A_{following-sibling}(x) &= \{x_1 \mid precedingSibling(x, x_1)\} \\ A_{following}(x) &= \{x_1 \mid x_1 \in A_{descendant-or-self}(x_2) \\ &\quad \wedge x_2 \in A_{following-sibling}(x_3)\} \\ &\quad \wedge x_3 \in A_{ancestor-or-self}(x)\} \\ A_{attribute}(x) &= \{x_1 \mid attributeOf(x_1, x)\} \\ A_{attribute}(x) &= \{x_1 \mid namespaceOf(x_1, x)\} \end{aligned}$$

The function T describes the *NodeTest* rule from the grammar.

$$T : NodeTest \rightarrow Node \rightarrow Boolean$$

$$\begin{aligned} T_*(x) &= \{true\} \\ T_n(x) &= \{hasName(x, n)\} \\ T_{node()}(x) &= \{type(x,' node')\} \\ T_{text()}(x) &= \{type(x,' textNode')\} \\ T_{element()}(x) &= \{type(x,' elementNode')\} \end{aligned}$$

The function P describes the *Predicates* rule from the grammar. There are a lot of different predicates but defining all is out of the scope of this document. As an example we define here the predicate that expresses the existence of a specific sub-tree as a condition.

$$P : Predicate \rightarrow Node \rightarrow Boolean$$

$$P[[p]]_x = \{\exists x_1 \in E[[p]]_x\}$$

3.3 XPath translation to RDQL

RDQL [13] is the popular RDF query language from HP Labs Bristol. The specification of RDQL was submitted to the W3C in 9 January 2004. Each XPath *axis* can be mapped into one or more triple patterns of the target RDQL query. Analogously each *nodetest* and *predicate* can be mapped also with just one ore more triple patterns. The output RDQL query always takes the form:

```
SELECT *
WHERE
  (?v1, <rdf:type>, <xmloverrdf:document>)
  [triple pattern 2]
  [triple pattern 3]
  ...
  [triple pattern N]
USING
  xmloverrdf  FOR <http://dmag.upf.edu/xml#>
```

The translation can be deduced from the XPath formal semantics. For example, the *following* axis is described as:

$$\begin{aligned} A_{following}(x) = \{x_1 \mid\ & x_1 \in A_{descendant-or-self}(x_2) \\ & \wedge\, x_2 \in A_{following-sibling}(x_3)\} \\ & \wedge\, x_3 \in A_{ancestor-or-self}(x)\} \end{aligned}$$

So the *following* axis must be translated to:

```
(?vi, <xmloverrdf:ancestor-or-self>, ?vi-1)
 i = i + 1
(?vi, <xmloverrdf:following-sibling>, ?vi-1)
i = i + 1
(?vi, <xmloverrdf:descendant-or-self>, ?vi-1)
i = i + 1
```

There are also simple conversion rules for all *nodeTests* and *predicates* but we omit them to save space. The notation used includes variable names like *vi* and *vi-1* where *i* begins with value *2* (because of the first triple pattern is always the same as shown before). So if we would have just the expression:

```
/child::movies/child::movie
```

We will translate the first child axis to:

```
(?v2, <xmloverrdf:childOf>, ?v1)
```

The first node test to:

```
(?v2, <xmloverrdf:hasName>,
          <http://dmag.upf.edu/xmlrdf/names#movies>)
```

The second child axis to:

```
(?result, <xmloverrdf:childOf>, ?v2)
```

And the second node test to:

```
(?result, <xmloverrdf:hasName>,
           <http://dmag.upf.edu/xmlrdf/names#movie>)
```

The complete WHERE clause will appear as:

```
WHERE
  (?v1, <rdf:type>, <xmloverrdf:document>)
  ,(?v2, <xmloverrdf:childOf>, ?v1)
  ,(?v2, <xmloverrdf:hasName>,
            <http://dmag.upf.edu/xmlrdf/names#movies>)
  ,(?result, <xmloverrdf:childOf>, ?v2)
  ,(?result, <xmloverrdf:hasName>,
             <http://dmag.upf.edu/xmlrdf/names#movie>)
```

3.4 Example results

An example query could be:

```
/child::movies/child::movie/child::title
(in abbreviated form /movies/movie/title)
```

That is translated to:

```
SELECT *
WHERE
        (?v1, <rdf:type>, <xmloverrdf:document>)
        , (?v2, <xmloverrdf:childOf>, ?v1)
        , (?v2, <xmloverrdf:hasName>,
            <http://dmag.upf.edu/xmlrdf/names#movies>)
        , (?v3, <xmloverrdf:childOf>, ?v2)
        , (?v3, <xmloverrdf:hasName>,
             <http://dmag.upf.edu/xmlrdf/names#movie>)
        , (?result, <xmloverrdf:childOf>, ?v3)
        , (?result, <xmloverrdf:hasName>,
             <http://dmag.upf.edu/xmlrdf/names#title>)

Result: 6, 9 (node numbers, see figure)
```

4 Incorporating schema-awareness

4.1 Mapping XML Schema to RDF

Having an XML instance represented with RDF triples opens a lot of possibilities. As we have seen before, we can use OWL constructs (*subPropertyOf*, *transitiveProperty*, *sameAs*, *inverseOf*, etc.) to define the relationship between the different properties defined in the ontology. In our ontology for the XML model, the object of the *hasName* property is not a literal but a resource (an RDF resource). This key aspect allows applying to *hasName* all the potential of the OWL relationships (e.g. defining ontologies with names relationships). So, if we want our XPath processor to be schema-aware, we just need to translate the XML Schema language to RDF, and to add to our XML/RDF Syntax ontology the necessary OWL constructs that allow the inference engine to understand the semantics of the different XML Schema components. The added axioms in Description Logics syntax ($\mathcal{SHIQ}$-like style [21]) would be:

$$
\begin{aligned}
hasName &\sqsubseteq fromSubstitutionGroup \\
&\quad\ \mathcal{T}rans(fromSubstitutionGroup) \\
hasName &\sqsubseteq fromType \\
&\quad\ \mathcal{T}rans(fromType) \\
fromType &\sqsubseteq subTypeOf
\end{aligned}
$$

4.2 A simple example of schema-aware XPath processing

The next example illustrates the behavior of our processor in a schema-related XPath query. Take this simple XML document:

```
<A>
  <B id='B1' />
  <B id='B2'>
```

```
    <C id='C1'>
     <D id='D1'></D>
    </C>
   </B>
   <B id='B3'/>
 </A>
```

And its attached schema:

```
<schema>
 <complexType name='BType'>
   <complexContent>
      <extension base='SUPERBType'></extension>
   </complexContent>
 </complexType>
 <element name='B'
          type='BType' substitutionGroup='SUPERB' />
</schema>
```

When evaluating the XPath query *//SUPERB*, our processor will return the elements with IDs 'B1', 'B2' and 'B3'. These elements have a name with value 'B', and the schema specifies that this name belong to the substitution group 'SUPERB', so they match the query. Also, when evaluating the query *//SUPERBType*, the processor will return 'B1', 'B2' and 'B3'. It assumes that the query is asking for elements from the type SUPERBType or one of its subtypes.

5 Implementation and performance

The work has been materialised in the form of a Java API. We have used the Jena 2 API [8] for RDQL computation and OWL reasoning. To process XPath expressions we have modified and recompiled the Jaxen XPath Processor [7]. An on-line demo can be found at *http://dmag.upf.edu/contorsion*.

Though performance wasn't the target of the work, it is an important aspect of the processor. We have realised a performance test over a Java Virtual Machine v1.4.1 in a 2GHz Intel Pentium processor with 256Mb of memory. The final delay depends mainly on two variables, the size of the target documents, and the complexity of the query. Table 1 shows the delay of the inferencing stage for different document depth levels and also for some different queries.

The processor behaves good with medium-size documents and also with large ones when simple queries are used (queries that not involve transitive axis), but when document size grows the delay related to the complex queries increases exponentially. Some performance limitations of the Jena's OWL inference engine have been described in [22]. We are now working on this problem, trying to obtain a more scalable inference engine. However, the current processor's performance is still acceptable for medium-size XML documents.

Table 1. Performance for different document depth levels

expression	5d	10d	15d	20d
/A/B	32ms	47ms	47ms	62ms
/A/B/following-sibling::B	125ms	46ms	48ms	47ms
/A/B/following::B	125ms	62ms	63ms	47ms
/A//B	172ms	203ms	250ms	219ms
//A//B	178ms	266ms	281ms	422ms

6 Uses of the obtained XPath processor

6.1 XML Schema for metadata interoperability

The object-oriented nature of some XML Schema constructs allows using them to increase the interoperability of applications or to fix interoperability problems in an elegant way. For example, the *substitutionGroup* inheritance mechanism can be used to bind the names of two different XML languages. Take for example a simple schema for describing movies records. The schema defines the elements *movies, movie, title, year, country, runtime*, etc. It could be interesting in some context to have the possibility to write the element and attribute names in a language different from English. The next XML fragment is an instance of the previous schema but using Spanish instead of English for element names.

```
<pelculas>
  <pelicula id='m1'>
   <titulo>Blade Runner</titulo>
   <estreno>1982</estreno>
   <pais>USA</pais>
   <duracion>117</duracion>
</peliculas>
```

We can generate a schema that binds the different names from the Spanish version to the (master) English version.

```
<schema>
<element name='peliculas' substitutionGroup='movies'>
 <xs:complexType>
  <xs:sequence>
   <element name='pelicula' substitutionGroup='movie'>
    <complexType>
     <sequence>
      <element name='titulo' substitutionGroup='title'/>
      <element name='estreno'substitutionGroup='year'/>
      <element name='pais'substitutionGroup='country'/>
      <element name='duracion'substitutionGroup='runtime'/>
     </sequence>
     <attribute name='id'/>
    </complexType>
```

```
   </element>
  </sequence>
 </complexType>
</element>
</schema>
```

Now, using our schema-aware XPath processor, if we ask for */movie/country* we will obtain the same as for the */pelicula/pais*. So, we can develop applications that are not tied to a particular schema but to an abstract one. Applying this idea in a scenario with *a posteriori* interoperability needs could not be always possible, because these situations use to be better faced with procedural tools like XSLT. However the analysis of the possibilities of this strategy is still work-in-progress, because OWL offers enough expressivity to define complex declarative mappings among schemas.

6.2 Application to model-mapping

Executing XPath queries over non-XML data Our approach allows executing XPath queries over a RDF representation of data from some data model for which a mapping with XML has been defined. There is no need of a transcoding between the XPath query and the new data model, because the primitives of the two models (XML and for example the Relational Model) have been already mapped and the inference engine can resolve the query.

Example: XPath over relational data For a better understanding of the concept let's see a simple example. Take a relational version of the movies XML document as illustrated in the table 2.

Table 2. Relational version of the movies example

Title	Director
Paris, Texas	Wim Wenders
Rio Bravo	Howard Hawks

We have developed a test OWL ontology for the Relational Model. Fig. 4 shows a graphical view of an instance representing the movies example (we omit the ontology and the RDF serialization of the example for space reasons). We also have defined the mapping between the two models in a different OWL ontology. Here comes an extract. Now we can execute a XPath query over the RDF serialization of the relational version of the movies example. For example the XPath query:

```
/child::node()/descendant::title
```

Will be translated to the query:

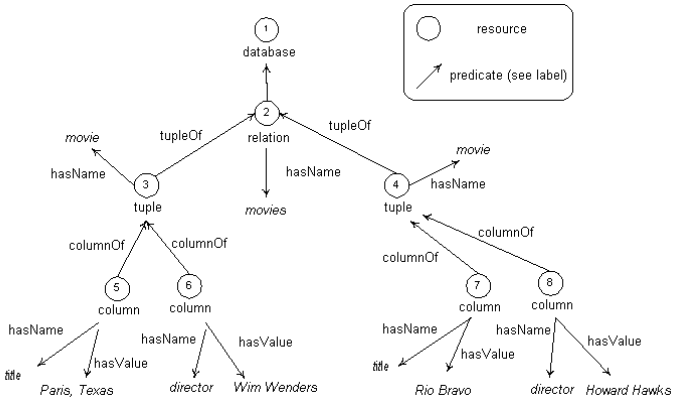

Fig. 4. RDF representation of the relational version of the movies example

```
SELECT *
WHERE
        (?v1, <rdf:type>, <xmloverrdf:document>)
        , (?v2, <xmloverrdf:childOf>, ?v1)
        , (?result, <xmloverrdf:descendant>, ?v2)
        , (?result, <xmloverrdf:hasName>, "title")
USING
        xmloverrdf  FOR <http://dmag.upf.edu/xml#>

Result: 5, 7 (node numbers, see figure)
```

The query expresses restrictions with XML constructs, but these constructs are mapped by the inference engine to the equivalent relational operations. Inversely, if we would describe another query language like for e.g. SQL with the appropriate RDQL operations (to operate over the RDF representation of the relational model), we could execute SQL queries directly over XML.

XPath over reified RDF data If we want to apply the strategy to execute XPath expressions over natural RDF data (not derived from XML), we must provide an RDF/RDF Syntax (although it might seem rather redundant) because we need to express the RDF primitives in OWL to be able to define an OWL mapping between them and the XML/RDF Syntax. However, this is exactly what RDF reification does. Reification is the ability to treat an RDF statement as a resource, and hence to make assertions about that statement. RDF represents a reified statement as four statements with particular RDF properties and objects: the statement (S, P, O), reified by resource R, is represented by:

```
R rdf:type rdf:Statement
R rdf:subject S
R rdf:predicate P
R rdf:object O
```

Fig. 5 shows a natural RDF version of the movies example. If we reify all the statements of the model we obtain the quadlets of Fig. 6 (the {x, rdf:type, Statement} have been excluded from the figure).

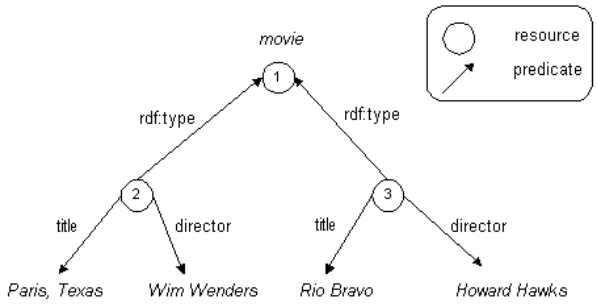

Fig. 5. Natural-RDF representation of the movies example

We have partially mapped the reified RDF model (we could call it RDF/RDF Syntax) to our XML/RDF Syntax and obtained promising results with simple XPath queries. Now if we translate the following XPath query to RDQL:

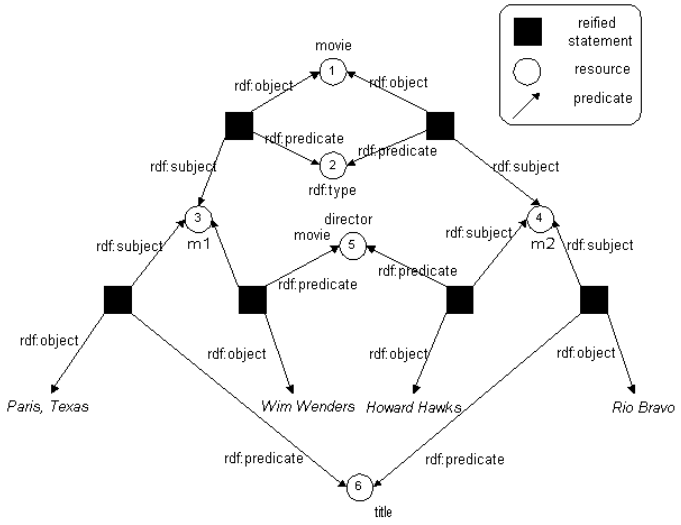

Fig. 6. Reified statements from the natural-RDF representation of the movies example

```
/child::movie/child::title
```

We will obtain:

```
SELECT *
WHERE
  (?v1, <xmloverrdf:hasName>,
                <http://dmag.upf.edu/eg#movie>),
  (?v2, <xmloverrdf:childOf>, ?v1),
  (?v2, <xmloverrdf:hasName>,
                <http://dmag.upf.edu/eg#title>),
  (?v3, <xmloverrdf:childOf>, ?v2),
  (?v3, <xmloverrdf:hasValue>, ?result)

Result: "Paris, Texas", "Rio Bravo"
```

The XML/RDF-RDF/RDF mapping is still work in progress, and the discussion around it overcomes the scope and the physical limits of this paper.

7 Conclusions

In this paper we have proposed the *model-mapping* approach for the XML-to-RDF mapping, contributing to the recent research trend that suggests the mapping of XML documents to triples allowing XML documents interoperate at the semantic level [3]. We expose why an RDF representation based on the *model-mapping approach* allows to retain the node order, in contrast with the usual *structure-mapping* approach. We also have outlined the algorithm used internally to translate the XPath expressions to RDQL queries. The work has been materialised in the form of a Java API, an on-line demo can be found at *http://dmag.upf.edu/contorsion*.

As we have shown the processor can be used to express schema-aware queries, to face interoperability among different XML languages or to integrate XML with RDF sources. We have also evaluate the possibility to define a mapping between another data model and XML, in such a way that RDQL queries obtained from XPath expressions over non-XML data can be resolved by inference, without an explicit mapping between XPath and the external model. We have shown examples for the Relational Model and also for RDF (with the help of reification).

References

1. B.Amann,C.Beeri,I.Fundulaki,and M.Scholl.Ontology-Based Integration of XML Web Resources. In Proceedings of the 1st International Semantic Web Conference (ISWC 2002),pages 117-131,2002.
2. Christophides, V., Abiteboul, S., Cluet, S., and Scholl, M. 1994. From structured documents to novel query facilities. SIGMOD Rec. 23, 2 (June), 313-324.
3. Cruz, I., Xiao H., Hsu F. An Ontology-based Framework for XML Semantic Integration. University of Illinois at Chicago. Eighth International Database Engineering and Applications Symposium. IDEAS'04. July 7-9, 2004 Coimbra, Portugal.
4. Florescu,D.AND Kossmann, D. 1999. Storing and querying XML data using an RDMBS. IEEE Data Eng. Tech. Bull. 22, 3, 27-34.

5. A. Y. Halevy, Z. G. Ives, P. Mork, I. Tatarinov: Piazza: Data Management Infrastructure for Semantic Web Applications, 12th International World Wide Web Conference, 2003
6. XML Information Set (Second Edition) W3C Recommendation 4 February 2004 http://www.w3.org/TR/xml-infoset/
7. Jaxen: Universal Java XPath Engine http://jaxen.org/
8. Jena 2 - A Semantic Web Framework http://www.hpl.hp.com/semweb/jena.htm
9. M.C.A.Klein. Interpreting XML Documents via an RDF Schema Ontology.In Proceedings of the 13th International Workshop on Database and Expert Systems Applications (DEXA 2002),pages 889-894, 2002.
10. L.V.Lakshmanan and F.Sadri.Interoperability on XML Data.In Proceedings of the 2nd International Semantic Web Conference (ICSW 03),2003.
11. P.F.Patel-Schneider and J.Simeon.The Yin/Yang web:XML syntax and RDF semantics.In Proceedings of the 11th International World Wide Web Conference (WWW2002),pages 443-453,2002.
12. RDF/XML Syntax Specification (Revised) W3C Recommendation 10 February 2004 http://www.w3.org/TR/rdf-syntax-grammar/
13. RDQL - A Query Language for RDF W3C Member Submission 9 January 2004 http://www.w3.org/Submission/RDQL/
14. RPath - RDF query language proposal http://web.sfc.keio.ac.jp/ km/rpath-eng/rpath.html
15. Shanmugasundaram, J., Tufte, K., He, G., Zhang, C., Dewitt,D.J.,and Naughton,J.F. 1999. Relational databases for querying XML documents: Limitations and opportunities. In Proceedings of the 25th International Conference on Very Large Data Bases (VLDB, Edin-burgh, Scotland, Sept. 7-10). Morgan Kaufmann, San Mateo, CA, 302-314.
16. XML Path Language (XPath) 2.0 W3C Working Draft 23 July 2004 http://www.w3.org/TR/xpath20/
17. M. Yoshikawa, T. Amagasa, T. Shimura and S. Uemura, XRel: A Path-Based Approach to Storage and Retrieval of XML Documents using Relational Databases, ACM Transactions on Internet Technology, Vol. 1, No. 1, June 2001.
18. Zhang, J. 1995. Application of OODB and SGML techniques in text database: an electronic dictionary system. SIGMOD Rec. 24, 1 (Mar.), 3-8.
19. Dave Reynolds. Jena 2 Inference support http://jena.sourceforge.net/inference/
20. OWL Web Ontology Language Overview. W3C Recommendation 10 February 2004 http://www.w3.org/TR/owl-features/
21. Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In Harald Ganzinger, David McAllester, and Andrei Voronkov, editors, Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99), number 1705 in Lecture Notes in Artificial Intelligence, pages 161-180. Springer, 1999.
22. Dave Reynolds. Jena 2 Inference support http://jena.sourceforge.net/inference/