Contorsion. A Semantic XPath Processor

Rubén Tous [1]

Dpt. de Tecnologia
Univesitat Pompeu Fabra (UPF)
Barcelona, Spain

Jaime Delgado [2]

Dpt. de Tecnologia
Univesitat Pompeu Fabra (UPF)
Barcelona, Spain

Abstract

This work describes the architecture of Contorsion, a semantic XPath processor that acts over an RDF mapping of XML. It contributes to a recent research trend that defines an XML-to-RDF mapping allowing XML documents interoperate at the semantic level. We use a *model-mapping approach* to represent instances of XML and XML Schema in RDF. This representation retains the node order, in contrast with the usual *structure-mapping* approach. The 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.

Key words: data interoperability, semantic integration, XML, XPath, RDF, ontologies, OWL

1 Introduction

1.1 Motivation

Generally, applications where XML documents are involved define or reuse one or more XML schemas. These schemas are mainly used for instance validity check. However sometimes is necessary to consider the inheritance hierarchies

[1] Email: ruben.tous@upf.edu
[2] Email: jaime.delgado@upf.edu

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

defined in the schemas for other purposes, e.g. when evaluating queries or conditions that can refer to concepts not directly present in the data, but related to them through an inheritance chain. Today it is also becoming common the use of RDFS/OWL ontologies to define semantic connections among application concepts. All this *structural* and *semantic* knowlege is hard to access for developers, because require a specific treatment, like defining multiple extra queries for the schemas, or using complex RDF tools to access the ontologies information.

To overcome this situation here we present the architecture of a schema-aware and ontology-aware XPath processor that acts over an RDF mapping of XML. 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 [2]. This allows e.g. to declaratively defining the semantic connections between two different XML schemas or to make XPath queries truly schema-aware. The Contorsion 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. With the existing XPath processors to achieve the same result we should rewrite 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.

We use a *model-mapping approach* to represent instances of XML and XML Schema in RDF. This representation retains the node order, in contrast with the usual *structure-mapping* approach, so it allows a complete mapping of all XPath axis. XML model defines an ordered tree, and some XPath constructs like *following sibling* depend on the ordered nature of XML instances. The interest on the node-order can vary depending of the application, but an XML-to-RDF mapping cannot be considered complete without this feature.

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

The origins of this work can be found in a research trend that tries to exploit the advantages of an XML-to-RDF mapping [1][2][3][4][5][6][7]. However, the concepts of *structure-mapping* and *model-mapping* are older. In 2001, [8] defined these terms to differentiate between works that map the structure of some XML schema to a set of relational tables and works that map the XML model to a general relational schema respectively.

More recently, [4] takes a *structure-mapping* approach and defines a direct way to map XML documents to RDF triples ([2] classifies this approach as

Direct Translation). [1], [2], and [3] take also a *structure-mapping* approach but focusing on defining semantic mappings between different XML schemas ([2] 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 [5] or [6] 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 [2] as *Encoding Semantics*). Finally it's worth mention the RPath initiative [7], 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 Architecture of the semantic XPath processor

2.1 Overview

Figure 1 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. In contrast with the *structure-mapping* approach, that maps the specific structure of some XML schema to RDF constructs, we map the XML Infoset [9] using RDFS and OWL axioms. This allows us to represent any XML document without any restriction and without losing information about node-order. We use the same approach with XSD, obtaining an RDF representation of the schemas. Incorporating alternative OWL

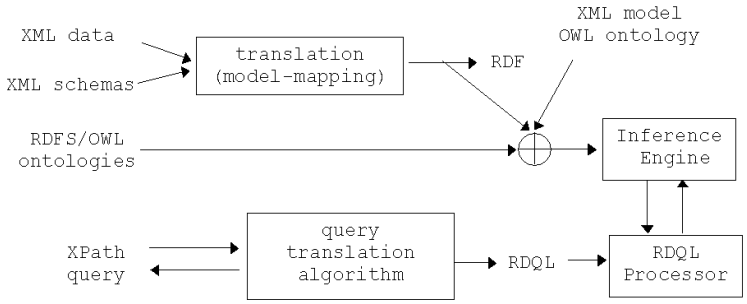

Fig. 1. Semantic XPath processor architecture overview

or RDFS ontologies is straightforward, because they 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.

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

We tried to represent the XML Infoset [9] using RDFS and OWL axioms. A simplified description of the ontology in Desctiption Logics syntax ($\mathcal{SHIQ}$-like

style [15]) 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. 3 shows graphically how the example of fig. 2 will be represented using the classes and properties defined with OWL.

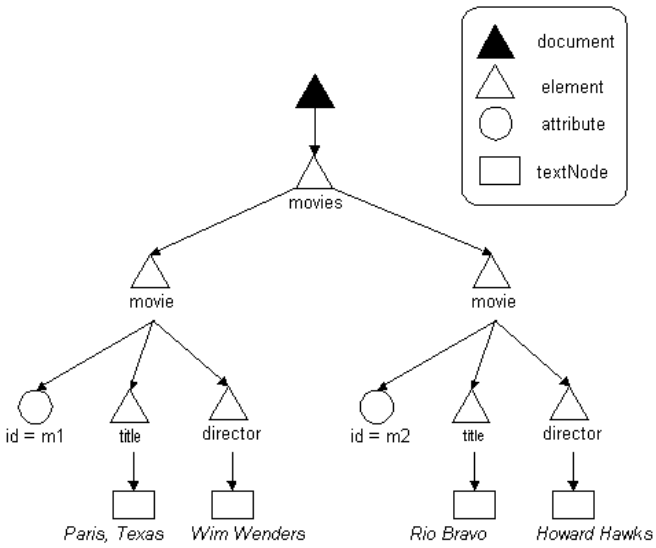

Fig. 2. XML simple example describing two movies

2.3 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

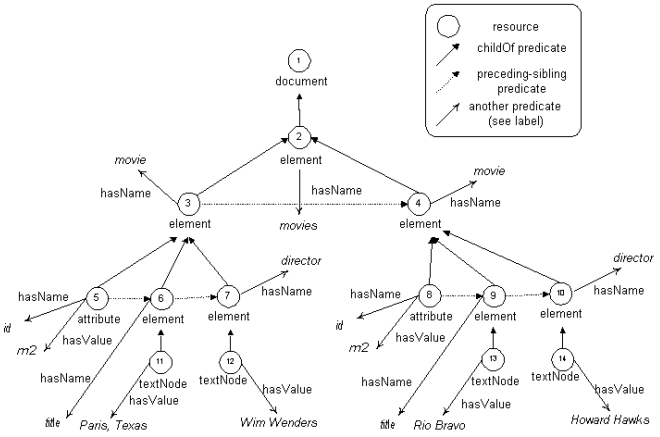

Fig. 3. RDF graph for movies example

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 [10].

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

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

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

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

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

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

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.

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

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

2.4 XPath translation to RDQL

Each XPath *axis* can be mapped into one or more triple patterns of the target RDQL [13] 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:

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

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>)
```

2.5 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>, "movies")
        , (?v3, <xmloverrdf:childOf>, ?v2)
        , (?v3, <xmloverrdf:hasName>, "movie")
        , (?result, <xmloverrdf:childOf>, ?v3)
        , (?result, <xmloverrdf:hasName>, "title")

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

3 Incorporating schema-awareness

3.1 Mapping XML Schema to RDF

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 to apply to *hasName* all the potential of the OWL relationaships (e.g. defining ontologies whith 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 Desctiption Logics syntax ($\mathcal{SHIQ}$-like style [15]) would be:

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

3.2 A simple example of schema-aware XPath processing

The next example ilustrates the behaviour 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.

4 Implementation and performance

The work has been materialised in the form of a Java API. We have used the Jena 2 API [12] for RDQL computation and OWL reasoning. To process XPath expressions we have modified and recompiled the Jaxen XPath Processor [11]. 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 [14]. We are now working on this prob-

lem, 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	20d (Xalan XPath processor)
/A/B	32ms	47ms	47ms	62ms	16ms
/A/B/following-sibling::B	125ms	46ms	48ms	47ms	15ms
/A/B/following::B	125ms	62ms	63ms	47ms	16ms
/A//B	172ms	203ms	250ms	219ms	31ms
//A//B	178ms	266ms	281ms	422ms	32ms

5 Uses of the obtained XPath processor

5.1 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.

5.2 Application to model-mapping

5.2.1 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.

5.2.2 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 seralization of the relational version of the movies example. For example the XPath query:

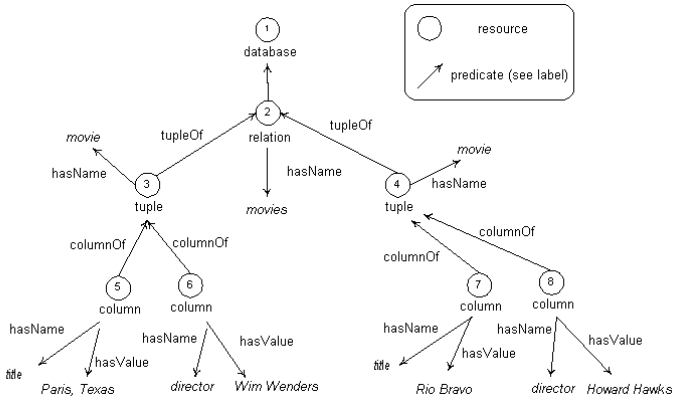

Fig. 4. RDF representation of the relational version of the movies example

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

Will be translated to the query:

```
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.

5.2.3 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 a 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). We have partially mapped

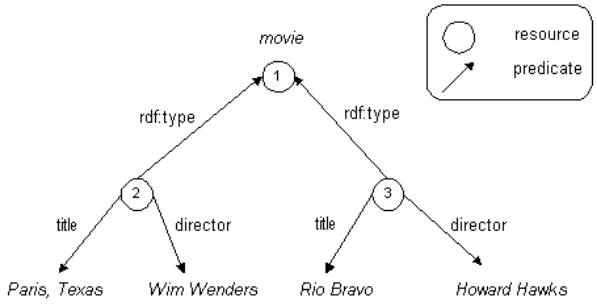

Fig. 5. Natural-RDF representation of the movies example

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

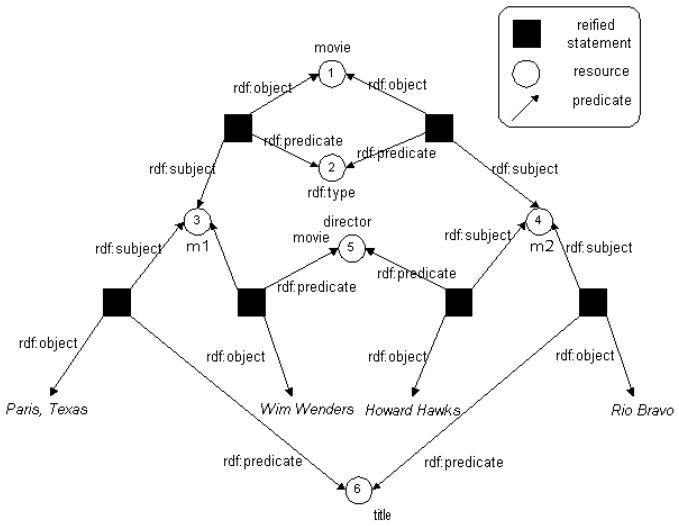

Fig. 6. Reified statements from the natural-RDF representation of the movies example

translate the following XPath query to RDQL:

```
/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.

6 Conclusions

In this paper we have described the architecture of Contorsion, an innovative semantic XPath processor that acts over an RDF mapping of XML. The processor resolve the queries taking in consideration the structural and semantic connections described in the schemas and ontologies provided by the user. 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 outlited 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] 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

[2] 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.

[3] 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.

[4] 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.

[5] L.V.Lakshmanan and F.Sadri.*Interoperability on XML Data.*In Proceedings of the 2nd International Semantic Web Conference (ICSW 03),2003.

[6] 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.

[7] *RPath - RDF query language proposal* http://web.sfc.keio.ac.jp/ km/rpath-eng/rpath.html

[8] 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.

[9] *XML Information Set (Second Edition)* W3C Recommendation 4 February 2004 http://www.w3.org/TR/xml-infoset/

[10] XML Path Language (XPath) 2.0 W3C Working Draft 23 July 2004 http://www.w3.org/TR/xpath20/

[11] *Jaxen: Universal Java XPath Engine* http://jaxen.org/

[12] *Jena 2. A Semantic Web Framework* http://www.hpl.hp.com/semweb/jena.htm

[13] *RDQL - A Query Language for RDF* W3C Member Submission 9 January 2004 http://www.w3.org/Submission/RDQL/

[14] Dave Reynolds. *Jena 2 Inference support* http://jena.sourceforge.net/inference/

[15] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. *Practical reasoning for expressive description logics*. Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning. Springer, 1999.