Architecture of a Semantic XPath Processor. Application to Digital Rights Management

Rubén Tous, Roberto García, Eva Rodríguez and Jaime Delgado

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

Abstract. This work describes a novel strategy for designing an XPath processor that acts over an RDF mapping of XML. 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 described in the schemas and ontologies. Such behavior, schema-awareness and semantic integration, can be useful for exploiting schema and ontology hierarchies in XPath queries. We test our approach in the Digital Rights Management (DRM) domain. We explore how the processor can be used in the two main rights expression languages (REL), MPEG-21 REL and ODRL.

1 Introduction

1.1 Motivation

Usually XML-based applications use one or more XML schemas. These schemas are mainly used for instance validity check. However, it is sometimes necessary to consider the inheritance hierarchies 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* knowledge is hard to access for developers, because it requires 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 we present the architecture of a schema-aware and ontology-aware XPath processor. 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 described in the schemas and ontologies. 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.

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).

The target to achieve a semantic behavior for XPath/XQuery has also been faced in [23]. This approach consists also in translating the XML schemas to OWL, but the authors define an XQuery variant for the OWL data model called SWQL (Semantic Web Query Language). The difference between this approach and ours is that our work does not need a translation between the semantic queries (instances of SWQL in the related approach) and XPath/XQuery expressions. We have developed a new XPath processor that manipulates conventional queries but taking in consideration the semantic relationships defined in the schemas and/or ontologies.

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 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.

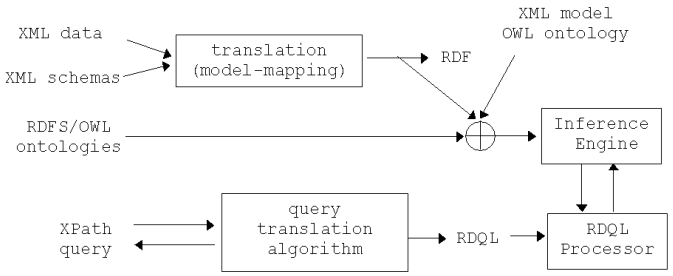

Fig. 1. Semantic XPath processor architecture overview

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

We have tried to represent the XML Infoset [9] using RDFS and OWL axioms. A simplified description of the ontology in Description Logics syntax ($\mathcal{SHIQ}$-like style [17]) would be:

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

2.3 XPath to RDQL translation algorithm

RDQL [13] is the popular RDF query language from HP Labs Bristol. 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 or 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]
```

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-2, <xmloverrdf:ancestor-or-self>, ?vi-3)
(?vi-1, <xmloverrdf:following-sibling>, ?vi-2)
(?vi, <xmloverrdf:descendant-or-self>, ?vi-1)
```

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

3 Incorporating schema-awareness

3.1 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 [17]) 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 [11] for RDQL computation and OWL reasoning. To process XPath expressions we have modified and recompiled the Jaxen XPath Processor [10]. 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 well 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 [18]. 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

5 Testing in the DRM Application Domain

The amount of digital content delivery in the Internet has made Web-scale Digital Rights Management (DRM) a key issue. Traditionally, DRM Systems (DRMS) have dealt with this problem for bounded domains. However, when scaled to the Web, DRMSs are very difficult to develop and maintain. The solution is interoperability of DRMS, i.e. a common framework for understanding with a shared language and vocabulary. That is why it is not a coincidence that organisations like MPEG (Moving Picture Experts Group), OMA (Open Mobile

Alliance), OASIS (Organization for the Advancement of Structured Information Standards), TV-Anytime Forum, OeBF (Open eBook Forum) or PRISM (Publishing Requirements for Industrial Standard Metadata) are all involved in standardisation or adoption of rights expression languages (REL). Two of the main REL initiatives are MPEG-21 REL [22] and ODRL [20].

Both are XML sublanguages defined by XML Schemas. The XML Schemas define the language syntax and a basic vocabulary. These RELs are then supplemented with what are called Rights Data Dictionaries [21]. They provide the complete vocabulary and a lightweight formalisation of the vocabulary terms semantics as XML Schemas or ad hoc ontologies. ODRL and MPEG-21 REL have just been defined and are available for their implementation in DRMS. They seem quite complete and generic enough to cope with such a complex domain. However, the problem is that they have such a rich structure that they are very difficult to implement. They are rich in the context of XML languages and the "traditional" XML tools like DOM or XPath. There are too many attributes, elements and complexTypes (see Table 2) to deal with.

Table 2. Number of named XML Schema primitives in ODRL and MPEG-21 REL

	Schemas	**xsd:attribute**	**xsd:complexType**	**xsd:element**	**Total**
ODRL	EX-11	10	15	23	127
	DD-11	3	2	74	
MPEG-21	EL-R	9	56	78	330
	REL-SX	3	35	84	
	REL-MX	1	28	36	

5.1 Application to ODRL license processing

Consider looking for all constraints in a right expression, usually a rights license, that apply to how we can access the licensed content. This would require so many XPath queries as there are different ways to express constraints. For instance, ODRL defines 23 constraints: industry, interval, memory, network, printer, purpose, quality... This amounts to lots of source code, difficult to develop and maintain because it is very sensible to minor changes to the REL specs. Hopefully there is a workaround hidden in the language definitions.

As we have said, there is the language syntax but also some semantics. The *substitutionGroup* relations among *elements* and the *extension/restriction base* ones among *complexTypes* encode generalisation hierarchies that carry some lightweight, taxonomy-like, semantics. For instance, all constraints in ODRL are defined as XML elements substituting the *o-ex:constraintElement*, see Figure 2. The difficulty is that although this information is provided by the XML Schemas, it remains hidden when working with instance documents of this XML Schemas. However, using the semantics-enabled XPath processor we can profit from all this information. As it has been shown, the XML Schemas are translated to OWL ontologies that make the generalisation hierarchies explicit, using

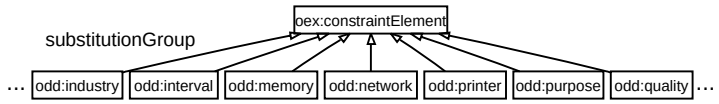

Fig. 2. Some ODRL constraint elements defined as *substitutionGroup* of *constraintElement*

subClassOf and *subPropertyOf* relations. The ontology can be used then to carry out the inferences that allow a semantic XPath like "//o-ex:constraintElement" to retrieve all *o-ex:constraintElement* plus all elements defined as its *substitutionGroup*.

5.2 Application to the MPEG-21 authorization model

In MPEG-21 an authorisation algorithm that is a decision making process resolving a central question *"Is a Principal authorized to exercise a Right such a Resource?"* is defined. In this case, the semantic XPath processor help us when determining if the user has the appropriate rights taking into account the rights lineage defined in the RDD (Rights Data Dictionary).

In contrast with ODRL, that uses XMLSchemas both for the language and dictionary definitions, MPEG-21 has an ontology as dictionary (RDD). The semantics that it provides can also be integrated in our semantic XPath processor. To do that, the MPEG-21 RDD ontology is translated [19] to the ontology language used by the Semantic XPath Processor, i.e. OWL. Once this is done, this ontology is connected to the semantic formalisation build up from the MPEG-21 REL XML Schemas. Consequently, semantic XPath queries can also profit from the ad hoc ontology semantics. For instance, the acts taxonomy in MPEG-21 RDD, see Figure 3, can be seamlessly integrated in order to facilitate license checking implementation. Consider the scenario: we want to check if our set of licenses authorises us to uninstall a licensed program. If we use

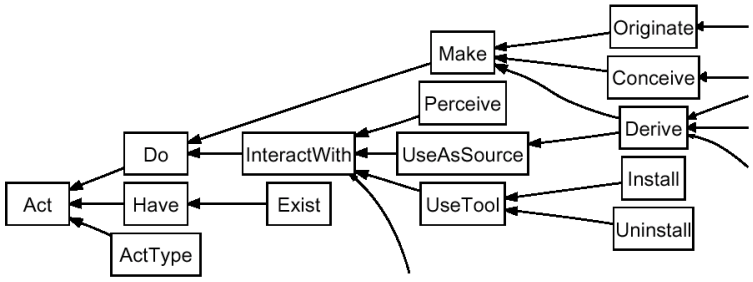

Fig. 3. Portion of the acts taxonomy in MPEG-21 RDD

XPath, there must be a path to look for licenses that grant the *uninstall* act, e.g. "//r:license/r:grant/mx:uninstall". Moreover, as it is shown in the taxonomy, the *usetool* act is a generalisation of the *uninstall* act. Therefore, we must also check for licenses that grant us *usetool*, e.g "//r:license/r:grant/mx:uninstall". An successively, we should check for *interactwith*, *do* and *act*.

However, if we use a semantic XPath, the existence of a license that grants any of the acts that generalise *uninstall* implies that the license also states that the *uninstall* act is also granted. This is so because, by inference, the presence of the fact that relates the license to the granted act implies all the facts that relate the license to all the acts that specialise this act. Therefore, it would suffice to check the semantic XPath expression "//r:license/r:grant/mx:uninstall". If any of the more general acts is granted it would match. For instance, the XML tree */r:license/r:grant/dd:usetool* implies the trees */r:license/r:grant/dd:install* and */r:license/r:grant/dd:uninstall*.

6 Conclusions and future work

In this paper we have described a novel strategy for designing a semantic XPath processor that acts over an RDF mapping of XML. 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 obtained processor resolves the queries taking in consideration the structural and semantic connections described in the schemas and ontologies provided by the user. It can be used to express schema-aware queries, to face interoperability among different XML languages or to integrate XML with RDF sources.

In the context of DRM implementation, the Semantic XPath Processor has shown its benefits. First of all, less coding is needed. The Semantic XPath processor allows reusing the semantics hidden in the XML Schemas so we do not need to recode them. Moreover, the code is more independent from the underlying specifications. If there is a change in the specifications, which causes a modification of the XML Schemas, it is only necessary to regenerate the corresponding ontologies. Now we are working to embed the processor in an XQuery implementation to achieve the semantic behaviour also for XQuery expressions.

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. Jaxen: Universal Java XPath Engine http://jaxen.org/
11. Jena 2 - A Semantic Web Framework http://www.hpl.hp.com/semweb/jena.htm
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. XML Path Language (XPath) 2.0 W3C Working Draft 23 July 2004 http://www.w3.org/TR/xpath20/
15. Dave Reynolds. Jena 2 Inference support http://jena.sourceforge.net/inference/
16. OWL Web Ontology Language Overview. W3C Recommendation 10 February 2004 http://www.w3.org/TR/owl-features/
17. 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 (LPAR99), number 1705 in Lecture Notes in Artificial Intelligence, pages 161180. Springer, 1999.
18. Dave Reynolds. Jena 2 Inference support http://jena.sourceforge.net/inference/
19. J. Delgado and I. Gallego and R. Garca. Use of Semantic Tools for a Digital Rights Dictionary. E-Commerce and Web Technologies: 5th International Conference, 2004. K. Bauknecht, K. and M. Bichler and B. Prll. LNCS Volume 3182 (338-347) Springer-Verlag
20. R. Iannella. Open Digital Rights Language (ODRL), Version 1.1. World Wide Web Consortium 2002 (W3C Note). http://www.w3.org/TR/odrl.
21. G. Rust and C. Barlas. The MPEG-21 Rights Data Dictionary. IEEE Transactions on Multimedia, 2005 volume 7 number 2.
22. X. Wang and T. DeMartini and B. Wragg and M. Paramasivam. The MPEG-21 Rights Expression Language. IEEE Transactions on Multimedia 2005 volume 7 number 2
23. Lehti and Fankhauser (2004). XML Data Integration with OWL: Experiences & Challenges, SAINT 2004: 160-170.