Krextor – An Extensible XML→RDF Extraction Framework

Christoph Lange

Computer Science, Jacobs University Bremen, ch.lange@jacobs-university.de

Abstract. The semantics of an XML-based language can be specified by mapping an XML schema to an ontology, thus enabling the wide range of XML applications to contribute to the Semantic Web. The Krextor XML→RDF extraction framework proposes a practical solution to this problem, novel in its extensibility. We present its architecture and show how it can easily be extended to support additional input and output languages.

1 Introduction: Semantic Markup and RDF

In the well-known and hotly debated layer cake architecture of the Semantic Web (see [9] for a survey of its different incarnations), XML has always been placed below RDF. This must not be misunderstood in a way that XML should only be used for encoding higher-layer formalisms like RDF or OWL in a standardized way (e. g. RDF/XML [5]). Other XML-based languages can also be given a semantics, and by that, we are not just considering XML's inherent semantics of a tree structure, identifiers, and cross-references. Semantic XML languages are widely used for domain-specific and general-purpose applications. The key difference of an XML document compared to an RDF graph is its *sequential order* – suited for the way humans *read*. Style languages like CSS and XSL further help presenting XML to non-technical readers. Schema languages allow for defining the syntax of new XML languages for any domain. A well-engineered XML schema[1] can lead to a much more concise and intuitive knowledge representation than an RDF graph, as it does not force authors to break all statements down to triples; compare the direct XML serialisation of OWL [18] to its RDF/XML serialisation. Given adequate editors, both authoring XML and RDF can be made easy for domain experts – but our own experience in the domain of mathematics shows that many more domain experts are familiar with domain-specific markup languages than with Semantic Web standards.

The semantics of XML languages is usually defined in a human-readable specification and hard-coded into applications. Therefore, XML has often been blamed for not being sufficiently semantic. We argue against that and bridge the semantic gap by improving the extraction of RDF from XML. XML languages

[1] The lowercase term "schema" is used in the general sense here, not referring to the particular schema language XML Schema.

having a well-defined *formal semantics* in terms of RDF already exist. Obvious examples are XML representations for RDF and ontologies themselves. Then, there is RDFa [1] for embedding RDF into the widely-supported but non-semantic XHTML. Less formal alternatives, such as microformats, can also be considered semantic if an RDF semantics has been specified for them and there is a way of obtaining RDF from such documents, e. g. by a GRDDL link from the document to the implementation of a translator to RDF. For data-centric XML languages, e. g. XML representations of relational databases, it is also straightforward to specify an RDF semantics. Finally, there are *semantic markup languages* – XML languages with a formal semantics that have explicitly been designed for knowledge representation. Consider, for example, OMDoc (Open Mathematical Documents), which is developed in our group [12]. While the formal core of OMDoc (symbol declarations, modular theories, proof objects) has a model- and proof-theoretic semantics that is much more expressive than Semantic Web ontologies (see [20]), we have specified an OWL-DL semantics for large parts of OMDoc's semi-formal structural aspects (document structure, mathematical statements, structured proofs; see [13]).

A widened connection between the XML and RDF layers of the layer cake has most notably been suggested by Patel-Schneider and Siméon, who developed a unified model theory for both [19]. However, the benefit of that approach is rather theoretical, as it makes impractically restrictive assumptions about the XML structure (see [17] for details). Moreover, XML and RDF have evolved in parallel for years and gained a wider tool support each. Therefore, we take the more practical approach of extracting RDF from XML on the level of syntax and thus giving XML languages an RDF semantics by providing 1. rules that translate XML to RDF and, if needed, 2. an ontology providing the vocabulary for the extracted RDF.

2 The Krextor XML→RDF Extraction Framework

The Krextor XML→RDF extraction framework originated from the need to manage OMDoc documents in a Semantic Web application [14,13]. Having modeled an OWL-DL ontology for OMDoc, an OMDoc→RDF extraction was needed, which we hard-coded in XSLT from scratch, after an older, hard-coded Java implementation had proven to be too unflexible to maintain. The RDF was output in the RXR notation (Regular XML RDF [4]), from which it was parsed by a Java library. Later, the same was required for OpenMath content dictionaries, a language similar to OMDoc. This led to the decision to create a generic XSLT-based framework (cf. fig. 1) that allows developers to define translations ("extraction modules") from any XML language to RDF more easily than in pure XSLT, as will be shown in the following.

A generic module provides convenience templates and functions for defining extraction rules in a way that abstracts from the concrete output format and instead defining the semantics of XML structures on a high level, in terms of resources and properties. Krextor's generic "representation" of XML is a transient one; the generic module is just a step in the pipeline, grouping extracted data into triples and forwarding them to the selected output module. Supported output formats, besides RXR, are: RDF/XML [5], the text notation Turtle [6], and, thanks to the Saxon XSLT processor [11], a direct interface to Java, for a more efficient integration into applications. In RDF/XML and Turtle output, the triples are grouped by common subjects and predicates. This is achieved by first obtaining RXR and then transforming it to the notation desired using XSLT grouping – a compromise between efficiency and a clean separation of concerns. Syntactic sugar, offered by some RDF notations, has only partly been implemented. At the moment, there is no support for author-defined namespace prefixes, "anonymous" blank nodes (bnodes) without identifiers, and RDF containers or collections in the output. Semantically, that does not make a difference. After all, our target "audience" are not humans, who usually do not want to read raw RDF, but applications that further process the RDF and conveniently prepare it for users – as, e. g., the semantic wiki SWiM does [13]. Nevertheless, some syntactic sugar remains on our agenda, as it facilitates testing Krextor during development.

Krextor is available as a collection of XSLT style sheets, with an optional Java wrapper for direct integration into applications. For scripting and debugging, there is a shell script frontend, reading XML from the standard input and writing RDF in the desired notation to the standard output.

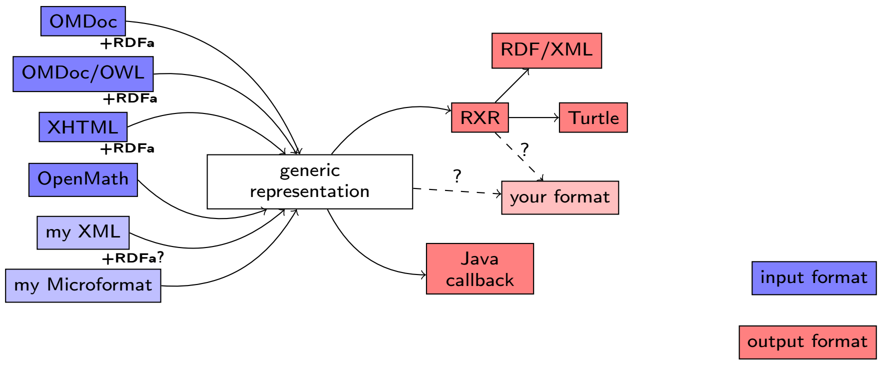

Fig. 1. Krextor's extraction process and modules

Besides the input formats mentioned so far, Krextor also supports RDFa – embedded in XHTML or other host languages, such as Open Document[2], and in the following section, we will show how it can be extended to microformats. Moreover, we are working on a translation from OMDoc to OWL, implemented as a Krextor extraction module, which allows for authoring Semantic Web ontologies with integrated documentation and in a more modular way (see [16] for details).

[2] See http://rdfa.info/2008/03/13/rdfa-support-coming-in-odf-12/ and stay tuned for Open Document 1.2 ☺

Thus, Krextor can even be used as a bridge from the XML layer into the ontology layer of the Semantic Web cake. To add an input format, one has to provide XSLT templates that map XML structures of the input to calls of Krextor's generic templates, as shown in the following section. We follow the paradigm of making easy things easy and hard things possible – an inexpensive claim, actually, but considerable efforts have been made to implement convenience templates and functions for common extraction tasks, which are easier to use than plain XSLT. There are predefined templates for creating a resource that is instance of some class and for adding literal- or URI-valued properties to the current resource. Several ways of generating resource URIs are provided, including fragment URIs of the form "document's URI" # "fragment's xml:id", but an extraction module can also implement its own URI generator(s). (The latter has been done for OMDoc, which uses a *document*/*theory*/*symbol* URI pattern [16].) The information that the developer has to provide explicitly is kept at a minimum level: When no subject URI for a resource is given, Krextor auto-generates one using the desired generation function. When no object value for a property is given, it is taken from the currently processed XML attribute or element. As an alternative for very simple formats, where XML elements directly map to ontology classes and properties, a declarative mapping can be given as annotated literal XML. For complex input formats like the above-mentioned OMDoc, the full computational power of XSLT can be used, at the expense of readability. A new output module merely has to implement one template for low-level RDF generation, accepting the parameters subject (URI or bnode ID), subject type (URI or bnode), predicate (URI), object, object type (URI, bnode ID, or literal), language, and datatype. More complex output modules can be realized by post-processing output from existing output modules.

3 Use Cases and Applications

One application area of Krextor is the semantic wiki SWiM [13]. Mathematical documents can be imported and edited in their original formats, which allows for building on existing tool support. An RDF outline is only extracted from them after storing them in the database; the RDF plus the background knowledge from the ontologies then powers semantic services – currently navigation, querying, and problem-solving assistance [13,15]. OMDoc particularly needs complex extraction rules: Its mathematical symbols have their own URI schema, and it can mix formal and informal knowledge. The RDF graph extracted from a full-featured OMDoc document consists of two parallel trees, one tree of the mathematical structure, and one of the rhetorical structure, interwoven via an annotation ontology. Despite this complexity, 21 out of the 44 templates in the extraction module for OMDoc have completely been implemented using Krextor's convenience templates only. 15 make use of additional XPath constructs, 5 use additional, more complex XSLT constructs, and 3 use both. OMDoc as a frontend for OWL ontologies, as mentioned above and detailed in [16], will eventually be integrated into SWiM. The extraction of OWL from special OMDoc documents has also been implemented using Krextor. In these documents, ontologies are modeled as mathematical theories, resources are declared as symbols having definitions, axioms, and theorems. Many of these mathematical statements are modeled in a way that is more familiar to people with a logics background: the range and domain of a property is, e. g., represented by a single relation type declared for the property symbol [16]. The OMDoc→OWL module makes considerably more use of XPath and XSLT than the above-mentioned module that obtains the structure of OMDoc documents as RDF, but still it paid off to implement it within Krextor, as part of the required functionality could be shared with the former module.

We exemplify Krextor's extensibility, a major design goal, by an extraction module for a simple language, the hCalendar microformat [7], using the RDF Calendar vocabulary [8]. The extraction rules for an event and its start date are given in listing 2, which is considerably shorter than an equivalent implementation in plain XSLT. The first template matches any element of class "vevent" and creates an instance of the *ical:Vevent* class from it. When a child link annotated as the URI of the event is present, its target is used to identify the event; otherwise, a bnode is created for the event. The second template matches any element of class "dtstart" and adds an *ical:dtstart* property of datatype *xsd:date* to the current resource. Krextor's convenience templates automatically take care of recursing to child elements, keeping track of the current resource, and reading the values of properties if they are given in a reasonable place, such as the text content of an element. Given the following sample input, Turtle output can be

```
<stylesheet version="2.0">
  <!-- we generate resource URIs ourselves -->
  <param name="autogenerate-fragment-uris" select="()"/>
  <template match="*[@class='vevent']">
    <!-- Take URL property if given, otherwise create a bnode -->
    <variable name="subject" select="a[@class='url']/@href"/>
    <call-template name="krextor:create-resource">
      <with-param name="subject" select="$subject"/>
      <with-param name="blank-node" select="not($subject)"/>
      <with-param name="type" select="'&ical;Vevent'"/>
    </call-template></template>
  <template match="*[@class='dtstart']">
    <call-template name="krextor:add-literal-property">
      <with-param name="property" select="'&ical;dtstart'"/>
      <with-param name="datatype" select="'&xsd;date'"/>
    </call-template></template> <!-- ... and so on ... -->
```

Fig. 2. A hCalendar extraction module

obtained e. g. by calling `krextor hcalendar..turtle infile.xhtml` on the command line:

```
<div class="vevent">
 <a class="url" href="http://www.eswc2009.org">ESWC</a>
 starts on <span class="dtstart">2009-05-31</span>.</div>
```

```
<http://www.eswc2009.org>
  a <http://www.w3.org/2002/12/cal/ical#Vevent> ;
  <http://www.w3.org/2002/12/cal/ical#dtstart>
    "2009-05-31"^^<http://www.w3.org/2001/XMLSchema#date> .
```

4 Related Work and Conclusion

Swignition's [10] architecture is very similar to Krextor's. For end-users and web developers, it offers much richer features, including support for many microformats, other legacy ways of embedding RDF into HTML, and GRDDL links. For knowledge engineers or developers who quickly want to define an RDF translation from a new XML language, Krextor performs better, being extensible by additional input formats with much less lines of code than the Swignition Perl library. So far, GRDDL is only "supported" by Krextor in the obvious sense that it facilitates the XSLT-based implementation of an XML→RDF translation that can then be linked to a schema or to documents using GRDDL; automatically choosing the right extraction module by interpreting GRDDL annotations in the input document is not yet supported. Both systems approach integration into semantic applications differently: Swignition comes with a TCP/IP interface, whereas Krextor features a Java API and benefits from the wide support for XSLT. The authors of **XSDL** [17] have done substantial theoretical elaboration on a semantics-preserving translation of XML into RDF and provide a concise declarative syntax mapping XML to OWL-DL. To the best of our knowledge, XSDL has not been implemented, though. As its syntax uses XML and XPath, we consider it feasible to prove the theoretical results the authors have obtained for Krextor as well by rewriting XSDL definitions into equivalent Krextor extraction modules. This would also make XSDL usable as a convenient input language for Krextor, making extraction modules look less like XSLT and XPath. **XSPARQL** [2] mixes SPARQL into XQuery, resulting in a query language that fully breaks the boundaries between XML and RDF. It avoids the necessity of first converting from one representation into the other. However, persistently storing the result of such a conversion is often desired in applications, whereas the current implementation of XSPARQL focuses on one-time queries.

Conclusion: The Krextor framework supports many XML→RDF conversion tasks and can easily be extended by additional input and output formats and integrated into semantic applications. Thereby, we have opened new paths from the XML layer of the Semantic Web architecture to the RDF and higher layers. When designing new ontologies, knowledge engineers can now take the creative challenge of developing a convenient XML syntax for domain-specific knowledge and provide a Krextor extraction module that translates this XML to RDF in terms of these ontologies. We will continue using Krextor for mathematical markup but are also interested in proving its extensibility on other semantic markup languages. A future research direction that we want to explore is adding extraction rules as annotations to XML schema languages like RELAX NG [21], thereby unifying two tasks that belong together but have been separated so far: specifying the syntax and the semantics of an XML language.

Acknowledgments: The author would like to thank Richard Cyganiak for providing him with a Java-based RDFa test suite, and several reviewers for constructive feedback.

References

1. B. Adida, M. Birbeck, S. McCarron, and S. Pemberton. RDFa in XHTML: Syntax and processing. Recommendation, W3C, 2008.
2. W. Akhtar, J. Kopecký, T. Krennwallner, and A. Polleres. XSPARQL: Traveling between the XML and RDF worlds – and avoiding the XSLT pilgrimage. In Bechhofer et al. [3].
3. S. Bechhofer, M. Hauswirth, J. Hoffmann, and M. Koubarakis, editors. *5th European Semantic Web Conference*, volume 5021 of *LNCS*. Springer, 2008.
4. D. Beckett. Modernising semantic web markup. In *XML Europe*, 2004.
5. D. Beckett. RDF/XML syntax specification. Recommendation, W3C, 2004.
6. D. Beckett. Turtle – terse RDF triple language, 2007.
7. T. Çelik. hCalendar. Microformat specification, Technorati, 2008.
8. D. Connolly and L. Miller. RDF calendar. Interest Group Note, W3C, 2005.
9. A. Gerber, A. v. Merwe, and A. Barnard. A functional semantic web architecture. In Bechhofer et al. [3].
10. T. A. Inkster. Swignition. http://buzzword.org.uk/swignition/, 2009.
11. M. Kay. Saxonica: XSLT and XQuery processing. http://www.saxonica.com.
12. M. Kohlhase. OMDoc – *An open markup format for mathematical documents*. Number 4180 in LNAI. Springer, 2006.
13. C. Lange. SWiM – a semantic wiki for math. knowledge. In Bechhofer et al. [3].
14. C. Lange. Krextor. http://kwarc.info/projects/krextor/, 2009.
15. C. Lange, T. Hastrup, and S. Corlosquet. Arguing on issues with mathematical knowledge items in a semantic wiki. In J. Baumeister and M. Atzmüller, editors, *LWA (Lernen, Wissensentdeckung und Adaptivität)*, volume 448, 2008.
16. C. Lange and M. Kohlhase. A mathematical approach to ontology authoring and documentation. In *Mathematical Knowledge Management*, LNAI. Springer, 2009. https://svn.omdoc.org/repos/omdoc/trunk/doc/blue/foaf/mkm09.pdf.
17. S. Liu, J. Mei, A. Yue, and Z. Lin. XSDL: Making XML semantics explicit. In C. Bussler, V. Tannen, and I. Fundulaki, editors, *SWDB*, volume 3372, 2004.
18. B. Motik and P. F. Patel-Schneider. OWL web ontology language: XML serialization. Working draft, W3C, Dec. 2008.
19. P. F. Patel-Schneider and J. Siméon. The Yin/Yang web: A unified model for XML syntax and RDF semantics. *IEEE TKDE*, 15(4), 2003.
20. F. Rabe. *Representing Logics and Logic Translations*. PhD thesis, Jacobs University Bremen, 2008.
21. RELAX NG. http://www.relaxng.org/.