# RDF syntax normalization using XML validation

Denny Vrandečić, Frank Dengler, Sebastian Rudolph, Michael Erdmann[1]

Institut AIFB, Universität Karlsruhe, Germany
{dvr,fde,sru}@aifb.uni-karlsruhe.de
[1]ontoprise GmbH, Karlsruhe, Germany
erdmann@ontoprise.de

**Abstract.** Basing the Semantic Web stack on XML promised to introduce a lot of advantages by reusing tools and expertise. In general, these promises have not been fulfilled. In this paper we describe a method that re-uses existing standards creatively in order to provide a guaranteed serialization of an RDF graph that can again be used easily with widely spread XML tools and techniques like SAX, DOM, or XSLT. We provide a proof-of-concept implementation and provide a use case to illustrate the advantages of our approach. Alternative approaches are discussed and compared.

## 1 Introduction

The Semantic Web was created as a set of standards building on top of each other, and incorporating already existing and well established standards like URIs as identifiers [1] or XML for serialization [2]. The inherent promise was that due to the reuse of these standards, widely deployed tools and expensively built expertise will not be lost but remain relevant and in continued use. This promise has not been fully realized.

The standard serialization for RDF is the XML-based RDF/XML syntax [3], but we will show in Section 2 that most XML oriented tools can only deal superficially with RDF/XML files. Since the RDF graph can be expressed in so many different ways in XML, creating applications using the XML set of tools and expertise is often very expensive and unreasonably hard.

We present an approach that uses well-established standards to sufficiently constrain the serialization of RDF/XML in order to allow them to be used in classic XML processing approaches. We suggest to use the same approach humans would use in order to serialize their conceptualizations in XML, namely following an XML schema. Three popular XML schema languages are currently widely used, the XML-native Document Type Definitions (DTD) [2], the W3C XML Schema Definition language (XSD) [4], and the Regular Language for XML Next Generation (RELAX NG) [5]. Due to its legibility and high availability we choose DTD for our prototype implementation, but the work can be extended to use the other schema languages as well, as will be discussed in Section 7. Note that by restricting RDF/XML with DTDs, the resulting files are still fully compliant RDF files, and therefore can be used by other Semantic Web tools out of the box.

The major theoretical challenges are the details of the interpretation of the XML schema file as a specification for generating the requested serialization XML schemas

can be roughly regarded as formal grammars meant to check a given input XML document for validity. Instead, we use the very same XML schema file in order to query a given RDF graph and then define how to serialize the results. This interpretation forces us to define a number of constraints on the covered XML schemas (i.e., arbitrary DTDs can not be used, cf. Section 3). We offer a dialog-based tool to quickly create RDF/XML-compliant DTDs that contain the data that is requested by the developer, thus allowing the developer to easily adhere to these constraints without deep understanding of Semantic Web standards.

We provide a prototype implementation of the approach, and present an example instantiation of the method in order to demonstrate its usefulness and applicability. For our example we took FOAF files from the web, pipe them through our implementation, and then use standard XML processing techniques (XSLT) in order to generate HTML pages. We provide the example workflow as a web service that can be tested by the reader. The source code is also made available.

The following section gives a more detailed motivation for this work, explaining the expected advantages. Section 3 describes the approach to create the serializations. Section 4 explains how the dialog-driven DTD creation tool works. This is followed by a description of the current prototype implementation and demonstration in Section 5. Section 6 describes related approaches towards the goal we are outlining, before we close with an outlook at future work. As a running example we describe a web service that takes arbitrary FOAF-files [6] and translates them to an HTML presentation.

## 2 Motivation

Today, one of the major hindrances towards achieving the wide deployment of Semantic Web technologies is the lack of sufficient expertise. If at all, Semantic Web technologies have only recently been introduced to students' curricula, and most general audience books on Semantic Web technology have only been published in 2009 [7–9]. On the other hand, XML expertise is widely available. There are numerous books and courses on XML technologies, and many websites provides access to communities of practice, often even within the specific domain of the user.

Also, even though the number of Semantic Web tools is growing rapidly, many of them require a deeper understanding of the technology than is readily available. Strengthening the ties between RDF and XML allows not only to reuse existing expertise, but also to re-enable the already existing tools.

To illustrate the problems with using RDF/XML, consider a model with the following single triple expressed in N3:

```
aifb:Rudi rdf:type foaf:Person .
```

The following three documents are examples that all serialize this single triple in RDF/XML:

```
<rdf:RDF>
 <rdf:Description rdf:about="&aifb;Rudi">
  <rdf:type>
```

```
1 <!ELEMENT rdf:RDF (foaf:Person*)>
2 <!ATTLIST rdf:RDF xmlns:rdf  CDATA #FIXED
3       "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
4 <!ATTLIST rdf:RDF xmlns:foaf CDATA #FIXED "http://xmlns.com/foaf/0.1/">
5 <!ELEMENT foaf:Person (foaf:name foaf:mbox*)>
6 <!ATTLIST foaf:Person rdf:about CDATA #REQUIRED>
7 <!ELEMENT foaf:name (#PCDATA)>
8 <!ELEMENT foaf:mbox EMPTY>
9 <!ATTLIST foaf:mbox rdf:resource CDATA #REQUIRED>
```

**Fig. 1.** A DTD for a fragment of FOAF.

```
    <rdf:Description rdf:about="&foaf;Person"/>
   </rdf:type>
  </rdf:Description>
 </rdf:RDF>
```

An object may be moved to the property element as an attribute value:

```
<rdf:RDF>
 <rdf:Description rdf:about="&aifb;Rudi">
  <rdf:type rdf:Resource="&foaf;Person"/>
 </rdf:Description>
</rdf:RDF>
```

Typing information can be moved to the element:

```
<rdf:RDF>
 <foaf:Person rdf:about="&aifb;Rudi"/>
</rdf:RDF>
```

All documents have very different XML infosets and very different XML serializations, but equal RDF semantics. Thus, creating a simple list of all persons according to the RDF semantics is not trivial without using an RDF parser. An XQuery or an XSLT transformation will be cumbersome to write. In order to simplify the authoring of XML-based solutions, we provide a workflow that will normalize the RDF/XML serialization following a given XML schema.

The example in Figure 1 shows an XML DTD (automatically generated by our approach) that normalizes the way to express FOAF data in XML files. It states that the root element has an arbitrary number of `foaf:Persons` (line 1) and each `foaf:Person` must have a `foaf:name` and may have some mail address (given by `foaf:mbox`, line 5). Lines 2-4 define the namespaces [10] of the resulting document. Note that the namespaces are fixed. This is to circumvent the fact that DTDs per se are not namespace-aware (unlike other XML schema languages like XSD or RelaxNG). Line 6 ensures that every `foaf:Person` instance has a URI and may not be a blank node.

A FOAF-file that validates against the given DTD in Figure 1 is given in Figure 2. The result is a reasonably legible XML file and also a fully valid RDF file, which can be used with all Semantic Web tools.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:foaf="http://xmlns.com/foaf/0.1/">
 <foaf:Person rdf:about=
  "http://www.aifb.uni-karlsruhe.de/Personen/viewPersonOWL/id57instance">
   <foaf:name>Rudi Studer</foaf:name>
   <foaf:mbox rdf:resource="mailto:studer@aifb.uni-karlsruhe.de" />
 </foaf:Person>
</rdf:RDF>
```

**Fig. 2.** FOAF-file after normalization.

One of the advantages of this approach is that we have certain guarantuees with regards to the content of the resulting file. Unlike the cardinality axiom

`SubClassOf(foaf:Person ExactCardinality(1 foaf:name))`

which allows us to infer that every `foaf:Person` has to have a `foaf:name` property (be it given or not), validating against the above DTD will guarantee us that we *know* the actual value of the `foaf:name` property for every `foaf:Person`. No person will lack a name. So for certain properties we do not only know that they exist, but we also know that we know their values. Using autoepistemic logic extending the usual DL syntax [11] we could state that $\texttt{foaf} : \texttt{Person} \sqsubseteq \mathbf{K}\texttt{foaf.name}.\top$, which allows us to write simpler code in the post-processing of the result since a certain data completeness is guarantueed.

Even though our approach does not allow to create arbitrary XML – specifically, it does not allow to create XML documents that are not valid RDF – and thus cannot be used as a one-step solution towards reusing already existing DTDs, we will show that one can create a custom DTD to serialize the RDF file first, and then translate it to any representation the user requires (even in non-XML formats) by using readily available XSLT tools and expertise. Therefore it is possible to ground any RDF file into an already existing file format in two steps, given that all required information is available.

Thus, the approach described in this paper will hopefully weaken the effects of missing tools and expertise in the area of the Semantic Web, decrease costs for implementing Semantic Web solutions, and increase the number of opportunities where Semantic Web technologies can be applied.

## 3 Normalizing the serialization

In order to create the normalized serialization we use the provided DTD to generate a series of SPARQL queries. The results of the queries are then used to write the actual serialization. In this section we describe how this is accomplished.

The given DTD has to fulfill a number of constraints that will be listed explicitly in the following section. First we need to read and define all given namespaces from the root element and declare them appropriately in the SPARQL query. Furthermore we add the RDF and RDFS namespaces, if not already given.

Next we go through every possible child element type of the root element `rdf:RDF`. In our example we can see in line 1 that there is only one possible child element type,

foaf:Person. This is the type of the instances we are looking for, i.e. we translate it into the following query fragment:

```
SELECT ?individual WHERE {  ?individual rdf:type foaf:Person . }
```

Next, for every required child element of the found elements, we add a respective line to the WHERE-clause. In our example, foaf:Person requires only foaf:name (line 5, foaf:mbox is optional). So we add the following line (introducing a new variable every time):

```
?individual foaf:name ?v1 .
```

If the foaf:name itself would have pointed to another element, this element would be the type of ?v1, and in return we would add all required properties for ?v1 by adding the subelements of this type, and so on.

The result of this query is a list of all individuals that are described with all the necessary properties as defined by the DTD. In the example they are, thus, not only instances of foaf:Person but also of $\mathbf{K}$foaf.name.$\top$.

Next we iterate over the list, asking for every required and optional property individually. In the example, we would issue the following two queries:

```
SELECT ?result WHERE { aifb:id57instance foaf:name ?result } LIMIT 1

SELECT ?result WHERE { aifb:id57instance foaf:mbox ?result }
```

Since line 5 of the DTD states that there is only one foaf:name, we limit the first query to 1. Again, if the subelement of a property would have been stated in the DTD, it would have been required to add all required properties for ?result, just as it was done for ?v1 above. Furthermore, each required and optional property of ?result also has to be gathered in the same way as we do for all the instances of the root element's children.

With the results of the queries, we can start generating the serialization. Fig.2 shows such a serialization that was created with the normalization tool.

One final note on the approach: since the resulting queries are all describing conjunctive queries, we can use the queries on a reasoning-aware SPARQL endpoint. This allows us to employ the power of RDFS and OWL to provide for a fully transparent ontology mapping. For example, even if the individuals in the input file are actually having the class swrc:Student, as long as this is a subclass of foaf:Person the results will come back as expected and the normalized serialization will contain all instances of direct and indirect subclasses. This provides a powerful and easy way to quickly add further knowledge bases using new vocabularies to an existing tool chain. It has to be noted that a DTD asking for instances of both classes will get redundancies, though.

## 4 Creation of compliant schemas

The provided DTDs have to fulfill a number of constraints so that they can be used by the normalization tool. This section lists those constraints explicitly. Since it would

require quite some expertise with regards to RDF/XML-serializations (which we explicitly do not expect) to create DTDs fulfilling these constraints, we also offer a dialog-based tool to easily create such DTDs. This tool is described in this section.

An RDF-compliant DTD, that can be used by the normalizer described in the previous section, has to fulfill the following constraints:

- the resulting XML-file must be a valid RDF/XML-file
- all used namespaces have to be fixed in the DTD and defined in the root element of the resulting XML-file
- the root element of the XML-file has to be `rdf:RDF`
- since DTDs are not context-sensitive, the DTD can use each element only a single time

Especially the last constraint is a severe restriction necessary due to the shortcomings of DTDs. In Section 7 we will take a look at possible remedies for this problem using other, more modern XML schema languages.

The first constraint is basically impossible to fulfill without deep knowledge of the RDF/XML serializations. Because of that we have created a tool that analyses a given dataset or ontology and then guides the developer through understandable dialog options to create a conformant DTD. The tool follows the following approach:

1. the tool loads a number of RDF files. It does not matter if these files contain proper ontologies (i.e. definitions of the terms), simple knowledge bases, or a mix of both.
2. the tool offers the developer to select a class from the given RDF files. `rdfs:Class` and `owl:Class` are both considered. The developer has to decide if the result should contain exactly one instance, or an arbitrary number of instances.
3. for the selected class, all sensible properties are offered. Sensible properties are those that either are defined with a domain being the given class, or where the instances of the given class have assertions using the property. For each selected property the developer has to decide if this property is required, can be repeated arbitrary times, or both.
4. for each selected property the developer has to decide on the type of the filler, especially if it is a datatype value or an individual, and if the latter, if it is of a specific class (which again will be selected from a provided list, based both on the range of the property and the classes of the actual fillers in the knowledge base).
5. if a class was selected, enter recursively to Step 3.
6. as soon as a property is selected and fully described, the developer can select another property by repeating from Step 3 on.
7. as soon as a class is fully described, the developer can continue with another class by repeating from Step 2.

The tool has to be careful not to allow the selection of any element twice. This constraint is stronger than required, and we expect future work and more thorough analysis to relax it and thus extend the expressivity of the tool. This is especially true when moving to more powerful schema languages that are aware of the context an element is used in. We are also aware that the list of sensible properties may be incomplete. We discuss this in Section 7.

We are aware that the intuitiveness of this tool is the main argument for our claim that no RDF expertise is required in order to use our approach. For now we can only argue that the above steps indeed do not require any RDF expertise on the side of the developer, since the RDF structure is never exposed and the developer is fully guided through a number of decisions stated in their domain language. We plan to perform a proper evaluation to test this claim.

## 5 Implementation

In order to demonstrate the feasibility of our approach, we have developed a prototype web service implementation. The input is an arbitrary RDF file. It should describe a person using the FOAF vocabulary. The web service uses an extension of the DTD given in Fig. 1. The full DTD can be accessed online from our demonstration site.[1]

The first step of the output is an RDF file describing one person, using a specified syntax (an incomplete example is given in Fig. 2, for complete examples please refer to the demo site).

Now developers can post-process the output of the serializer with the same ease they would have with any XML files, and they do not need to have any further knowledge of Semantic Web technologies to do so. There are numerous technologies for the further processing of XML files. In our example implementation we use the XML transformation language XSLT [12]. XSLT is capable of translating XML files either to other XML files, or to any other format as well. In our example, we provide an XSLT transformation to turn the resulting RDF file into a vCard [13]. vCard is a pre-XML IETF standard for exchanging contact data, which is in wide use. Fig. 3 gives the full XSLT for this translation. An XSLT file offering the same results over arbitrarily serialized RDF would have been much longer, harder to write, and to maintain.

Applying XSLT transformations to the sample RDF file yields the the result given in Fig. 4.

## 6 Related approaches

In this section we discuss alternative approaches towards bridging the gap between the Semantic and the Syntactic Web. Note that most are more expressive and complete than our approach for now, and thus in some practical settings our may not yet be applicable, but we expect that to change with more maturity (see Future Work in Section 7).

1. The main related approach is to combine or extend XSLT with capabilities to seamlessly deal with RDF, and still continue to provide the same output formatting power. There are a number of implementations towards this goal, like RDF Twig,[2] TreeHugger,[3] or RDFXSLT.[4] XSPARQL[5] provides similar capabilities, but by ex-

[1] http://km.aifb.uni-karlsruhe.de/services/RDFSerializer/
[2] http://rdftwig.sourceforge.net/
[3] http://rdfweb.org/people/damian/treehugger/index.html
[4] http://www.wsmo.org/TR/d24/d24.2/v0.1/20070412/rdfxslt.html
[5] http://xsparql.deri.org/spec/

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                xmlns:foaf="http://xmlns.com/foaf/0.1/">
<xsl:output method="text" indent="yes" media-type="text/x-vcard" />
<xsl:template match="foaf:Person">
BEGIN:VCARD
VERSION:3.0
UID:<xsl:value-of select="@rdf:about" />
N;CHARSET=UTF-8:<xsl:value-of select="foaf:family_name" />;
<xsl:value-of select="foaf:firstName" />;;;
FN;CHARSET=UTF-8:<xsl:value-of select="foaf:name" />
<xsl:for-each select="foaf:mbox">
EMAIL;TYPE=internet:<xsl:value-of select="@rdf:resource" />
</xsl:for-each>
URL:<xsl:value-of select="foaf:homepage/@rdf:resource" />
CLASS:PUBLIC
ORG;CHARSET=UTF-8:;
END:VCARD
</xsl:template></xsl:stylesheet>
```

**Fig. 3.** XSLT for transforming FOAF to vCard.

tending SPARQL. These approaches have all proven feasible, but do not resolve the original issue: they all require the developer to understand RDF.

2. RxSLT and RxPath [14] do not extend the XPath or XSLT syntax, but rather reinterpret them over an abstract XML infoset that is build out of the RDF model. This powerful approach does not require a two step procedure for generating arbitrary output. It has the disadvantage that there exists no actual XML file that the developer can examine and read, but otherwise it achieves most of the goals of our approach as well.
3. A very early approach is DTDMmaker [15] which automatically creates DTDs based on the vocabulary defined in (F-logic) ontologies. It also reports about the same issues identified here (about the shortcomings of DTDs) but is not flexibly customizable and also does not create RDF-conformant DTDs.
4. Another approach is to write custom hard-coded translators, that first read the RDF graph and then create the serialization. This may be the easiest way, but it requires hard-coding a proprietary solution for each use case, and it requires the programmer to learn the API of at least one RDF parser. Our approach does not require any newly written, proprietary code.
5. One approach is to use SPARQL first, and then use the SPARQL XML result format as the basis for the XSLT translations. This approach is viable, but it requires the developer both to understand SPARQL and the SPARQL result format, and then create XSLT scripts to deal with the representation of their data in the SPARQL result format instead of an XML vocabulary that is close to the domain. It has the advantage of not requiring any additional tools to the current set of existing tools.

```
BEGIN:VCARD
VERSION:3.0
UID:http://www.aifb.uni-karlsruhe.de/Personen/viewPersonOWL/id57instance
N;CHARSET=UTF-8:Studer;Rudi;;;
FN;CHARSET=UTF-8:Rudi Studer
EMAIL;TYPE=internet:mailto:studer@aifb.uni-karlsruhe.de
URL:http://www.aifb.uni-karlsruhe.de/Personen/viewPerson?id_db=57
CLASS:PUBLIC
ORG;CHARSET=UTF-8:;
END:VCARD
```

**Fig. 4.** A vCard file created by transforming RDF/XML.

6. Instead of using RDF/XML the developer may also use a more XML-friendly RDF serialization like TriX [16]. But these serializations often simply reify the triple structure in XML, does leading to unwieldy XSLTs and unintuitive XML. Besides that, this approach is feasible and requires only already existing translators to these XML-friendly serializations.
7. The serializers of RDF generating tools could be rewritten so that they always return the required serialization. But this would mean that only output of these tools can be used, and we lack interoperability with other tools using the same ontology, since the XML serialization would usually not be specified. If the serialization indeed is specified, then this would have been done using an XML schema, and therefore it just hard-codes our proposed approach into the RDF source. This is the approach that was chosen for RSS.
8. Another approach is to change the tool using the data so that it becomes RDF aware, i.e. instead of translating the RDF graph to the required format we can enable the tool to use RDF. Even though we personally would prefer this approach, in our example use case this would require to extend all existing tools that can consume vCard to also accept RDF descriptions of persons. This renders this solution unlikely.

We conclude that the major difference of our approach is in the zero-knowledge assumption in using it: no knowledge of Semantic Web technologies is required. Expertise in wide-spread XML technologies is sufficient to start using Semantic Web knowledge bases as data sources.

## 7 Discussion and Future Work

This paper describes an approach towards bridging the gap between classic XML-technologies and the novel Semantic Web technology stack. The current implementation is already usable, but it exhibits a number of limitations. We list these limitations here in order to explicitly name open challenges.

– The given approach can be redone using other, more powerful and modern XML schema languages. These languages add further features like cardinalities.

– DTDs do not allow for context sensitive grammars. Therefore elements can only appear once in every DTD, which severely constraints their expressive power. For example, it is not possible to ensure that every person in a FOAF-file requires a name and mailbox and may have friends (which are persons), and at the same time define that in the friend position the person should not have name and mailbox. Using a context-sensitive XML schema language can remedy this.
– Even without moving to more powerful schema languages, the given constraints in Section 4 can be relaxed. Further analysis is required to understand this bigger language fragment.
– A number of features have not been explored for this first implementation, that future work will take into account, e.g. datatypes, language tags, the `collection` and `xmlliteral` parsetypes, and blank nodes.
– Since OWL depends heavily on blank nodes for its RDF representation, and since blank nodes have not been fully explored yet, it is unclear how to exploits parts of OWL with this approach.
– As we have seen, DTDs have to use fixed namespaces and prefixes, whereas XML namespace-aware schema languages could deal with them more elegantly.
– The creation of compliant schemas can exploit ontologies more thoroughly by using not only domain and range axioms, but also property restrictions, cardinalities and nominals. Furthermore, more expressive descriptions of classes could be used besides simple class names.

For now we provide the current implementation and a web-accessible demonstration workflow to show the advantages of the described approach. We expect that the approach will be applied in a number of early use cases in order to gain more insight in the actual usage and to direct the future development of the given project.

*Acknowledgements* The authors wish to thank Holger Lewen, Malgorzata Mochol, Maria Maleshkova, Markus Krötzsch, Jacek Kopecky, Daniel Herzig, and Sebastian Speiser for their ideas and useful discussions. Research reported herein is supported by the EU project NeOn (IST-2006-027595, `http://www.neon-project.org/`).

## References

1. Berners-Lee, T., Fielding, R., Masinter, L.: Uniform Resource Identifier (URI): Generic Syntax. Technical Report 3986, Internet Engineering Task Force (2005) RFC 3986 (available at `http://www.ietf.org/rfc/rfc3986.txt`).
2. Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F.: Extensible markup language (XML) 1.0 (fifth edition). W3C Recommendation 26 November 2008 (2008) available at `http://www.w3.org/TR/REC-xml`.
3. Beckett, D.: RDF/XML syntax specification (revised). W3C Recommendation (2004)
4. Fallside, D.C., Walmsley, P.: XML schema part 0: Primer second edition (2004) W3C Recommendation 28 October 2004.
5. Clark, J., Murata, M.: RELAX NG Specification (2001) OASIS committee specification.
6. Brickley, D., Miller, L.: The Friend Of A Friend (FOAF) vocabulary specification (2005)
7. Pollock, J.T.: Semantic Web for Dummies. Wiley (2009)
8. Segaran, T., Taylor, J., Evans, C.: Programming the Semantic Web. O'Reilly (2009)

9. Hitzler, P., Rudolph, S., Krötzsch, M.: Foundations of Semantic Web Technologies. Chapman and Hall (2009)
10. Bray, T., Hollander, D., Layman, A.: Namespaces in XML. W3C Recommendation 14 January 1999 (1999) available at http://www.w3.org/TR/REC-xml-names/.
11. Grimm, S., Motik, B.: Closed world reasoning in the semantic web through epistemic operators. In Cuenca Grau, B., Horrocks, I., Parsia, B., Patel-Schneider, P., eds.: OWL: Experiences and Directions, Galway, Ireland (2005)
12. Clark, J.: XSL Transformations (XSLT). W3C Recommendation 16 November 1999 (1999) available at `http://www.w3.org/TR/1999/REC-xslt-19991116`.
13. Dawson, F., Howes, T.: vCard MIME directory profile. RFC 2426, Internet Engineering Task Force (1998)
14. Souzis, A.: Building a semantic wiki. IEEE Intelligent Systems **20**(5) (2005) 87–91
15. Erdmann, M., Studer, R.: How to structure and access XML documents with ontologies. Data and Knowledge Engineering **36**(3) (2001) 317–335
16. Carroll, J., Stickler, P.: RDF triples in XML. In: Extreme Markup Languages 2004, Montréal, Québec, Canada (2004)