

RDF or XML for the Semantic Web?

Joachim Peer

November 25, 2005

Abstract

This position paper questions the view that exclusively RDF should be used to form the data layer of the Semantic Web and it argues that the XML format is a legitimate choice for any Semantic Web application.

1 Introduction

Since the early days of Semantic Web research there has been only limited consensus about what the Semantic Web should be and what is needed to build it. One issue that is under constant dispute¹ is the importance of RDF [3] for the Semantic Web.

There are two schools of thought: One group believes that a globally functioning Semantic Web needs RDF as a common underpinning. The idea is that RDF's triple-concept is generic enough so that other conceptualizations can be constructed upon it to allow all kinds of reasoning², and that that RDF's emphasis on URIs, resources and namespaces provides the necessary prerequisites to achieve global interoperability. Proponents of this school of thought warn that the use of a non-RDF underpinning for Semantic Web applications harms the development of the Semantic Web, eventually leading to a cluster of non-interoperable AI applications instead of the desired Web of knowledge.

The other group argues that data can and should be represented using any type of markup, not just RDF. Their argument builds partly on the consensus among most researchers that there is no and will never be a single type of reasoning technique for Semantic Web applications: Some applications will perform closed-world database-like operations, other applications will resemble AI planners and schedulers, others will perform open-world description logic reasoning and so forth. So why shouldn't the difference in data semantics and reasoning be reflected in the data itself? Why would it be harmful to use XML for data markup considering that XML supports (like RDF) global identifiers (URIs) and namespaces, enabling globally interlinked data repositories?

This paper aims at strengthening the argument of the second group by illustrating that there is very little difference between RDF and XML from the perspective of Intelligent Agents, leading to the conclusion that RDF can not be the exclusive underpinning for the Semantic Web.

¹e.g. in the public-sws-ig@w3.org mailing list

²This has already been exercised by defining languages like OWL and OWL-S (syntactically) on RDF.

The paper starts by rehashing the common consensus about the nature of the Semantic Web and the resp. role of data and inference mechanisms in it, in Sect. 2. After that it elaborates on the core of the “RDF vs. XML” discussion, which is the nature of the mapping of data markup to “semantics”; it describes how this mapping is achieved by RDF and XML respectively, and points out the differences. The weight of these differences and their implications are then discussed in the final Section 4.

2 Consensus about the Semantic Web

There is consensus that the Semantic Web provides data that is – unlike the free-text and HTML-pages of the classical Web – structured in such a way that automated agents can better process it, eventually applying formal reasoning to the data.

There is also consensus that the formal reasoning methods relevant to Semantic Web agents are numerous and may easily proliferate in future, rendering any form of “unified semantic Web calculus” impossible; even if all possible reasoning methods would already be known to date, the complexity of a resulting unified inference method would be far too high to be practically usable.

Further there is consensus that data *per se* is agnostic to the reasoning methods that might be applied to it. From this follows, that the Semantic Web can be separated into two large entities: a) the data and b) the inference methods to be applied to the data.

This resembles the basic idea of modern database and knowledge representation approaches, where data is separated from data processing. Hence, the Semantic Web can be viewed as a (kind of) KR system that is built of a fact base (FB) containing facts about the world and some flexible inference mechanisms. By choosing proper inference mechanisms, the resulting KR system can be used to answer questions using the information stored in the fact base. For instance, given calendaring data of various doctor offices and some GIS and public transport information, an engine might solve problems such as identifying an optimal doctor appointment, as illustrated in [1].

There is also little dispute about the fact that XML and RDF can be used to markup the *raw data* to be processed. The remaining question therefore is: How do XML and RDF differ in transforming the raw data into formal logic based structures, and is there any superior method or exclusion criteria? The answer to this question is prepared in the next section.

3 Mapping Data to Meaning

How can RDF and XML be interpreted logically, i.e. how are agents supposed to generate logical formulas from RDF and XML documents? The answer to this questions illustrates a subtle difference between XML and RDF which is worth some elaboration. As visualized by Fig. 1, the mapping from RDF to logic is implicitly given by RDF, while XML needs some form of additional mapping information that tells the agent how the logics is extracted from the raw data document.

The main point of this paper is that (for the purpose of feeding data into

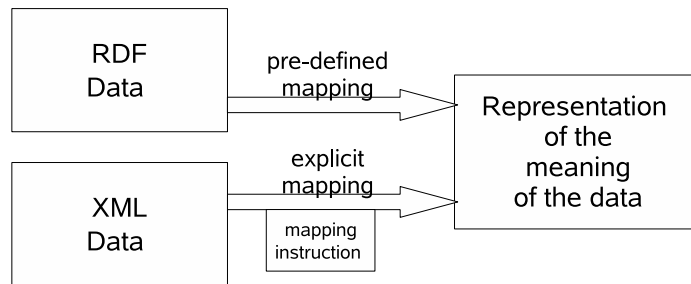


Figure 1: Mapping Data to Meaning

Semantic Web applications) RDF and XML differ only in this single aspect. And, as we will discuss below and in Sect. 4, this is only a theoretical difference with no real impact.

3.1 The RDF Way

In RDF there is a direct mapping between the content of an RDF document and a logic/KR-based representation of the content. This mapping is very straightforward: Each RDF triple $t = \langle \textit{subject}, \textit{predicate}, \textit{object} \rangle$ in the document model can be seen as a binary atomic formula, whose predicate is given by $t.\textit{subject}$, and whose only argument term is named $t.\textit{property}$ and has a value defined by $t.\textit{value}$.

```

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:m="http://some.org/some-voc"
  <rdf:Description rdf:about="http://some.org/B000050HT0">
    <m:artist>Bob Dylan</m:artist>
    <m:title>Essential Bob Dylan</m:title>
    <m:label>Sony</m:label>
    <m:asin>B000050HT0</m:asin>
  </rdf:Description>
  <rdf:Description rdf:about="http://some.org/B00000DQSZ">
    <m:artist>Kraftwerk</m:artist>
    <m:title>Trans-Europe Express</m:title>
    <m:label>Capitol</m:label>
    <m:asin>B00000DQSZ</m:asin>
  </rdf:Description>
</rdf:RDF>
  
```

The above RDF document can be directly transformed into the following set of binary (2-ary) atomic formulas/facts/atoms³

```

( http://some.org/B000050HT0 m:artist="Bob Dylan" )
( http://some.org/B000050HT0 m:title="Essential Bob Dylan" )
( http://some.org/B000050HT0 m:label="Sony" )
  
```

³Note that I use KIF-like “(” and “)” brackets to point out the distinction between XML/RDF markup and the representation of logical formulas!

```
( http://some.org/B000050HT0 m:asin="B000050HT0" )
( http://some.org/B00000DQSZ m:artist="Kraftwerk" )
( http://some.org/B00000DQSZ m:title="Trans-Europe Express" )
( http://some.org/B00000DQSZ m:label="Capitol" )
( http://some.org/B00000DQSZ m:asin="B00000DQSZ" )
```

...whereby "m:" is expanable to `http://some.org/some-voc`. This logical interpretation of RDF documents is shared by many RDF based systems, such as TRIPLE[2].

To sum up, RDF provides a very generic data markup model and provides an implicitly given way of transforming raw data into logic based structures. A limitation of the approach is that it is restricted to triples. An application that requires n-ary triples needs further processing, for instance transforming clusters of triples to n-ary atoms.

3.2 The XML Way

Being just a grammar definition language (designed without KR or the Semantic Web in mind), XML does not provide for a built-in mapping to KR/logic-structures as RDF does.

However, such mappings can be supplied "from the outside" for each XML document (or even for a whole class of documents that adhere to the same syntactic schema). This means that we can tell Semantic Web agents how to (logically) interpret the contents of arbitrary XML files.

Note that this makes us very flexible concerning the targeted representational structure: We can chose to logically interpret XML files as triples (binary atoms) or in terms of a catalog of n-ary atoms, or perhaps as some other type of KR structure.

To sum up, what a Semantic Web agent needs to "logically" represent an XML document is

1. the XML document
2. a kind of "mapping instruction" to be used for the logical interpretation of the document

As mentioned before, there is an arbitrary number of logical target schemas one can chose for XML documents. We briefly examine the following three options:

1. to apply a generic algorithm that transforms all elements and attributes of an XML file into triples (i.e. binary atomic formulas)
2. to apply a generic algorithm that transforms all elements and attributes of an XML file to n-ary atomic formulas
3. to use customized versions of the above-mentioned algorithms

To illustrate these various options, we use the records-example from above. An XML version of the RDF document discussed above might look as shown below:

```

<cd-list xmlns="http://some.org/some-voc">
  <cd>
    <artist>Bob Dylan</artist>
    <title>Essential Bob Dylan</title>
    <label>Sony</label>
    <asin>B000050HT0</asin>
  </cd>
  <cd>
    <artist>Kraftwerk</artist>
    <title>Trans-Europe Express</title>
    <label>Capitol</label>
    <asin>B00000DQSZ</asin>
  </cd>
</cd-list>

```

3.2.1 Transforming XML Documents to Sets of Triples

To illustrate that the transformation of XML documents to triples is a generic concept that can be effortlessly applied, we show a simple algorithm that performs this transformation:

```

void function toTriples(XMLElement elem) {
  subj = getResourceOrLiteral(elem)
  foreach att in elem.attributes {
    create Triple t
    t.subject = subj
    t.predicate = att.name
    t.object = att.value
    add t to fact base
  }

  foreach child in elem.children {
    create Triple t
    t.subject = subj
    t.predicate = child.QName
    t.object = getResourceOrLiteral(child)
    add t to fact base

    toTriples(child) // recursive call
  }
}

function getResourceOrLiteral(XMLElement e) {
  if e has ID attribute
    return e.ID
  else
    return e.textValue
}

```

The data structures in this example should be largely self-explanatory: A Triple structure represents an RDF-triple, which has the member fields `subject`,

predicate and object. An `XMLElement` structure represents an XML element; it may have an `ID` member field, it has a `QName` member field and it has an ordered list of children accessible through a member field `children`. Further, an `XMLElement` structure has a field `attributes` which contains a list of `Attribute` structures, representing XML attributes, each of which having a `name` and a `value`. Finally, the (optional) textual content of an XML Element can be accessed via the field `textValue`.

Shown next is the result of the application of the generic algorithm to the XML document⁴, which may be used for further logic-based processing by the agent:

```
( _1 m:cd _2 )
( _1 m:cd _3 )
( _2 m:artist="Bob Dylan" )
( _2 m:title="Essential Bob Dylan" )
( _2 m:label="Sony" )
( _2 m:asin="B000050HT0" )
( _3 m:artist="Kraftwerk" )
( _3 m:title="Trans-Europe Express" )
( _3 m:label="Capitol" )
( _3 m:asin="B00000DQSZ" )
```

Note that if we would want to skip the unnecessary triples (1 m:cd 2) and (1 m:cd 3), then we would have to customize the algorithm (cf. Sect. 3.2.3).

3.2.2 Transforming XML Documents to Sets of n-ary Ground Formulas

A generic algorithm to create sets of n-ary atoms from XML documents may look as follows:

```
void function toAtoms(XMLElement elem) {
    create Atom a
    a.predicate = elem.QName
    foreach att in elem.attributes {
        a.addTerm(att.name, att.value)
    }

    foreach child in elem.children {
        if(child has no children) {
            a.addTerm(child.name, child.textValue)
        } else {
            a.addTerm(child.name, getResource(e))
            toAtoms(child) // recursive call
        }
    }
    add a to fact base
}
```

⁴with "m:" being expanable to <http://some.org/some-voc>

The data structure `Atom` represents a ground n-ary atomic formula, made up by a predicate (accessible via field `predicate`) and by parameters (terms) which can be added via the `addTerm(name, value)` method.

```
( m:cd-list m:cd some_path_to_cd1 )
( m:cd-list m:cd some_path_to_cd2 )
( m:cd m:artist="Bob Dylan" m:title="Essential Bob Dylan"
  m:label="Sony" m:asin="B000050HT0" )
( m:cd m:artist="Kraftwerk" m:title="Trans-Europe Express"
  m:label="Capitol" m:asin="B00000DQSZ" )
```

Again, if we would want to skip certain unnecessary facts, e.g. the first facts which are created because of the root element, we would need to customize the algorithm (cf. Sect. 3.2.3).

3.2.3 Using Customized Versions of the Above Algorithms

The algorithms sketched in the previous sections can be rewritten to allow customization parameters. For instance, we might want to tell the agent to...

- ignore structures that match a certain XPath, e.g. to ignore the root element `/cd-list` of the XML document above. This suppresses the generation of unnecessary or unwanted facts that do not convey any intended semantics.
- to flatten out nested structures. In case of n-ary facts we may want the following structure...

```
<person id="Joe">
  <occupation role="programmer">
    <company>some.org</company>
  </occupation>
</person>
```

...not be interpreted as...

```
(person id=Joe occupation="_1")
(_1 role="programmer" company="some.org")
```

...rather as:

```
(person id=Joe role="programmer" company="some.org")
```

- to reverse element nestings
e.g. we may want the following structure...

```
<person id="Joe">
  <occupation role="programmer">
    <company>some.org</company>
  </occupation>
</person>
```

...not be interpreted as...

```
(person id=Joe occupation="_1")
(_1 role="programmer" company="some.org")
```

... rather as:

```
(occupation person="_1" role="programmer" company="some.org")
(_2 id=Joe)
```

4 Discussion and Conclusion

	Advantage (+)	Disadvantage (-)
RDF	Provides implicit mapping support	Mapping limited to binary predicates
XML	Mapping not limited to binary predicates	Needs explicit mapping instructions

Table 1: RDF and XML as a basis for Semantic Web applications

Table 1 sums up the discussion so far: RDF provides a generic data markup model and allows a direct mapping of raw data to a logical language. A downside of the approach is that this logical language is limited to triples. An application that requires n-ary triples needs further processing, for instance transforming clusters of triples to n-ary atoms.

XML on the other hand does not provide any pre-defined mapping to the world of logics, but as illustrated in this paper, this kind of mapping knowledge can be explicitly provided. In some cases it suffices to name a generic algorithm to be used to perform the proper mapping, while in other cases explicit customizations are required. The advantage of the latter method is that it is neither restricted to triples nor to any particular flavor of logics.

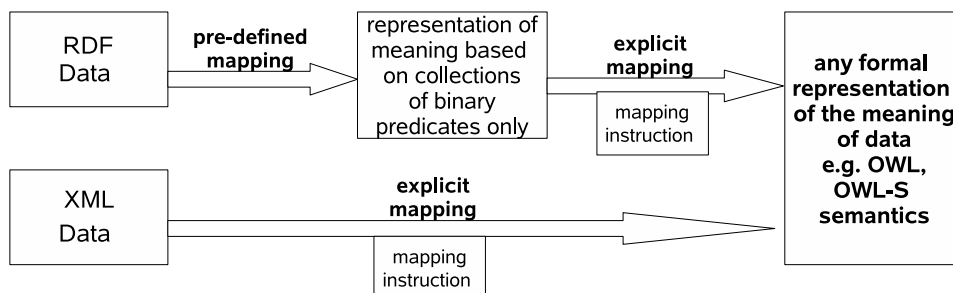


Figure 2: Mapping Data to Meaning - The Complete Picture

We can further strengthen this line of arguments by emphasizing the previously mentioned fact that a set of binary logical formulas (triples) as supported

by the built-in mapping of RDF is rarely the desired semantics for a data document on the Web. For instance, an agent processing an OWL document needs additional mappings from the triple semantics to description logic semantics. An OWL-S document requires even additional knowledge to be efficiently reasoned with. Figure 2 illustrates this: It demonstrates that for all reasoning tasks involving non-RDF semantics (which includes OWL, OWL-S, etc.) some kind of additional mapping from data to semantics is required in any case, no matter if XML or RDF is used.

This analysis shows that the two approaches are very similar. None of them represents a silver bullet for communicating the intended semantics of a piece of data to automats. All non-trivial applications have their own specialized semantic structures to work with, requiring specialized knowledge to map data to semantics structures and reasoning rules.

But when there is no convincing technical argument for any of two discussed languages, it is impossible to rule out one of them as a basis for the Semantic Web, confirming the thesis of this paper.

References

- [1] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.
- [2] S. Decker, M. Sintek, and W. Nejdl. Triple: A logic for reasoning with parameterized views over semi-structured data, 2002.
- [3] O. Lassila and R. Swick. Resource Description Framework (RDF) Model and Syntax Specification, 1999.