

# XRDF - An eXtensible Resource Description Framework

Wolfram Conen  
Xonar IT  
conen@gmx.de

Reinhold Klapsing, Eckhart Köppen  
Information Systems and Software Techniques  
University of Essen, Germany  
{Reinhold.Klapsing, Eckhart.Koeppen}@uni-essen.de

Discussion Paper -- Version: 1.0, January 2001

## Abstract

This paper presents a nested triple model for expressing relations found in the Web. The model allows grouping of atoms and statements on subject and object position. It preserves the structural context in which resource are used. Additionally, we propose a (pure) XML serialization syntax and a graphical representation which equivalently express the formal concepts. On top of the basic structural layer, semantic definitions and interpretations can be layered. One such layer is presented. Finally, the relation of this approach to RDF is discussed and it is argued, that most of the perceived deficiencies of RDF are non-issues in the context of XRDF.

**Keywords:** RDF, simple XML syntax, nested triple, structural context, extensible semantics

**Approximate Word Count:** 5500

## 1 Design Issues

The demand for semantic annotation of resources in the World Wide Web has resulted in a number of standards and techniques that are supposed to create a “Semantic Web”.

Techniques for the specification of meta-data are subject to certain design issues which shall be explained here. Foremost, both syntactic and semantic interoperability have to be achieved, allowing the exchange of machine-readable semantic information. To facilitate the authoring and deployment of meta-data, syntactical and structural simplicity is needed. This is also important to avoid the confusion associated with other meta-data standards. Further, a layered approach has proven to be useful for the definition of models and techniques (this is especially obvious in the context of XML-based standards, where XML is the basis for other standards like namespaces which in turn are used in the definition of XSLT). In the proposed model, we define the syntax on the lowest layer, corresponding to the structural primitive statement and sequence. To these structures, meaning can be attributed via schemata.

An approach that is closely related to our proposal is the Resource Description Framework (RDF). The RDF Model and Syntax specification [10] and the RDF Schema draft [4] have addressed the above mentioned design issues not always in an appropriate way. Lengthy discussions show that the semantics expressed by RDF are not easy to communicate and that the various constructs are defined rather vaguely. Coupled with the cumbersome syntax of RDF, widespread adoption has not yet taken place. Besides,

many related standards or techniques that are used in similar areas such as XML Schema [15, 2], XLink [8], SOX [7], or RELAX [14] have further evolved while RDF has been left behind.

## 2 Model

XRDF will be used to express relations among resources, statements and their groupings. Described informally, XRDF consists of resources and nested expressions over these resources. The key ingredients of this nestable expressions are resources, statements, and lists of resources and statements. Each statement denotes a 3-ary relation (triple) of the form [subject, predicate, object]. In a straightforward interpretation, the resource at predicate position is being viewed as the name of the "property" relation between subject and object. However, the predicate resource provides more than a name for the relation. It is a first-class resource too and can thus have properties on its own. Apart from allowing to define relations between individual resources, XRDF offers the possibility to group resources and statements and to define relations between groups. This is necessary to ease the modeling of nested structures.

### 2.1 Structure

In the following definition, the (infinite) alphabet  $A^*$  will be used.  $A^*$  will denote all possible instances of PCDATA in well-formed XML.

We define the structure  $R$  recursively as an expression over  $A^*$  as

$$R ::= r \mid R, R \mid [[R], r, [R]] \quad (1)$$

Here,  $\mid$  denotes the usual choice. The terminals  $r$  denote elements of  $A^*$ . Further terminal symbols in this expression language are  $[$  and  $]$  and  $,$ , with the latter being provided for readability purposes.

The set  $\mathcal{R}$  of all possible structures  $R$  is called the *Resource set*.

*Remarks:* Each element of  $\mathcal{R}$  can be interpreted as an XRDF model  $m$ . Each model  $m$  contains a set of elements of  $\mathcal{R}$ . The maximal set of these sub-expression of  $m$  is denoted by  $R(m)$ . The expressions contained in  $R(m)$  are either atomic (i.e., elements of  $A^*$ ), lists (i.e., of the form  $R_1, R_2, \dots, R_n$ ) or triples (i.e., of the form  $[[R_s], r_p, [R_o]]$  with  $r_p$  being an element of  $A^*$ ). It is possible to map each element  $r$  of  $R(m)$  to the structure of  $m$  with unique addresses. A formalization is given in the appendix. An example follows:

*Example:*

```
[
  [Reinhold],
  says,
  [
    [ [Wolfram], is, [nice] ],
    [ [Eckil], is, [nice] ]
  ]
]
```

Now, the following addresses for the subexpression `is` can be given:  $3/1/2$  and  $3/2/2$ . Note, that the inverse mapping of  $p$  can be used to retrieve expression from path expressions. This addressing mechanism will be used to allow for a congruent mapping of the formal model to the following representations, XML syntax and graphs.

## 2.2 XML DTD

The model described above forms the basis for the XML representation of meta-data. Its syntax is defined through the following element declarations.

```
<!ELEMENT statement (subject, predicate, object)>
<!ELEMENT list      (statement | atom)+>
<!ELEMENT atom      (#PCDATA)>
<!ELEMENT subject   (atom|statement|list)>
<!ELEMENT predicate (atom)>
<!ELEMENT object    (atom|statement|list)>
```

The atoms and structures which are to be described with meta-data are encapsulated in `atom`, `statement` and `list` elements. These elements directly correspond to the types of sub-expressions in the structural model defined above. The utilization of dedicated elements for these structures facilitates their referencing. Note that the content of an `atom` element carries no further connotation, i.e. at this level of the model, solely strings of character data are described. The `atom` element serves as a container for those strings and makes it possible to define lists of atoms in the `list` element.

A statement is represented by the `statement` element which in turn contains `subject`, `predicate` and `object` elements. This corresponds to the triple-type subexpressions of the structural model. At the subject and object position, lists of atoms and/or statements can be used. The predicate contains exactly one atom. Definition and semantic annotation of predicates are also part of the schema layer of XRDF.

The above *example* expressed in XML syntax is shown in the following.

```
<statement>
  <subject><atom>REINHOLD</atom></subject>
  <predicate><atom>SAYS</atom></predicate>
  <object>
    <list>
      <statement>
        <subject><atom>WOLFRAM</atom></subject>
        <predicate><atom>IS</atom></predicate>
        <object><atom>NICE</atom></object>
      </statement>
      <statement>
        <subject><atom>ECKI</atom></subject>
        <predicate><atom>IS</atom></predicate>
        <object><atom>NICE</atom></object>
      </statement>
    </list>
  </object>
</statement>
```

These declaration are slightly more verbose than the formal structure would require. It is, however, easily possible to map it to the structural model (and vice versa), as will be demonstrated in section 3.

## 2.3 Graphical Representation

In the following anything that is addressable via XPath [5] and XPointer [9] is considered as a resource. Note, that this assumption is in accordance to the definitions given in the XRDF model.

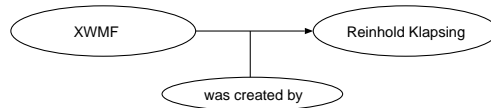
The graphical language provides the constructs oval (representing resources) and directed labeled arcs (representing relations). Each oval representing a resource of the `atom`-type has inscribed a content taken from the alphabet  $A^*$  given as valid PCDATA in well-formed XML.

Each embedded oval will be augmented with a number that is unique within the oval it is directly embedded in. The oval representing the model will be left out. Each top-level oval will be numbered with respect to this implicit oval. Numbers will be left out where possible (ie., in statements, where the ordinal number follows from the direction of the arc, and in nestings with one element only). In the following, some examples are given. A precise transformation to and from the structural model is given in the section 3.

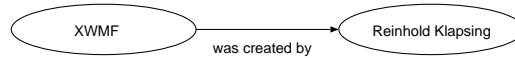
The following proposition shall be represented as a graph.

XWMF was created by Reinhold Klapsing.

This can be represented as

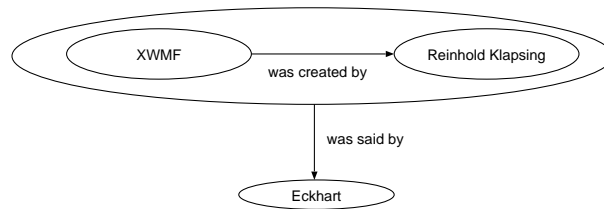


or, for the sake of brevity, as

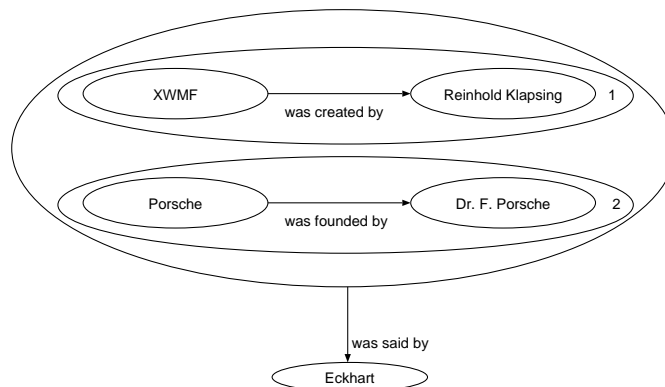


The following proposition is a statement about the previous statement:

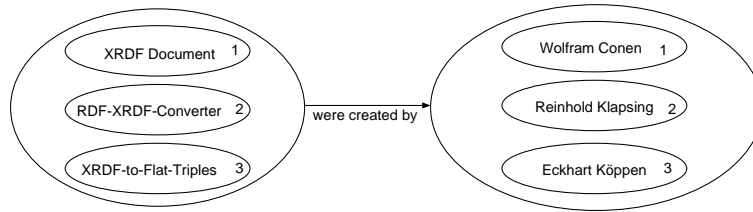
"XWMF was created by Reinhold Klapsing" was said by Eckhart.



If Eckhart made more than one statement, the representation could be



This can also be extended to groups of statements respective resources on both ends of the relation:



## 3 Transformations

### 3.1 From XML Syntax to Nested Triple

The following XSLT stylesheet is provided as a sufficiently precise and complete description of the conversion:

```

<!DOCTYPE xsl:stylesheet>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="xml" />
<xsl:strip-space elements="*" />

<xsl:template match="statement"
              name="statement">[<xsl:apply-templates />]</xsl:template>
<xsl:template match="atom"
              name="atom"><xsl:apply-templates /></xsl:template>
<xsl:template match="subject">[<xsl:apply-templates />], </xsl:template>
<xsl:template match="predicate"><xsl:apply-templates />, </xsl:template>
<xsl:template match="object">[<xsl:apply-templates />]</xsl:template>

<xsl:template match="list">
  <xsl:for-each select="*">
    <xsl:if test="name()='statement'">
      <xsl:call-template name="statement" />
    </xsl:if>
    <xsl:if test="name()='atom'">
      <xsl:call-template name="atom" />
    </xsl:if>
    <xsl:if test="position() &lt; last()">, </xsl:if>
  </xsl:for-each>
</xsl:template>
</xsl:stylesheet>

```

### 3.2 From Graph to Syntax

Deriving the transformation is straightforward. We give an informal procedural description.

1. An oval containing two ovals connected by an arc is transformed to a statement element.
2. An oval having an outgoing arc is transformed to an subject element.
3. A labeled arc is transformed to a predicate element with the arc label as content.
4. An oval having an incoming arc is transformed to an object element.
5. An oval (including the outermost, implicit oval) containing numbered resources is transformed to a list element. The ordinal numbering of embedded ovals will determine their position in the resulting list construct.

6. An expression of  $A^*$  (see formal model) is transformed to an atom element.

Note, that this procedure is invertible.

### 3.3 From Nested Triple to Syntax

A nested triple, as defined in the structural model, can be transformed to the syntax without restrictions. Recall, that each XRDF model is an expression over  $A^*$ , potentially with three types of subexpressions: atomic, triple, or list. The (recursive) transformations  $T$ , starting with a model  $m$  as input, are as follows:

Be  $e$  an element of  $\mathcal{R}$ . The transformation  $T(e)$  is given as:

- If  $e$  is the type *atomic*,  $T(e) = \langle \text{atom} \rangle e \langle / \text{atom} \rangle$ .
- If  $e$  is of the type *list*, i.e.  $e = R_1, \dots, R_n$ ,  $T(e) = \langle \text{list} \rangle T(R_1), \dots, T(R_n) \langle / \text{list} \rangle$ .
- If  $e$  is of the type *triple*, i.e.  $e = [[R_s], r_p, [R_o]]$ ,  
 $T(e) = \langle \text{statement} \rangle \langle \text{subject} \rangle T(R_s) \langle / \text{subject} \rangle$   
 $\langle \text{predicate} \rangle \langle \text{atom} \rangle r_p \langle / \text{atom} \rangle \langle / \text{predicate} \rangle$   
 $\langle \text{object} \rangle T(R_o) \langle / \text{object} \rangle \langle / \text{statement} \rangle$ .

### 3.4 From RDF to XRDF

Models expressed in RDF syntax also can be expressed in XRDF syntax. This depends on the correct interpretation of the RDF syntax. We've build upon the SWI-Prolog RDF Parser to construct an Online RDF-to-XRDF Converter [13] as a proof of concepts.

## 4 Semantics

This section will demonstrate an approach to (extensibly) specify the intended semantics of XRDF models. This is not (yet) a normative and ultimate presentation. However, this section may help to guide the development of future semantic interpretations of XRDF.

We present some key primitives that allow to define how predicates should be applied to list structures and how predicates are related to an underlying knowledge representation (and inference) mechanism. While we tried to keep this set of primitives small, we will try to demonstrate that it is general enough to form a solid base for the construction of meaningful semantics for various potential application domains of XRDF.

The structure-defining primitives *triple* (or *statement*) and *list* (or *sequence*) are the key elements introduced so far. We will now complement this with a mechanism that can be used to attribute meaning to resources and their relations by means of defining predicates.

The semantics of a predicate is ultimately tied to structural transformations (that is, a triple is – due to the nature of the contained predicate – transformed into another representation). On an interpretational level, a predicate names a specific relation between subject and object. In the context of nested triples, with lists allowed on subject and object position, the following four basic types of relations will be distinguished:

1. *Flat (1:1)* – the predicate denotes the relation between the subject (sequence) and the object (sequence).

2. *Expanded (n:m)* – the predicate denotes n\*m relations. It relates each element of the subject sequence with each element of the object sequence.
3. *Flat subject, expanded object (1:n)*
4. *Expanded subject, flat object (n:1)*

The above classification of predicates forms the first aspect of semantic definitions of predicates.

The second important aspect is related to *representation*. By means of the predicate `representedBy`, it is possible to express that an object can represent a subject. This object can then be used as subject or object in other relations. Predicates that have to deal with representatives require a mechanism to express if the relation denoted with the property relates to the representing atom or the represented atoms. To distinguish this, a *d* (for direct) or an *i* (for indirect) will be added to the above  $(x : y)$ -notation.<sup>1</sup>

Example:

```
[ [ Reinhold, Eckhart ], representedBy, [ Friends ] ]
[ [ Wollli ], likes, [ Friends ] ]
```

Now, one possibility is that `Wollli` likes the resource `Friends`, that is `likes` denotes a flat, direct relation ( $d1 : d1$ ).

```
[ [ Wollli ], likes, [ Friends ] ]
    is transformed to
[ [ Wollli ], likes, [ Friends ] ]
```

Another (and slightly more likely) interpretation is, that `Wollli` likes the group of resources that is represented by `Friends`, that is `likes` is of the  $(d1 : i1)$ -type (if the object is a representative).

```
[ [ Wollli ], likes, [ Friends ] ]
    is transformed to
[ [ Wollli ], likes, [ Reinhold, Eckhart ] ]
```

A third (and even more likely) interpretation is, that `Wollli` likes each of the represented resources, that is `likes` is of the  $(d1 : im)$ -type.

```
[ [ Wollli ], likes, [ Friends ] ]
    is transformed to
[ [ Wollli ], likes, [ Reinhold ] ]
[ [ Wollli ], likes, [ Eckhart ] ]
```

Note, that we expect an algorithm fixing the processing of transformations to apply indirection recursively.

*Non-deterministic choice* is the third key aspect. A predicate can be defined to non-deterministically choose an element from the subject and/or the object sequence to relate a resource to it<sup>2</sup>. This non-deterministic choice will be denoted by substituting *c* for *n*, *m* or  $1^3$  in the above predicate type declarations.

---

<sup>1</sup>Note, that an *i* will be treated as a *d* if the subject resp. object is not a representative. If neither *d* or *i* are given, *d* is assumed. Note also, that 1 is a subcase of *n*. The default type for a predicate will be  $1 : 1$ .

<sup>2</sup>Note, that the choice allows to formulate an equivalent to existence quantification on relational level. “For all” quantification can be expressed with expansion.

<sup>3</sup>Which is meaningless but does no harm.

The type of the predicates will be attached to the predicates with the predicate `pType`.

Example:

```
[ [ Wolli ], likes, [ Friends ] ]  
[ [likes], pType, [dl:in] ]
```

A final aspect is the specification of a mapping of triples with specific predicates to languages that define further semantic aspects. For this purpose, the predicate `isDefinedAs` is introduced.

## 4.1 Semantic Templates

The primitives introduced so far allow to abbreviate the syntactic formulation of models. It is possible to define this formally as a transformation. The predicate `isDefinedAs` is necessary to define the semantics of predicates in terms of the underlying knowledge representation. The examples given will be in Prolog, however, the approach is open to other representations.

In the following, the definition of a predicate that allows to conveniently track transitive relationships is demonstrated. Please, note, that the knowledge base will have to represent the XRDF statements in some reasonable manner to make predicate definitions meaningful. Here, it is assumed that a (Prolog) predicate *statement* exists, that contains all statements that can recursively be discovered within a given XRDF model.<sup>4</sup>

```
[ [ path ],  
  isDefinedAs,  
  [ path(X,Y) :- statement(X,path,Y).  
    path(X,Z) :- statement(X,path,Y), path(Y,Z). ]  
]
```

Now, the knowledge level predicate *path* could be used to compute the transitive closure following from the relation that is given by the statements with the resource `path` in predicate position.

```
[ [ A ], path, [ B ] ]  
[ [ B ], path, [ C ] ]
```

Given these facts, *path(A,C)* will be a consequence of the `path` definition in the knowledge level representation of the XRDF model.

## 4.2 Transforming XRDF to a Knowledge Base

With respect to prolog, the relative order of knowledge becomes important. With XRDF and its inherent notion of order, this poses no problem.

The process of transforming the XRDF document to a knowledge representation is as follows:

1. The model, consisting of an expression over  $A^*$ , is expanded according to the transformations defined above.

---

<sup>4</sup>Additionally, for each such statement, the set of addresses of its occurrences as discussed in the subsection on the formal model and in the appendix, should be retrievable. An Online version of a Prolog-based XRDF parser with semantic extensibility is available at <http://wonkituck.wi-inf.uni-essen.de/xrdf.html>.

2. The expanded model is asserted into the knowledge base, giving the base facts.
3. The rules that are defined for predicates by means of `isDefinedAs` are added to the knowledge base.

The details of step 2 and step 3 will depend on the underlying knowledge processing mechanism. The basic structural transformations described above can be applied in a pre-processing step or can be directly encoded into the predicate definitions. Two additional, convenient types of transformation are described in the appendix. The XRDF Parser mentioned above uses a generic pre-processing step to compute the expansions and determines all sub-expression addresses for further reference<sup>5</sup>. Further details of schema level semantics will be discussed in a follow-up paper.

## 5 Discussion

In the following we discuss some of the issues which are problematic with the syntax defined in the RDF Model and Syntax specification (RDF-M+S) and which are resolved by the syntax presented in this paper. Further approaches are discussed which also propose the usage of alternative RDF syntaxes.

In RDF-M+S the ID attribute<sup>6</sup> is used to give a conceptual entity a name. Even though it is possible to define in our syntax an ID attribute in accordance to the XML recommendation we think the interpretation of a string as a name already should be shifted to a semantic level. At a schema level a predicate (`hasName`, for example) can be defined to express this relationship. Referencing a resource by name can be done by expressing this relationship at a semantic level. This also avoids the constraint to use a fragment identifier to refer to a named resource.<sup>7</sup>

In RDF-M+S XML namespaces [3] are used to associate each property with the schema that defines the property. However, RDF-M+S postulates that every “named” resource is named by an URI (plus an optional optional anchor id). In the triple model there exists no mechanism to capture a `namespace name`, a `namespace Prefix`, and a `LocalPart name`. Also, the derivation of a `namespace name` and a `LocalPart name` from an URI is not formally defined.<sup>8</sup> This underpins the above argumentation that giving a resource a name should be done by explicitly expressing this relationship on a semantic level. Further RDF-M+S is unclear about when a namespace prefix referencing the RDF namespace should be used. The examples given in RDF-M+S sometimes use a namespace prefix on attributes and sometimes not. In accordance with [3] unprefixes RDF attributes on element types in the RDF namespace are not equivalent to attributes with a prefix. Besides this, according to the EBNF given in RDF-M+S, attributes actually *must not* be associated with a namespace which means in fact all RDF attributes are unqualified. The approach presented in this paper does not constrain one to use XML namespaces for capturing RDF models. However, XML namespaces can be used complementary to our approach to mix markup from different DTDs with the markup presented in this paper.

---

<sup>5</sup>For further details, please, see <http://wonkituck.wi-inf.uni-essen.de/xrdf.html>. Examples for semantics capturing the RDFS rules and constraints discussed in [6] can also be found.

<sup>6</sup>The way the ID attribute is used in RDF is not in conformance with the XML recommendation because no DTD is given which defines the RDF attribute ID of type ID. This is one problem resulting from the fact that it is not possible to give a general DTD for the RDF syntax.

<sup>7</sup>Using a fragment identifier is problematic because the meaning of a fragment identifier depends on the MIME type of an entity associated with the resource identified by the URI part. (1) Currently there exists no MIME type for RDF documents and (2) the MIME type negotiation is part of the HTTP protocol and therefore out of scope of RDF. In an RDF model it is not clear what entity exactly is referenced by a fragment identifier.

<sup>8</sup>See also discussion in the `www-rdf-interest` mailing list starting at <http://lists.w3.org/Archives/Public/www-rdf-interest/2000Jul/0037.html>

The example of XML namespaces shows that there are some issues that are captured by the RDF/XML serialisation syntax but not by the model. Another examples are the `xml:lang` attribute and the `aboutEachPrefix` attribute. The information expressed with this attributes has no counterpart in the triple model. This leads to the general question: To what extent are the different representations of RDF equivalent? Four representations have been identified: (1) formal/data model, (2) graphical representation, (3) serialization syntax, (4) triple (a subset of (1), but often used as “the RDF model”). An interesting example of a problematic concept are anonymous resources. RDF-M+S is not clear about the representation of anonymous resources in the different representation. The necessity for anonymous resources stems from the constructs that are available in the serialization syntax. In the formal model and the triple representation, there exist no anonymous resources. The formal models states that “There is a set of resources”. No alphabet is given to allow to determine which entities belong to the set of resources. Assuming that an alphabet is necessary to complete the specification of the model, “anonymous” resources can not be part of the formal model, because any atom  $r$  will be a member of the alphabet, and on the level of the formal model, this “name” is all that is known about  $r$  (in essence,  $r$  is this name, and nothing else). What can be introduced easily, however, would be a *Name* (as the object of a property) for a resource (in addition to an identifier, which might be explicit or may follow from document structure). Then, an “anonymous” resource (on schema level) would simply be a resource without a *Name*. Note, that this is easily possible with the XRDF approach. However, the necessity for anonymous resources is almost not present, because statement about statement and groupings are easily possible without reification. To summarize: In the approach presented in this paper we propose a formal model, a serialization syntax and a graphical representation which equivalently express the concepts underlying our approach.

The definition of identity or existence of resources leads also to the question whether an RDF model should allow different statements with the same subject/predicate/object. We think this also should be shifted to the schema level where the semantics of an application should be defined. Applications may use the same statement several times but in different contexts. It will depend on the interpretation on the schema level whether identical resources (remember: statements, atoms or lists) are considered to be equivalent (neglecting the context) or not. The nested triple model presented in this paper supports context-aware applications allowing multiple usage of the “same” statement in different contexts.

In order to define relationships between collections of resources grouping is needed. In RDF-M+S the collection classes `Bag`, `Seq`, and `Alt` are defined. Each of this class has a different semantic which again should be expressed at schema level. But the grouping of resources and statements on which one want to express relations - we think - should have an counterpart in the different representations of metadata. Grouping is a a big strongness of XML which is exploited by XRDF. On subject position and object position lists of resources and statements are allowed. The underlying model and the graphical representation of XRDF also capture the grouping information. Note, that the nested triple model of our approach avoids the usage of predicates as `rdf:_1`, `rdf:_2`, . . . which are applied in RDF-M+S to capture sequences. Capturing sequences this way lead to some problems: (1) An RDF parser needs to have special knowledge about collection classes in order to recognize that the contained `rdf:li` properties are are sequences ordered by `rdf:_1`, `rdf:_2`, etc. (2) The usage of `rdf:_1`, etc. makes is difficult to add a resource to a collection, since the agent doing the assertion cannot be sure of knowing what property name to use next.

The model defined in RDF-M+S constraints that `Literals` (which are defined as inline resources) only can be used on object position. The result is that it is impossible to make a statement about an (anonymous) inline resource. This does not seem to be reasonable. The approach presented in this paper allows `Literals`<sup>9</sup> also to be used on subject position.

In [1] Tim Berners-Lee proposes an unstripped syntax for RDF in XML. Unstripped means that in this

<sup>9</sup>Note that associating any PCDATA (on subject and object position) to a certain class (Literal for example) is left to a primitive at the semantic level

approach nested elements which describe alternatively nodes and arcs are not used. Instead RDF arcs (properties) correspond to XML elements, and RDF nodes are implicit. Design issues which were taken into account are: (1) RDF can be carried within other XML information and can have other XML information inserted within the syntax (2) It should be possible to make a document which efficiently expresses information and allows an RDF parser aware of the syntax to extract the RDF graph without needing to read the namespace schema. (3) RDF can carry extensions which can be optional. Sergey Melnik introduces a proposal in [12] which is based on the ideas shown in [1]. A reference implementation is available. Additionally Melnik proposes in [11] to allow that every “legacy” XML document can have an RDF model. The above proposals have in common that unstripped syntax is used that does not explicitly describe the RDF graph. However, in both proposals it is assumed that an RDF graph can be built from the implicit RDF description. The syntax introduced in this paper is complementary to the above ones as an RDF graph is *explicitly* described by our syntax. Furthermore, the approaches mentioned above suffer from the restrictions of the RDF model and the problems related to RDF schema (no formal semantics, no formalisable extensibility mechanism). However, they share the benefit of standard XML syntax with our approach.

In order to summarize the discussion we want to point out the following issues. This paper presents a simple XML syntax and a nested triple model for expressing relations on the Web. The model allows grouping at the subject and object position and preserves the context (information) a resource is used in. A layered approach is applied. At the lowest level nothing is said about the semantic interpretation of the relations. At a higher level primitives can be introduced to express different semantic interpretations of the same low level syntax. We propose to shift the definition of as much semantics as possible to a higher level. At a schema level primitives are defined which are used to express the semantics of a certain application. The approach supports the expression of formal semantics and an extensibility mechanism. It is avoided to mix up syntactic and semantic primitives. The different representations (model, graphical representation, and XML serialization) of XRDF meta data are equivalent in expressiveness. We tried to avoid as much issues as possible we have seen the RDF community is feeling unhappy with in respect to the RDF. However, we have greatly profited from the ideas presented in RDF-M+S and the RDF schema specification.

## References

- [1] Tim Berners-Lee. A strawman Unstripped syntax for RDF in XML. Technical report, W3C, May 1999. <http://www.w3.org/DesignIssues/Syntax.html>.
- [2] Paul V. Biron and Ashok Malhotra. XML Schema Part 2: Datatypes. Candidate recommendation, W3C, October 2000. <http://www.w3.org/TR/2000/CR-xmlschema-2-20001024/>.
- [3] Tim Bray, Dave Hollander, and Andrew Layman. Namespaces in XML. Recommendation, W3C, January 1999. <http://www.w3.org/TR/1999/REC-xml-names-19990114/>.
- [4] Dan Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Specification 1.0. Candidate Recommendation, W3C, March 2000. <http://www.w3.org/TR/2000/CR-rdf-schema-20000327>.
- [5] James Clark and Steve DeRose. XML Path Language (XPath) Version 1.0. Recommendation, W3C, November 1999. <http://www.w3.org/TR/1999/REC-xpath-19991116>.
- [6] Wolfram Conen and Reinhold Klapsing. A Logical Interpretation of RDF. *Linköping Electronic Articles in Computer and Information Science*, 5, 2000. <http://www.ida.liu.se/ext/epa/cis/2000/013/tcover.html>.

- [7] Andrew Davidson, Matthew Fuchs, Mette Hedin, Mudita Jain, Jari Koistinen, Chris Lloyd, Murray Maloney, and Kelly Schwarzhof. Schema for Object-Oriented XML 2.0. Note, W3C, July 1999. <http://www.w3.org/1999/07/NOTE-SOX-19990730>.
- [8] Steve DeRose, Eve Maler, David Orchard, and Ben Trafford. XML Linking Language (XLink) Version 1.0. Candidate recommendation, W3C, July 2000. <http://www.w3.org/TR/2000/CR-xlink-20000703/>.
- [9] Ron Daniel Jr., Steve DeRose, and Eve Maler. XML Pointer Language (XPointer) Version 1.0. Candidate recommendation, W3C, June 2000. <http://www.w3.org/TR/2000/CR-xptr-20000607>.
- [10] Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. Recommendation, W3C, February 1999. <http://www.w3.org/TR/1999/REC-rdf-syntax-19990222>.
- [11] Sergey Melnik. Bridging the Gap between RDF and XML. Technical report, Stanford University, December 1999. <http://www-db.stanford.edu/~melnik/rdf/fusion.html>.
- [12] Sergey Melnik. Simplified Syntax for RDF. Technical report, Stanford University, December 1999. <http://www-db.stanford.edu/~melnik/rdf/syntax.html>.
- [13] Online RDF-to-XRDF Converter. <http://wonkituck.wi-inf.uni-essen.de/xrdf.html>.
- [14] RELAX (REgular LAnguage description for XML). <http://www.xml.gr.jp/relax/>.
- [15] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. XML Schema Part 1: Structures. Candidate recommendation, W3C, October 2000. <http://www.w3.org/TR/2000/CR-xmlschema-1-20001024/>.

## Appendix

### Addressing Resources within the Model

Given an expression  $m$ . It is possible to map each element  $r$  of  $R(m)$  to the structure of  $m$  as follows:

Let  $e_r$  be the (sub-)expression directly embracing  $r$ .

1. Atomic:  $p(r) = p(e_r(r)) = 1$  for any  $e_r$ .
2. In list: Given  $e_r = R_1, \dots, R_N$ ,  $r = R_i$ ,  $1 \leq i \leq n$ , then  $p(r) = p(e_r(r)) = i$
3. In triple: Given  $e_r = [[R_1], R_2, [R_3]]$ ,  $r = R_i$ ,  $1 \leq i \leq 3$ , then  $p(r) = p(e_r(r)) = i$ .

Now, for any given subexpression  $S$  of  $m$ , a path to each instance  $s$  of  $S$  can be given as follows:

$$p(s) = p(e_s) + / + p(e_s(s))$$

with  $+$  denoting a concatenation, and  $e_s$  being the (sub)expression which directly embraces  $s$ .

This allows to address uniquely all instances of  $s$  within a given expression  $m$ . An example is given in section 2.1.

## Convenient Predicate Types

As a useful abbreviation for predicate assignments, a fifth basic type can be introduced: *Descriptive* ( $1 \rightarrow n$ ). Be  $p$  a descriptive predicate. Let  $[[A], p, [B]]$  be a statement with  $A$  being an expression of arbitrary type and  $B$  being a list with an even number of elements, say  $n$ . It is required, that every element with an odd index is of type `atomic`. The statement will be transformed to  $n/2$  statements of the form  $[[A], b_i, [B_i + 1]]$ ,  $i \in 1, 3, \dots, n - 1$ . A subtype allowing for one-level indirection is ( $1 \rightarrow in$ ). Here,  $B$  is expected to be either a list or a representative for a list of the required structure. An brief example will demonstrate the application of descriptive predicates (in fact, it is not required to define more than one such predicate).

```
[ [describedBy], ptype, [1->in] ]
```

allows to formulate the following abbreviations:

```
[ [ Reinhold ], describedBy, [hasName, Klapsing, hasAdress, Essen] ]
```

which will be transformed to

```
[ [ Reinhold ], hasName, [Klapsing] ]
```

```
[ [ Reinhold ], hasAdress, [Essen] ]
```

A sixth type will be denoted with *Straight* ( $n \leftrightarrow n$ ). Be  $p$  a straight predicate. Let  $[[A], p, [B]]$  be a statement with  $A$  and  $B$  being both expressions of the type `list` with a length of  $n$  or both of the type `atomic` (resembles  $1 : 1$ ). The statement will be transformed to  $n$  statements of the form  $[[A_i], p, [B_i]]$ ,  $1 \leq i \leq n$ . Note that a recursive application of a straight predicate (in case of indirection) is only valid, if subject(s) and object(s) are of *deep* structural equivalence. The type can also be usefully combined with indirection. An example follows:

```
[ [hasName], ptype, [n<->n] ]
```

This makes it convenient to declare the types of several predicates with one statement, for example:

```
[ [ Reini,Ecki,Wolli ], hasName, [ Klapsing,Koeppen,Conen ] ]
```

Please note, that all the structural transformations will influence the (original) context of resources. If the context is relevant, for example for determining the equivalence of identical subexpressions, the semantics of the predicate definitions have to be crafted very careful. It may become necessary to map the procedural knowledge underlying the transformation into the knowledge level to allow to keep track of transformations. This can easily be done by leaving out the pre-processing and including the transformations into the knowledge-level definitions of the predicates (with `isDefinedAs` and appropriate rules).