

Conventions for Knowledge Representation via RDF

Philippe Martin and Peter Eklund

Griffith University, School of Information Technology, PMB 50 Gold Coast MC, QLD 9726 Australia
Tel: +61 7 5594 8271; Fax: +61 7 5594 8066; E-mail: {philippe.martin,p.eklund}@gu.edu.au

Article accepted at the WebNet 2000 conference.

Abstract. The Resource Description Framework [RDF] provides a basic model to describe relationships between objects. Ultimately, it is intended to permit the representation, combination and processing of most types of metadata from Web-accessible documents or databases. However, except for representing simple metadata, its current XML-based syntax [RDF syntax] and the set of basic concept/relation types that have been defined [RDF schema] are insufficient. To make extensions, the users are required to declare new concept/relation types in "schemas" or import schemas from other users. The problem is that similar/identical basic types or features will probably be introduced by various users via different type names or used in different ways, and this prevents the comparison, reuse and combination of the metadata. To maximize the reuse of metadata, we propose some lexical, structural and semantic conventions, inspired from various knowledge representation projects. These conventions would have to be agreed on and completed by the W3C committee.

Topics/Keywords. Data and Link Management, Metadata Representation/Retrieval/Reuse

1. Introduction

The Resource Description Framework [RDF] "can be characterized as a simple frame model" [RDF syntax]. It is sufficiently low-level and general for most other knowledge representation models to be translated into. However, since it is low-level, there are many possible ways for such translations, and there are many ways for users to represent the same fact. These various ways are not comparable and therefore, without additional features and conventions, RDF cannot support metadata exchange and reuse. As noted by [Berners-Lee et al., 1999], "work is needed to define common terms for [extending RDF to the power of usual knowledge representation systems]".

RDF currently has constructs for representing simple existential graphs plus some kinds of sets and contexts and some kinds of type declaration. [Berners-Lee, 1999] proposes some additional constructs for representing universal quantification. RDF users can declare new concept/relation types to introduce additional features, and give some constraints (essentially via relation signatures) but no real type "definition" is as yet possible.

A usual concern is that an increase in expressiveness leads to a model and a language too complex to handle efficiently. In actual fact, like other XML agents, the RDF/XML analyzers will exploit some terms (those representing features they know how to handle) and ignore others. Thus, the level of complexity dealt with is not determined by the language but chosen by the user via the selected analyzer. Some applications require the exploitation of rules, sets, logical negation and contexts, whereas simple structure matching may be sufficient for a search engine.

In Section 2, we propose lexical, structural and semantic general conventions, synthesizing conventions from various knowledge representation communities. In Section 3, we propose ways to apply and extend RDF/XML in various logical cases of knowledge representation.

2. General conventions

These conventions apply not only to RDF but any knowledge representation language that can be translated into a directed graph model such as RDF. Rather than using RDF terminology, we use the more intuitive terminology of Conceptual Graphs [CGs]. A "concept" refers to a node ("resource" in RDF; it may represent 1 or several objects). A "relation" ("property" in RDF) refers to a relationship between concepts. A "type" ("class" in RDF) refers to a term (or "identifier") referring to a certain kind of concept or relation. An "ontology" (or "schema") is a set of type declarations.

2.1. Lexical normalisation

InterCap style for identifiers. Identifiers in RDF/XML must have legal XML names. The "InterCap style" has been adopted, with a lower case first letter for relation types [RDF syntax (appendix C)] -- as in *rhetoricalRelation* and *subClassOf* -- and an upper case first letter for concept types [RDF schema (Section 1.2.2)] -- as in *TaxiDriver*.

High-level lexical facilities. To reduce lexical problems and promote metadata reuse, high-level languages or query interfaces should provide lexical facilities for the user. For instance, language analyzers could automatically normalize identifiers that include uppercase letters, dashes or underscores into the InterCap style, as well as exploit user-defined aliases. Such analyzers should also accept queries or representations that use undeclared type names (e.g. common words) when the relevant type names can be automatically inferred via the structural and semantic constraints in the queries or representations and the ontologies they are based upon. When different interpretations are possible, the user should be alerted to make a choice. This last facility, detailed in [Martin & Eklund, 1999], is particularly interesting when the exploited ontologies reuse a natural language lexical database such as WordNet [WN]: it spares the user the complex (and tedious) work of declaring and organizing each term used. This facility (along with high-level notations and interfaces) seems an essential step to encourage Web (human) users to build knowledge representations. Similar ideas for the exploitation of lexical databases such as WordNet are developed in Ontoseek [Guarino et al., 1999].

Nouns for identifiers. The convention of using nouns, compound nouns or verb nominal forms whenever possible within representations not only makes them more explicit, it also efficiently reduces the lexical and structural ways they may be expressed.

Concept types referred to by adjectives can rarely be organized by generalization relations but may be decomposed into concept types referred to by nouns. Concept types referred to by verbs can be organized by generalization relations but cannot be inserted into the hierarchy of concept types referred to by nouns (and therefore cannot be compared with them) unless verb nominal forms are used. These nominal forms, e.g. *Driving*, also recall the need to represent the time frame or frequency of the referred processes. For similar reasons, value restrictors should also be represented via nouns, e.g. *ImportantWeightForAMouse* and *ImportantWeightForAnElephant*, rather than via adjectives such as *Important*. Most identifiers in current ontologies are nouns (e.g. the Dublin Core [DC] or the Upper Cyc Ontology [CYC]), even in relation type ontologies such as the Generalized Upper Model relation hierarchy. Avoiding adverbs for relation type names is sometimes difficult, e.g. for spatial/temporal relations. What should be avoided is the introduction of relation types names such as *isDefinedBy* and *seeAlso* (both proposed in [RDFschema]). Better names are *Definition* and *AdditionalInformation*.

Singular nouns for identifiers. Most identifiers in ontologies are singular nouns. Category names must be in the singular in the Meta Content Framework Using XML [MCF/XML]. For the sake of normalization, it is therefore better to avoid the use of plural identifiers whenever possible, e.g. by using "distributive sets" (that is by using the RDF keyword "aboutEach" instead of "about" whenever possible).

2.2 Structural and semantic normalisation

Binary basic relations. As with most frame-based models, RDF only has binary and unary relations. Relationships of greater arity may still be represented by using structured objects or collections, or using more primitive relations. For instance, "the point A is between the points B and C" may be represented using the relation type *between* and a collection object grouping B and C, or using the relation types *left* and *right*, *above* and *under*, etc. Most often, decomposition makes a representation more explicit, precise and comparable with other representations.

Thus, relations should refer to simple/primitive relationships because complex relationships cannot be compared without special rules being added. As a rule of thumb, relations should not refer to processes and should -- whenever possible -- be named with simple "relational nouns", e.g. *part* and *characteristic*. Some complex relational nouns such as *child* and *driver* are often too handy to be avoided but imply additional lexical or structural facilities (e.g. those of Ontoseek).

Avoid disjunctions, negations and collections. Representations including disjunctions, negations or collections are generally less efficiently exploitable for logical inferencing than conjunctive existential formulas and IF-THEN rules based on these formulas [BRML].

It is often possible to avoid disjunctions and negations without loss of expressivity using IF-THEN rules or by exploiting type hierarchies. For instance, instead of writing that an object X is an instance of *DirectFlight* OR of *IndirectFlight*, it is better to declare X as an instance of a type *Flight* that has *DirectFlight* and *IndirectFlight* as exclusive subtypes (i.e. types that cannot have common subtypes or instances). Exclusion links between types (or between whole formulas) are kinds of negations that can be handled efficiently, and are included in many expressive but efficient logic models, e.g. Courteous logic on which the Business Rules Markup Language [BRML] is based. The introduction of identifiers for collections may also often be avoided using "distributive collections", i.e. in RDF by using the keyword "aboutEach". Distributive collections are often easy to handle since they can be considered as

syntactic shortcuts for representing relations about each of their members. Type definitions describing typical or necessary relations associated with the type instances are also a way of representing facts about collections of objects that knowledge representation systems generally handles more efficiently than if these relations were directly represented using (real) collections inside other assertions.

Precision, term definitions and constraints. The more precise the representation the less chance of conflict with another. The more primitive its components, the more likely the representation can be cross-checked and compared with others to respond to queries. Representations should be contextualized in space, time and author origin. No relevant concepts should be left implicit. It is stated in [RDF syntax, Section 2.3] that for some uses, writing property values without qualifiers is appropriate, e.g. "the price of that pencil is 75" instead of "the price of that pencil is 75 U.S. cents". However, a representation of the first sentence would be ambiguous, not comparable with other prices. This violates the original purpose of RDF.

To improve precision and allow consistency checks, it is important to use precise types and associate constraints about their use. At least, signatures should be associated to relation types, and exclusion between types represented. Ontologies ("schemas") organize terms and provide constraints for their use.

To improve the retrieval of technical types (terms) or representations using them, it is important that these types specialize usual terms. One way to do this is to specialize formal terms from a natural language ontology such as WordNet with the domain-oriented terms. Extending such an ontology is often quicker and safer than creating an ontology from scratch, ensures a better reusability of the representations and automatic comparisons with representations based on the same ontology. These issues are discussed and implemented in [LOOM].

3. Terms and notations for logical cases

We now propose some extensions to the RDF syntax or basic set of terms. Following our conventions, the terms we use in our examples -- apart from keywords, URLs and some relation types -- are WordNet nouns.

3.1. Simple graphs: individuals and existential quantifiers

RDF supports the representation of typed individuals, existentially quantified variables, and relations between them. The only extension that seems handy at this stage is a special relation attribute (named "of" for instance) to indicate that the direction of the relation is reversed. Example:

```
E (English): On 12/3/2000, an employee of IBM updated a line of IBM's home page.
RDF: <Company ID="ibm" name="IBM"/>
    <Update><agent><Person><employee of resource="#ibm"></Person></agent>
        <object><Line><part of><File><homePage of resource="#ibm"></File>
            </part></Line></object>
    <time>12/3/2000</time> </Update>
```

Representing the same information without the keyword "of" would imply more graphs (smaller ones) or the use of relation types such as *partOf*, *homePageOf* and *employer*. To permit comparisons of graphs, these relation types would have to be declared as inverse of *part*, *homePage* and *employer*. There is not yet a standard way to do so. If there was, parsers taking it into account would be less efficient.

3.2. Contexts

Syntactically, a context is a concept which embeds other concepts (possibly linked by relations). Semantically, a context represents a situation (i.e. relationships between objects in a real or imaginary world) or a statement (i.e. a description of a situation). As any other concept, a context may be referred to via an individual identifier or an existentially quantified variable. Thus, it is possible to describe relations from/to them and therefore about their content. In RDF/XML, the keyword "aboutEach" must be used to do so.

Some of these relations involve situations, e.g. to situate them in time, while others involve statements, e.g. to state that they have been authored by someone at a certain time. The signatures of such relations is important information for checking or classifying the types of contexts connected by such relations. However, explicitly typing contexts with *Situation* or *Statement* to comply with the relation signatures is not intuitive, and it leads to lengthy representations and rather arbitrary decisions. For instance, can logical relations be directly connected to Situation concepts or is an intermediary Statement concept always necessary? This problem has not yet been tackled by the RDF specifications [RDF syntax, RDF schema], only one type of context is used: *rdf:Description*.

We propose that the RDF parsers still accept *rdf:Description* as a generic type for contexts, but automatically deduce their adequate types (*Situation* or *Statement*) and the implicit intermediate contexts. Even with this facility, the notation for contexts quickly becomes cumbersome and would need to be adapted. The next example shows this need.

```
E: Tom believes that Mary now likes him (in 1999) and that before she did not.
RDF: <Person ID="Tom"/>
      <rdf:Description bagID="s"><Liking><agent><rdf:Description ID="Mary"/></agent>
          <object resource="#Tom"/>
      </Liking>
    </rdf:Description>
    <rdf:Description bagID="not_s" aboutEach="#s" truth="false"/>
    <rdf:Description bagID="p1" aboutEach="#s" time="1999"/>
    <rdf:Description bagID="p2" aboutEach="#not_s" before="1999"/>
    <rdf:Description aboutEach="#p1"><believer resource="#Tom"/></rdf:Description>
    <rdf:Description aboutEach="#p2"><believer resource="#Tom"/></rdf:Description>
```

In this example, we have represented negation using the relation type *truth*. A better convention might be to use the context type *Negated_description* or simply *Not*. [Berners-Lee, 1999] also proposes such terms ("truth" and "Not"), plus an IF-THEN construct to allow the representations of rules. Though we reuse this construct in the next section, an alternative is to use relations with types such as *implication* or *equivalence*. Modalities could be represented via the relation type *modality* and an instance of an agreed-on set of modality types. It is not the purpose of this article to propose an ontology of modalities but we would like to emphasize that for the sake of knowledge reuse such issues should be made part of the RDF standard.

3.3. Collections and intervals

Relations from a collection considered as a whole (i.e. as a single object) can be represented via existential statements. In the following example, we introduce the type name *Set* to specify that the members cannot be identical.

```
E: Fred, Wilma and another man collectively approved a resolution.
RDF: <Set ID="g"><rdf:li resource="#Fred"/> <rdf:li resource="#Wilma"/>
      <rdf:li><Man/></rdf:li> </Set>
    <rdf:Description about="#g">
      <approver of><Resolution></approver> </rdf:Description>
```

Representing relations applying to each member of a collection requires the use of a universal quantifier over the collection. When universally quantified variables and existentially quantified variables are mixed, their scope must be explicit. The RDF specifications do not tackle this issue. To solve it, the constructs proposed by [Berners-Lee, 1999] for universal and existential quantification can be re-used. This is what we do in the next examples of this article, when necessary. The next two examples illustrates the issue. The variable "g" refers to the set in the previous example.

```
E: Fred, Wilma and another man have each approved a certain resolution.
RDF: <rdf:Description aboutEach="#g">
      <approver of><Resolution></approver> <rdf:type resource="#Person"/>
    </rdf:Description>
```

```
E: Fred, Wilma and another person have each approved a resolution.
// (note: the persons may or may not have approved the same resolution)
RDF: <!-- if p is a person member of the set {Fred, Wilma, a man}
      then there exists a resolution r that has for approver p -->
    <forall var="p" about="#p">
      <if><rdf:type resource="#Person"/>
        <member of><Set><rdf:li resource="#Fred"/><rdf:li resource="#Wilma"/>
          <rdf:li><Man/><rdf:li>
        </Set></member>
        <then><exists var="r" about="#r">
          <rdf:type resource="#Resolution"/> <approver>#p</approver>
        </exists></then>
      </if></forall>
```

Often, one wants to say something about a certain subset of a collection. To permit this, many "set attributes" need to be added to RDF. For instance: "several", "many", "most", "mostly", "at most", "at least", "dozens", "hundreds", "thousands", "millions", "billions". Exact numbers and percentages should also be allowed. The next example illustrates the use of such attributes.

```
E: At least 3 persons, including Fred, approved most of the resolutions.
RDF: <Set ID="p" at_least="3"><rdf:li resource="#Fred"/></Set>
    <rdf:Description aboutEach="#p">
      <rdf:type resource="Person"/>
      <approver of><Set resource="r" most /></approver>
      <!-- the collection "r" is supposed to have been declared earlier -->
```

```
</rdf:Description>
```

Finally, the next two examples show how intervals can be represented using XOR-sets or normal sets.

```
E: Tom ran between 14 min and 15 min.
RDF: <rdf:Alt ID="i" at_least="14" at_most="15"/>
<rdf:Description aboutEach="#i">
  <rdf:type resource="Minute"/>
  <duration of><Run><agent resource="#Tom"/></Run>
</rdf:Description>

E: Tom ran between 14.00 and 15.00 (2pm to 3pm).
RDF: <Set ID="i" at_least="14" at_most="15"/>
<rdf:Description aboutEach="#i">
  <rdf:type resource="Hour__time_of_day"/>
  <duration of><Run><agent resource="#Tom"/></Run>
</rdf:Description>
```

3.4. Universal quantifiers and type definitions

The constructions (or keywords) for quantifying over the members of a collection may be reused to quantify over the instances of a type. Restrictions may be made via keywords or additional relations on the instances as shown by the following example.

```
E: At least 2% of persons like most of cats.
RDF: <forall at_least percent="2% var="p" about="#p">
  <if><rdf:type resource="#Person"/>
  <then><forall most var="c" about="#c">
    <if><rdf:type resource="#Cat"/>
    <then><object of><Liking><agent>#p</agent></Liking>
    </object of></then>
  </if></forall>
</then></if></forall>
```

These constructions may be viewed as type definitions, though in a format probably difficult to exploit. RDF also permits one to directly connect relations to a concept representing a type. This can be seen as a shortcut for representing relations "necessarily" connected to all instances of the type. For example, a generalization relation from a type t1 to a type t2, indicates that if an object is an instance of t1 then it is necessarily also an instance of t2.

```
E: A minivan is a van and a passenger vehicle.
RDF: <rdfs:Class ID="MiniVan"><rdfs:subClassOf resource="#Van"/>
  <rdfs:subClassOf resource="#PassengerVehicle"/>
</rdfs:Class>

E: Motorized and unmotorized vehicles are exclusive kinds of vehicles.
RDF: <Set><rdf:li><rdfs:Class ID="Motorized_vehicle"/></rdf:li>
  <rdf:li><rdfs:Class ID="Unmotorized_vehicle"/></rdf:li> </Set>
<rdf:Description about="#p" aboutEach="#p">
  <partition of><rdfs:Class ID="Vehicle"/></partition> </rdf:Description>

E: The relation Age relates a person to an integer
  and is a physical characteristic relationship.
RDF: <rdf:Property ID="age">
  <rdfs:domain resource="#Person"/>
  <rdfs:range resource="http://www.datatypes.org/useful_types#Integer"/>
  <rdfs:subPropertyOf resource="#PhysChrc"/>
</rdf:Property>
```

However, except in these three well-known special cases (generalization relations, exclusion relations and relation signatures), and unless additional conventions are adopted, this method should be avoided because it does not explicitly state how the instances are universally quantified. The following example illustrates the problem: does "any airplane for part a wing" or "any wing is part of a plane", or "a wing is part of any plane", etc.?

```
<rdfs:Class ID="#Plane"><part resource="#Wing"/></rdfs:Class>
```

5. Conclusion

Information can be represented in a number of different ways, especially with low-level general languages such as RDF/XML. For representations to be automatically comparable, conventions must be followed. We have proposed general lexical, structural and semantic conventions, then examined some issues associated to the most common logical

cases and proposed ways to use RDF in those cases. More issues need to be tackled and incorporated into the RDF specifications before this language can support knowledge reuse.

Acknowledgments

This work is supported by a research grant from the Australian Defense, Science and Technology Organisation (DSTO).

References

- T. Berners-Lee, The Semantic Toolbox: Building Semantics on top of XML-RDF, W3C Note, 24 May 1999.
<http://www.w3.org/DesignIssues/Toolbox.html>; see also the "Semantic Web Road map" at
<http://www.w3.org/DesignIssues/Semantic.html>
- T. Berners-Lee, D. Connolly, R. Swick., Web Architecture: Describing and Exchanging Data, W3C Note, 7 June 1999.
<http://www.w3.org/1999/04/WebData>
- N. Guarino, C. Masolo and G. Vetere, Ontoseek: Content-based Access to the Web, IEEE Intelligent Systems, Vol. 14, No. 3, pp. 70-80, May/June 1999
- Martin Ph. & Eklund P., Embedding Knowledge in Web Documents. Proceedings of WWW8, Eighth International World Wide Web Conference, special issue of The International Journal of Computer and Telecommunications Networking (in press), Toronto, Canada, May 11-14, 1999.
- BRML Business Rules Markup Language. <http://www.oasis-open.org/cover/brml.html>
- CGs Conceptual Graphs. <http://concept.cs.uah.edu/CG/Standard.html>. See also: J.F. Sowa, Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley, 1984.
- CYC <http://www.cyc.com/>
- DC Dublin Core. <http://purl.oclc.org/dc/>
- LOOM The LOOM knowledge representation system. <http://www.isi.edu/isd/LOOM/LOOM-HOME.html>. See also <http://www.isi.edu/isd/OntoLoom/hpkb/OntoLoom.html#RTFToC18>
- MCF/XML Meta Content Framework Using XML. <http://www.w3.org/TR/NOTE-MCF-XML/#secA>.
- RDF Resource Description Framework. <http://www.w3.org/RDF/>
- RDF syntax RDF Model and Syntax Specification. <http://www.w3.org/TR/REC-rdf-syntax/>
- RDF schema RDF Schema Specification. <http://www.w3.org/TR/1998/WD-rdf-schema/>
- WN WordNet. <http://www.cogsci.princeton.edu/~wn/>