Jena: Implementing the RDF Model and Syntax Specification

Brian McBride
Hewlett Packard Laboratories
Filton Road, Stoke Gifford
Bristol, UK
+44 117 312 9560

brian_mcbride@hp.com

ABSTRACT

Some aspects of W3C's RDF Model and Syntax Specification require careful reading and interpretation to produce a conformant implementation. Issues have arisen around anonymous resources, reification and RDF Graphs. These and other issues are identified, discussed and an interpretation of each is proposed. Jena, an RDF API in Java based on this interpretation, is described.

Keywords

RDF, XML

1. INTRODUCTION

Since the W3C's Resource Description Framework (RDF) Model and Syntax specification [1] completed its path to W3C recommendation several implementations have been developed. These differ in some aspects of their interpretation of the specification. There has been much discussion of these issues on the RDF Interest Mailing List [2] [3] [4], which so far, has not produced resolution. Inter-mixed with those discussions, have been others about changes and extensions to the specification.

All this has caused confusion and uncertainty that is inhibiting the acceptance and deployment of RDF. Tool builders wish to build tools that are correct and conformant. This they cannot do, because it is not clear what it means to be correct and conformant. Similarly producers and consumers of RDF wish to produce RDF whose interpretation is well defined. Uncertainty of interpretation inhibits them from doing so.

One reason for the lack of resolution is that issues are discussed individually. The issues themselves however, are interlinked. It is hard for a community discussing, say the subtleties of reification to agree when the have fundamentally different views on the nature of resources and their identification.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission by the authors.
Semantic Web Workshop 2001 Hong Kong, China
Copyright by the authors.

An implementer setting out to develop an implementation of an RDF tool must have an interpretation of the specification. This paper describes the interpretation developed for Jena [5], an RDF API in Java. The guiding principle for this interpretation was to implement, as far as possible, the specification as it is, without embellishment. It is documented here in the hope it will prove helpful to other developers.

Only issues concerning the RDF data model are discussed here; issues of RDF XML syntax are not considered.

2. INTERPRETING THE RDF MODEL AND SYNTAX SPECIFICATION

The RDF Model and Syntax specification defines an abstract data model. The model is abstract because it is defined in terms of abstract mathematical structures such as triples and sets. It is a data model only, because no formal semantics is given. It is suggested that RDF statements represent facts, but nothing formal is defined. Others [6] [7] have offered formal interpretations defined in terms of first order predicate logic.

The Model and Syntax specification also defines how to represent data conforming to this data model in XML. The XML serialization is a *representation* of the abstract model. Other representations are also possible. For example, an RDF graph may be represented by a data structure in computer memory or tables in a relational database. This structure is represented in figure 1.

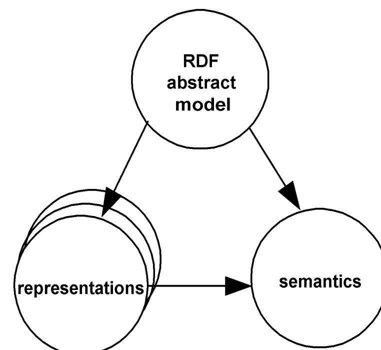

Figure 1

It is important, as will be seen below, to distinguish between the abstract data model and its representations. The specifications

define constraints which apply to the abstract data model. The abstract model is infinite; representations of the abstract data model must be finite and incomplete. The Model and Syntax specification defines no formal semantics for RDF.

2.1 Resources and URI's

RFC 2396 [8] defines a resource to be a conceptual mapping:

> *The resource is the conceptual mapping to an entity or set of entities, not necessarily the entity which corresponds to that mapping at any particular instance in time. Thus, a resource can remain constant even when its content ---the entities to which it currently corresponds---changes over time, provided that the conceptual mapping is not changed in the process.*

For example, a resource, identified by a specific URI, may represent the W3C logo. When a browser uses HTTP to request a representation of that logo, the particular representation it receives may depend on a number of factors such as time (the logo may change over time) and the file format (jpg, gif or png representation) required. In this case, the URI identifies the abstract concept of the W3C logo. A particular representation, say the JPEG representation, may have its own different URI.

Can a resource have more than one URI? This is a question not just for RDF, but for web and internet architecture as a whole, which, at the time of writing, has not finally been resolved.

The RDF Model and Syntax specification, however, takes a position on this question. No provision is made in the RDF data model for a resource to have multiple URI's. Provision is made for a resource to have one URI. Other URI's could be associated with a resource through some property, but the RDF specifications define no such property. The implication is clear, that as far as RDF is concerned, resources have a distinguished URI.

Web principles [9], however, dictate that there can be no central authority to allocate URI's to conceptual mappings. There is no way to stop many individuals independently assigning URIs to represent, say, the trees in a park. Each such URI defines a new resource. Thus there may be many resources that represent the same tree. The RDF specifications do not define a mechanism for stating the equivalence of resources, i.e. that multiple resources represent the same conceptual mapping. This is left to higher layers of the stack such as DAML-ONT [10].

2.2 Anonymous Resources

The Model and Syntax specification is unclear about anonymous resources. In section 2.1 it states:

> *Resources are always named by URIs plus optional anchor ids*

However, in figure 2 of the specification and its preceding text, it introduces the concept of an anonymous resource, that is a resource that does not have a URI, and subsequently refers to such resources in three places in section 6.

The repeated references to anonymous resources indicate clearly the intention of the authors that an RDF graph should be able to represent a resource without representing its URI. This can be reconciled with the statement quoted above if it is interpreted as meaning that whilst a resource must always have a URI, that is a constraint that applies to the abstract model. A particular representation of a resource need not include the URI.

An alternative interpretation, that all representations of RDF must have a URI for each resource is inconsistent with the rest of the Model and Syntax specification, seems draconian and is not enforceable.

Anonymous resources can be thought of as existentially qualified variables. The graph in figure 2 shows an anonymous resource with a number of properties. This graph can be thought of as stating that Ora created a specification, whose URI is not represented, called "RDF M&S".

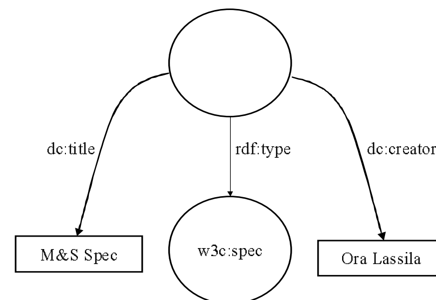

Figure 2

Applications creating RDF models are not required to supply a URI for all resources. In particular, RDF XML parsers should distinguish between resources for which a URI was encoded in the serialization and those that were anonymous. Parsers that fail to do so, prevent an application from 'round tripping', i.e. an application is unable to write an RDF graph to a file and recreate the same graph when the XML serialization is read back in.

It is unfortunate that the XML serialization defined for RDF does not permit the representation of all possible graphs containing anonymous resources.

2.3 Properties

Properties are resources that are identified by URI's. In an XML serialization of RDF, properties are often represented by XML QNames of the form nsprefix:LocalPart, in which case the URI of the property is the concatenation of the URI associated with the nsprefix and the LocalPart of the QName.

Care is needed interpreting what the Model and Syntax specification says about the relationship between properties and namespaces. Section 2.2.3. states:

> *In RDF, each predicate used in a statement must be identified with exactly one namespace, or schema.*

In section 6 it states:

> *It is recommended that property names always be qualified with a namespace prefix to unambiguously connect the property definition with the corresponding schema.*

Two issues arise with these statements:

- the second statement seems to undermine the first, in that it merely recommends that properties be connected with a namespace, whilst the former requires it.

- the first statement suggests that it is the use of a property that is associated with a namespace whilst the latter suggests it is the property that is associated with a namespace.

The first issue is resolved by taking the first statement as definitive. The second statement is explained by the fact that it is not possible for an RDF processor, given the URI of a property, to always determine unambiguously the namespace with which it is associated. Given a property with URI http://foo/bar, it is not possible algorithmically to determine whether the namespace is http://foo/ or http://foo/b or http://foo/ba. All are possible. The usual algorithm employed by processors is to search back from the end of URI for the first character that cannot appear in the LocalPart of an XML QName. This, however, is not guaranteed to be correct. The second statement therefore is an admonition to the creators of XML representations of RDF to remove this ambiguity by specifying the namespace explicitly.

RDF XML parsers and other RDF processors should retain this information, representing properties not just by their URI, but by the pair consisting of their namespace URI and LocalPart. This will enable them to acquire and process the RDF Schema [11] that describes each property and to write correctly an RDF graph as XML.

The second issue is that the first statement quoted above, allows an interpretation in which the property identified by http://foo/bar could be associated with the namespace http://foo/ in one statement and the namespace http:/foo/b in another. This would imply that a property, identified by a particular URI could have multiple interpretations. RDF Schema would be undermined by this interpretation, as it would not be possible, when asserting say a domain or range constraint on a property, to specify to which interpretation of the property, the constraint applied. This interpretation is therefore rejected.

2.4 Literals

Though the Model and Syntax specification is clear, the nature of literals is commonly misunderstood. A literal is not just a string of characters, but also optionally encodes a language identifier. This language identifier is part of the value of the literal and must be represented by implementations.

2.5 Statements

An RDF statement is defined to be a triple consisting of a predicate, a subject and an object. A triple is a mathematical structure that is uniquely defined by its three components. Thus, there can be only one statement with a given subject, predicate and object. There can be many representations of a single triple, e.g. in multiple XML files, databases or computer memories, but those are representations of a triple, not the triple itself.

The subject of a statement is defined to be a resource. The subject of a statement is not the URI of a resource, it is the resource itself. Representations of statements typically use URI's as part of the representation of a resource, but it is important to understand that the representations are not the same thing as the actual statements and resources.

Whilst section 5 of the Model and Syntax specification, the formal model for RDF, does not explicitly say so, the set of resources and the set of literals are disjoint. If *literal* is the literal "http://foo" and *resource* is the resource whose URI is http://foo, then the statement *(predicate, subject, literal)* is not the same statement as *(predicate, subject, resource)*. Implementations therefore, cannot use just the URI or the literal string to represent a resource or literal; they must have some way of distinguishing the two.

2.6 Reified Statements

RDF statements are not resources. Through a mechanism known as reification, there are resources that represent RDF statements. The Model and Syntax specification (in section 5, rule 9) defines the reification of an RDF statement to be a resource r which represents the statement along with four statements, one which defines the type of the resource to be an RDF statement, and three others which describe the subject, the predicate and the object of the statement. The *reification* of a statement is thus a small RDF graph containing these four statements.

Section 5 goes on to state:

> *The resource r in the definition above is called the reified statement. When a resource represents a reified statement; that is, it has an RDF:type property with a value of RDF:Statement, then that resource must have exactly one RDF:subject property, one RDF:object property, and one RDF:predicate property.*

The language here is rather loose. The phrase "When a resource represents a reified statement" should be read as "When a resource is a reified statement" to be consistent with the first sentence of the paragraph.

Thus a *reified statement* is the single resource that represents a statement.

The paragraph quoted above applies to the RDF abstract data model. In the abstract data model, every reified statement does have all four properties. A representation may represent only part of the abstract data model, and so need not include all the properties.

As with trees in the park, or any other object or concept, there is nothing to preclude statements being given multiple URIs. Thus, whilst there can only be one statement with a given subject, predicate and object, there may be many reified statements representing that statement. Since each such reified statement represents the same statement, the simplest semantics for RDF implies that any property of one is a property of them all.

2.7 Statements, Statings and Occurrences

An RDF statement is defined to be a triple of the form (predicate, subject, object). The need of some applications to represent occurrences of statements has been identified. For example, an application may wish to represent the fact that a particular statement occurred in a particular document at a

particular time. Occurrences of statements are often called 'statings'.

The term "occurrences" is preferred to "statings". Has a statement that occurs in a collection of fallacies been stated? It certainly occurs in that collection, but it is not clear that it has been stated.

The Model and Syntax specification states that a reified statement represents a statement. For example, in section 4.1 para 6:

> *A new resource with the above four properties represents the original statement...*

Despite this, there has been a suggestion in the RDF community that reified statements represent occurrences of statements. This can only be consistent with the Model and Syntax specification if a resource can represent both a statement and an occurrence of a statement. For any such resource, it is easy to construct a contradiction.

Consider a statement S that occurs in two documents http://foo and http://bar. Let RS be a reified statement representing both S and its occurrence in http://foo. Then the statement (occursIn, RS, http://foo) is true. Is the statement (occursIn, RS, http://bar) true? It is true of the statement S, but it is not true of the occurrence of S in http://foo. So this statement is both true and false of RS, a contradiction.

Thus reified statements represent statements, not occurrences of statements or statings.

2.8 RDF Graphs

The Model and Syntax specification refers to the concept of an RDF graph, i.e. a specific collection of RDF statements, but omits this concept from the formal model. Implementations deal with specific collections of statements and generally implement the concept of a graph, though it is frequently called a model.

There is a need to name with a URI, a specific collection of RDF statements. For example, RDF Schema is represented by a specific collection of RDF statements. Accessing the URI of RDF Schema will return an XML representation of that collection of statements. Implementations must manipulate specific named collections of statements. There is also a need to make statements about specific collections of statements, e.g. to state that the title of the collection of statements representing RDF Schema is "RDF Schema".

Since the RDF Model and Syntax Specification does not provide any formal language for graphs, some is suggested here.

A collection of RDF statements is known as an RDF graph. So that RDF may be used to describe an RDF graph, a graph may be represented by a resource. The reification of an RDF graph G consists of a resource g of type rdf:Bag together with a set of statements S of the form (rdf:_n, g, RSn) for n = 1 to the number of statements in G. For each statement s in G, there is an element of S with RSn = a reified statement representing s. g is known as a reified graph, or alternatively. It is permitted to represent a partial reification of a graph or model.

Is a graph a set of statements, i.e. each statement may appear only once in a graph, or is it a bag? The specification does not say and implementers are divided on this question. An informal poll of implementers had a majority implementing a graph as a set of statements.

The suggested interpretation of an RDF statement is as a fact. There is little point in including the same fact in a collection more than once. When graphs are merged, it is wasteful if statements that occur in more than one of the source graphs occur more than once in the resulting graphs. For this interpretation, a graph is a set of statements.

3. THE JENA RDF API

Jena is an API in the Java programming language, for the creation and manipulation of RDF graphs. It implements the interpretation of the RDF specifications described in section 2 above.

Jena was developed to satisfy two goals:

- to provide an API that was easier for the programmer to use than alternative implementations
- to be conformant to the RDF specifications

An open source implementation of the Jena API is available from:

http://www-uk.hpl.hp.com/people/bwm/rdf/jena

3.1 API Features

The Jena API is designed specifically for the Java programming language. API's can be programming language neutral; sometimes, like the document Object Model (DOM) API [12], defined using an interface definition language (IDL). A language binding can then be defined for any given programming language. This approach prohibits an API from exploiting the features of a specific programming language. The alternative approach, as exemplified by JDom [13], is to define an API that takes advantage of the features of a specific programming language and environment. Jena adopts the latter approach.

Previous RDF API's had adopted either a statement centric or a resource centric approach. In the statement centric approach, as implemented by SiRPAC [14], method calls are defined in terms of statements, which reflects the underlying implementation of an RDF graph as a collection of triples. Applications, however, are often more conveniently written in terms of resources and their properties, as in DATAX [15].

Jena integrates both programming styles into a single API. Applications can be written using a statement centric approach, a resource centric approach or a mixture of both. For example[1]:

```
Resource res = model.createResource();
model.addStatement(res, RDF.type, RDFS.Class);
model.addStatement(res, RDFS.label, "example");
model.addStatement(res, RDFS.comment, "…");
```

may also be written as:

```
model.createResource()
     .addProperty(RDF.type, RDFS.Class)
```

[1] Jena uses the term ‘model’ for an RDF graph.

```
        .addProperty(RDFS.label, "example")
        .addProperty(RDFS.comment, "…");
```

The RDF data model supports only string values in literals, whereas applications often need to represent integers, floats or application defined types. Jena provides convenience methods for the automatic conversion of both Java built in and application defined types to and from property values, e.g.:

```
r.addProperty(RDF.value, 5.5);
r.addProperty(FOO.date, myDate);
Double d =
    r.getProperty(RDF.value).getDouble();
```

Resources may be sub-classed to provide behaviour, a feature that is used to provide specific support for RDF containers. Subclasses of Resource implement general container behaviour and specific behaviour for BAGs, SEQs and ALTs. For example:

```
bag.remove("value");
seq.add(index, "value");
```

The first call will delete the appropriate value from the bag. The second will insert a new value into a sequence, and again renumber other members as needed.

A flexible query API is provided. All the query methods take a selector object as an argument. By defining new selector classes, new query languages can be added without disturbing the core API. A query on a graph may return either a new graph which is a sub-graph of the original, an iterator which will return all the statements matching the query or a table of values (represented as a JDBC ResultSet) matching variables in the query.

3.2 Implementing the Interpretation

The Jena API implements anonymous resources, i.e. resources need not have a known URI. The implementation tracks internally, the identity of resources, so it is able to determine when two anonymous resources are in fact the same resource. RDFFilter [16], the RDF XML parser that is integrated into Jena, does not create URI's (so called genid's) for anonymous resources as it parses.

Properties in Jena have an associated namespace. Property objects can be queried to determine that namespace. When a property object is constructed, either a namespace must be provided as an argument, or the implementation will attempt to determine the name space URI by splitting the property URI at the last character that is illegal in the LocalName part of an XML QName. The parser integrated into Jena retains the structure of the QName from the XML serialization and constructs property objects with the correct name space.

Literals in Jena have an associated language encoding. Literals are not equal unless their language encodings are equal.

RDF graphs are implemented as sets of statements. Adding a statement that is already present to a graph will have no effect.

Statements are implemented as a sub-class of resource. Whilst in the formal model statements are not resources, it is convenient in an API to be able to represent use a statement to represent its reified statement. For example, to add to a model the fact that the statement (RDF:value, res, "value") occurs in http://foo:

```
m.createStatement(res, RDF.value, "…")
 .addProperty(FOO.occursIn, "http://foo");
```

The Jena triple store uses a statement object to represent the reification of a statement. The presence of a statement object, as either the subject or object of a statement in a graph is equivalent to representing the four triples of the reification of the statement explicitly in the graph. This permits efficient representation of reification.

3.3 Jena API Implementation Architecture

The structure of the Jena implementation is shown in figure 3.

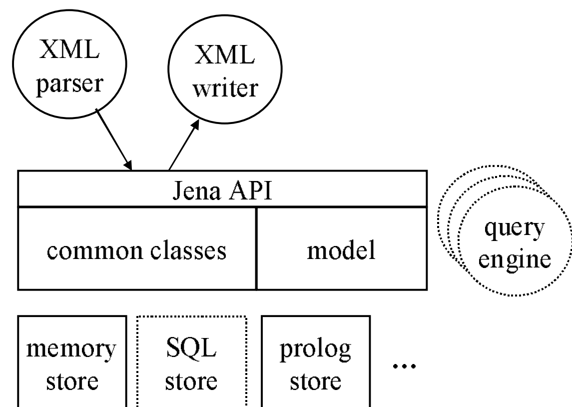

Figure 3

The implementation has been designed to permit the easy integration of alternative processing modules such as parsers, serializers, stores and query processors.

The API itself consists of a collection of Java interfaces representing resources, properties, literals, containers, statements and models. A common set of classes implement these interfaces, though these may be sub-classed or replaced to optimize particular implementations. The model class is a generic implementation of an RDF graph. A standard interface connects model to classes that implement storage and basic querying of RDF statements. A standard interface also enables integration of specialized query processors.

4. CONCLUSIONS

This paper has discussed a number of issues in the interpretation of the RDF Model and Syntax specification that implementers of RDF tools must address. A resolution of those issues, consistent with the specification as written has been described. Jena, a Java API for RDF, and its implementation is also described.

Acknowledgements

The dedicated community of the RDF Interest Group have greatly helped my understanding of the RDF specifications, as have many discussions with my friend and colleague, Stuart Williams. The motivation to develop the Jena API came originally from Ian Dickinson. The flexible query interface was suggested by Gabe Beged-Dov. Returning JDBC ResultSet's from queries was suggested by Dan Brickley. I am greatly indebted to Dave Reynolds for his support and encouragement.

References

1. O. Lasilla, R. Swick (eds): Resource Description Framework (RDF) Model and Syntax Specification, http://www.w3.org/TR/REC-rdf-syntax/.

2. Discussion Archive for the RDF Interest Group, http://lists.w3.org/Archives/Public/www-rdf-interest/

3. RDF Interest Group - Issue Tracking, http://www.w3.org/2000/03/rdf-tracking/

4. B. McBride, Issues Raised in the RDF Interest Group Mailing List, http://www-uk.hpl.hp.com/people/bwm/rdf/issues.htm

5. B. McBride, Jena, An RDF API in Java, http://www-uk.hpl.hp.com/people/bwm/rdf/jena

6. Wolfram Conen, Reinhold Klapsing: A Logical Interpretation of RDF, http://nestroy.wi-inf.uni-essen.de/rdf/logical_interpretation/index.html

7. Richard Fikes, Deborah L McGuinness, An Axiomatic Semantics for RDF, RDF Schema, and DAML-ONT, http://www.ksl.stanford.edu/people/dlm/daml-semantics/

8. T. Berners-Lee, R. Fielding, U. C. Irvine, L. Masinter, RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax, http://www.ietf.org/rfc/rfc2396.txt?number=2396

9. Tim Berners-Lee, Web Architecture from 50,000 feet, http://www.w3.org/DesignIssues/Architecture.html

10. Lynn Stein, Dan Connolly, Deborah McGuinness (eds), DAML-ONT Initial Release, http://www.daml.org/2000/10/daml-ont.html

11. Dan Brickley, R. V. Guha (eds), Resource Description Framework (RDF) Schema Specification 1.0, http://www.w3.org/TR/2000/CR-rdf-schema-20000327/

12. Arnaud le Hors et al (eds), Document Object Model (DOM) Level 2 Core Specification, http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/

13. JDom, http://www.jdom.org/

14. Janne Saaarela, Art Barstow, Sergey Melnick, Dan Brickley, SiRPAC - A Simple RDF Parser and Compiler, http://www.w3.org/RDF/Implementations/SiRPAC/

15. David Megginson, DATAX: Data Exchange in XML, http://www.megginson.com/DATAX/index.html

16. David Megginson, RDF Filter, http://www.megginson.com/Software/