Query + Metadata + Logic = Metalog

Massimo Marchiori, **Janne Saarela**
{massimo,jsaarela}@w3.org

The World Wide Web Consortium (W3C)

The Resource Description Framework (RDF) Model&Syntax Specification describes a metadata infrastructure which can accommodate classification elements from different vocabularies i.e. schemas. The underlying model consists of a labeled directed acyclic graph which can be linearized into eXtensible Markup Language (XML) transfer syntax for interchange between applications.

This paper will demonstrate how a new querying language, Metalog, allows users to write inference rules and queries in English-like syntax. We will demonstrate how these reasoning rules have equivalent representation both as RDF descriptions and as logic programs. We will also show how an automated compilation between these translations is possible.

For the sake of clarity, here we will just give an overview of the system, trying to avoid technicalities and cumbersome details.

Query Languages

In general, query languages are formal languages to retrieve data from a database. Standardadized languages already exist to retrieve information from different types of databases such as Structured Query Language (SQL) for relational databases and Object Query Language (OQL) and SQL3 for object databases.

Semi-structure query languages such as XML-QL [3] operate on the document level structure.

Logic programs consist of facts and rules where valid inference rules are used to determine all the facts that apply within a given model.

With RDF, the most suitable approach is to focus on the underlying data model. Even though XML-QL could be used to query RDF descriptions in their XML encoded form, a single RDF data model could not be correctly determined with a single XML-QL query due to the fact that RDF allows several XML syntax encodings for the same data model.

The Metalog Approach

RDF provides the basis for structuring the data present in the web in a consistent and accurate way. However, RDF is only the first step towards the construction of what Tim Berners-Lee calls the "web of knowledge", a World Wide Web where data is structured, *and* users can fully benefit by this structure when accessing information on the web. RDF only provides the "basic vocabulary" in which data can be expressed and structured. Then, the whole problem of *accessing* an *managing* these data structured arises.

Metalog provides a "logical" view of metadata present on the web. The Metalog approach is composed by several components.

In the first component, a particular data semantics is established. Metalog provides way to express logical relationships like "and", "or" and so on, and to build up complex *inference rules* that encode logical reasoning. This "semantic layer" builds on top of RDF using a so-called *RDF schema*.

The second component consists of a "logical interpretation" of RDF data (optionally enriched with the semantic schema) into logic programming. This way, the understood semantics of RDF is unwielded into its logical components (a logic program, indeed). This means that every reasonment on RDF data can be performed acting upon the corresponding logical view, the logic program, providing a neat and powerful way to reason about data.

The third component is a language interface to writing structured data and reasoning rules. In principle, the first component already suffices: data and rules can be written directly in RDF, using RDF syntax and the metalog schema. However, this is not convenient from the practical viewpoint. Indeed, RDF syntax aims at being more an encoding language rather than a user-friendly language, and it is well recognised in the RDF community and among vendors that the typical applications will provide more user-friendly interfaces between the "raw RDF" code and the user. Our proposed language is innovative in that it tries to stress user-friendliness as much as possible: a program is a collection of *natural language* assertions. We think this feature will be particularly important for the wide deployment not only of metalog, but of RDF itself: the measure of the success of metadata and proper structuring of information on the web is given by the number of people that will actually lose time and energy in write (and/or translate) data into the structured format. Therefore, it is of primary importance that the entry level is kept extremely easy, to avoid that the difficuly of just learning how to encode and structure data will just block the widespread diffusion of metadata in the web.

Another important feature of the language, in this respect, is indeed that it can be used just as an interface to RDF, without the metalog extensions. This way, users will be able to access and structure metadata using RDF in a smooth and seamless way, using the metalog language.

The Metalog Schema

The first correspondance in Metalog is between the basic RDF data model and the predicates in logic. The RDF data model consists of so-called *statements* Statements are triples where there is a subject (the "resource"), a predicate (the "property"), and an object (the "literal"). Metalog views an RDF statement in the logical setting as just a binary predicate involving the subject and the literal. For example, the RDF statement expressing the fact that *Tim Berners-Lee invented the Web* (formally, the RDF triple *{invented, Tim Berners-Lee, Web}*) is seen in logic programming as the predicate *invented(Tim Berners-Lee, Web)*.

Once estalished the basic correspondance between the basic RDF data model and predicates in logic, the next step comes easy: we can extend RDF so that the mapping to logic is able to take advantage of all of the logical relationships present in logical systems: that is to say, behind the ability of expresing *static facts*, we want the ability to encode *dynamic reasoning rules,* like in logic programming.

In order to do so, we need at least:

- the standard logical connectors (*and*, *or*, *not*)

- variables

The metalog schema extends plain RDF with this "logical layer", enabling to express arbitrary logical relationships within RDF. In fact, the metalog schema provides more accessories besides the aforementioned basic ones (like for example, the "implies" connector): anyway, not to heaven the discussion, we don't go into further details on this topic. What the reader should keep in mind is just that the Metalog schema provides the "meta-logic" operators to reason with RDF statements.

Technically, this is quite easy to do: the metalog schema is just a schema as defined by the RDF schema specification where, for example, *and* and *or* are subinstances of the RDF *Bag* connector.

The mapping between "metalog RDF" and logical formulas is then completely natural: for each RDF statement that does not use a metalog connector, there is a corresponding logical predicate as defined before. Then, the metalog connectors are translated into the corresponding logical connectors in the natural way (so, for instance, the metalog *and* connector is mapped using logical conjunction, while the metalog *or* connector is mapped using logical disjunction).

The Metalog Syntax

Note that the RDF metalog schema and the corresponding translation into logical formulas is absolutely general. However, in practicse, one need also to then be able to process the resulting logical formulas in an effective ways. In other words, while the RDF metalog schema nicely extends RDF with the full power of first order predicate calculus, thus increasing by far the *expressibility* of basic RDF, there is still the other, *computational*, side of the coin: how to process and effectively reason with all these logical inference rules.

It is well known that in general dealing with full first order predicate calculus is totally unfeasable computationally. So, what we would like to have is a subset of predicate calculus that is still expressible enough, and also computationally feasible: our choice went to *logic programming*. Logic programming (see e.g. [1]) is a well known programming paradigm that selects a subset of full first-order predicate calculus (so called Horn clauses); it is a very powerful and expressive paradigm, and has the further advantage that it has been widely studied in the database community (a subset of logic programming, **datalog**, has even the advantage of having computations always terminating, a feature of obvious interest for web queries).

The third level is then the actual syntax interface between the user and this "metalog RDF" encoding, with the constraint that the expressibility of the language must fit within the one provided by logic programming.

The metalog syntax has been explicitly designed with the purpose of being totally natural-language based, trying to avoid any possible technicalities, and therefore making the language extrememly readable and self-descriptive.

The way metalog reaches this scope is by a careful use of upper/lower case, quotes, and by allowing a rather liberal positioning of the keywords (an advanced parser then disambiguates the keywords from each metalog program line).

Upper/lower case is used to distingush between normal keywords and variables: variables are expressed using names all in upper case (for example, `FOO` is a variable). Words that are in lower case either are keywords (reserved words), or if not, they are ignored. For example, `then` is a keyword, while `foo` is not, and so it is just ignored (it is only syntactic sugaring). Other words can be either keywords, or they are just

ignored. In the current version of metalog, words cannot intermingle upper and lower case: this helps to reduce errors and to improve readability, since it strengthens the layout difference between variables and the other words.

Finally, any name which is between double quotes (for example, "John") is a datum (a fixed constant).

Keywords

The following set of keywords are reserved in metalog. Interpretaion of the keywords is dne in metalog on a positional basis: the position of the keyword with respect to other keywords and/or other data determines the interpretation of the sentence. The reserved keywords are:

- `then` is a keyword for the logical implication (=>)
  For example,
  `if SHE has a "degree" in "math" then SHE "is" "smart"`
  is translated into the logical formula
  degree(SHE,"math") => is(SHE,"smart")
  (Note that, here and in the following examples, we provide directly the translation into the logical formula, to save space; a more detailed translation would have to also show the intermediate RDF model, which is in any case trivial to derive).
- `imply` is a keyword for the logical implication (=>)
- `implies` is a keyword for the logical implication (=>)
- and can be either the metalog *and*, or it can be used to indicate the presence of an RDF Bag: this is disambiguated by the context
  For example,
  `if SHE has a "degree" in "math" and SHE has a "degree" in "computer science" as well then SHE "is" "really smart".`
  is translated into the logical formula
  (*degree(SHE,"math") and degree(SHE,"computer science"))=> is(SHE,"really smart")*
  On the other hand, as said, and can be used to denote an RDF Bag (a set):
  `the "technical report 231" has as "authors" "Mary" and "John".`
  is translated into the logical formulas (the translation here is more involved since the RDF Bag construct is used):
  ```
  authors("technical report 231",foo).
  rdf:type(0,rdf:Bag).
  rdf:_1(0,"Mary").
  rdf:_2(0,"John").
  ```
- `or` can be either the metalog *or*, or it can be used to indicate the presence of an RDF Alt (an alternatives list) : this is disambiguated by the context
- `order` the presence of this keyword turns an RDF Bag into an RDF Seq (an ordered list).
  For example,
  `the "technical report 231" has as "authors" "Mary" and "John" in this order.`
  is translated into the logical formulas (the translation here is more involved since the RDF Bag construct is used):
  ```
  authors("technical report 231",foo).
  rdf:type(foo,rdf:Seq).
  ```

```
rdf:_1(foo,"Mary").
rdf:_2(foo,"John").
```

- `not` can be combined with any other metalog constructs, and its interpretation is logical negation
  For example
  `if SHE has a "degree" in "math" then SHE "is" not "stupid".`
  is translated into the logical formula
  degree(SHE,"math") => not(is(SHE,"stupid"))

The dot is the separator between metalog program lines. For formatting purposes, carriage returns, line feeds can tabs can be used: they are simply ignored. Similarly, commas and semicolon can be used as well.

A trailing question mark is used to denote a query.

Note: metalog programs also have a facility to express namespaces via the keyword `namespace`. We will not go in further details since we won't be explicitly using namespaces sugaring here, but en passant we just mention that essentially the `namespace` keyword has the same functionality as the `xmlns` attribute for XML namespaces. Also, there are a number of other keywords that deal. for example, with numbers operations (e.g., `greater`, `less`, etc.), but for the sake of brevity we don't go in their (rather obvious) description.

A more detailed example

Suppose we want to encode the rule that if a person has written a document in some language (for example, English), then he can speak in that language. The corresponding metalog program would be:

```
if the "language" of a DOCUMENT is Y
and the "author" of the DOCUMENT is X
then X can "speak" Y.
```

This can be translated into the following piece of RDF syntax:

```
<Procedure>
  <Head>
    <and>
      <Predicate name="speak">
         <rdf:Seq>
           <rdf:li><Variable>X</Variable></rdf:li>
           <rdf:li><Variable>Y</Variable></rdf:li>
         </rdf:Seq>
       </Predicate>
     </and>
  </Head>
  <Body>
  <and>
    <Predicates>
      <rdf:Seq>
        <rdf:li>
          <Predicate name="creator">
            <rdf:Seq>
```

```
              <rdf:li><Variable>DOCUMENT</Variable></rdf:li>
              <rdf:li><Variable>X</Variable></rdf:li>
            </rdf:Seq>
          </Predicate>
        </rdf:li>
        <rdf:li>
      <Predicate name="language">
        <rdf:Seq>
          <rdf:li><Variable>DOCUMENT</Variable></rdf:li>
          <rdf:li><Variable>Y</Variable></rdf:li>
        </rdf:Seq>
      </Predicate>
    </rdf:li>
  </rdf:Seq>
  </Predicates>
  </and>
  </Body>
</Procedure>
```

And, finally, this corresponds to the logical formula

speak(X,Y) <= (author(DOCUMENT,X) and language(DOCUMENT,Y))

So, suppose we have already grabbed from somewhere in the web some pieces of RDF that tell us, for example, that "John" is the author of "technical report 231", and that the language of "technical report 231" is "English".
Then, if we want to know what language does John speak, we can just ask

```
what "language" does "John" "speak"?
```

which is translated into the corresponding query

speak("John",Y).

Running this query in the corresponding logic program gives the result that *Y="English"*, that is to say, the predicate *speak("John","English")* is true.
Hence, the corresponding metalog sentence, returned as answer, is:

```
"John" "speaks" "English"
```

As far as real data are concerned, among our examples we have run the above example using a set of 2700 RDF data model triples that correspond with the data available at the World Wide Web Consortium technical reports page. This page presents the public documents the consortium has published along with their authors, dates, and URIs. Therefore, one can get a complete knowledge basis regarding W3C's authors, that is flexible and elegantly extendable.

Related work

The use of Web infrastructure to accommodate logic programs has been suggested by (Sandevall, 1996) and (Loke & Davidson, 1996). The latter approach suggests using familiar logic program notation to place facts and queries on HTML pages. The embedded rules also have the ability to refer to other HTML pages with other predicates using a namespace mechanism. In this way, their evaluation context increases over the amount of HTML pages they retrieve to find facts that satisfy the queries.

Conclusions

To the best of our knowledge, this is the first work that addresses the problem of querying RDF models and extending it with the ability of expressing reasoning rules. The metalog model that we have sketched is general enough to be of wide use, and powerful enough to fulfill most of the generic user's needs. Moreover, it is elegantly integrated within the "big picture" of W3C's standards, with a particular eye geared toward extendability and future improvements. It tries to lower the "access level" to metadata and reasoning management by using a top-level syntax using natural language, enabling not only easy and fast writing of complex relationships, but also an extremely high readability. Finally, it can be used even without the logical extensions, just to provide a user-friendly interface to RDF. Future work that we plan to do within W3C is the deployment of a publicly accessible prototype of the system, so to foster on a large scale use of structured metadata on the web.

Acknowledgements

The authors would like to thank Bert Bos for his help in running the test sets.

References

1. Das, S.K. (1992). *Deductive Databases and Logic Programming*. Addison Wesley.
2. Dan Brickley, R.V. Guha, A. Layman, "Resource Description Framework (RDF) Schema Specification". W3C Draft.
   http://www.w3.org/TR/1998/NOTE-xml-ql-19980819
3. Lassila, O., Swick, R. (1998). *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Working Draft.
   http://www.w3.org/TR
4. Loke, S.W., Davison, A. (1996). Logic Programming with the World Wide Web. *Proc. of the 7th ACM Conf. on Hypertext.*
   http://www.cs.unc.edu/~barman/HT96/P14/lpwww.html
5. Niemelä, Simons, P. (1997). Smodels -- an implementation of the stable model and well-founded semantics for normal logic programs *Proc. of the 4th Int. Conf. on Logic Programming and Non-Monotonic Reasoning*. Dagstuhl, Germany.
   http://saturn.hut.fi/pub/papers/lpnmr97-sd.ps.gz
6. Ramakrishnan, R., Srivastava, D., Sudarshan, D. (1992). CORAL: Control, Relations and Logic. *Proc. of the Int. Conf. on VLDB.*.
7. Sandewall, E. (1996). Towards a World-Wide Data Base. *Proc. of the 5th Int. WWW Conf.*.

Appendix A - Query schema in RDF

In the following, we provide (part of) the Metalog schema, to provide the technically oriented reader with more inside on how the schema effectively uses RDF schema facilities.

```
<RDF xmlns="http://www.w3.org/TR/WD-rdf-syntax#"
     xmlns:rdf="http://www.w3.org/TR/WD-rdf-syntax#"
     xmlns:rdfs="http://www.w3.org/TR/WD-rdf-schema#">
<rdfs:Class ID="Procedure" />

<Predicate ID="Head">
  <rdfs:comment xml:lang="en">Head of the procedure</rdfs:comment>
  <rdfs:domain rdf:resource="#Procedure"/>
  <rdfs:range rdf:resource="#Connector"/>
  <rdfs:range rdf:resource="#Predicate"/>
</Predicate>

<Predicate ID="Body">
  <rdfs:comment xml:lang="en">Body of the procedure</rdfs:comment>
  <rdfs:domain rdf:resource="#Procedure"/>
  <rdfs:range rdf:resource="#Connector"/>
  <rdfs:range rdf:resource="#Predicate"/>
</Predicate>

<Predicate ID="Predicates">
  <rdfs:comment xml:lang="en">Predicates combined with a connector</rdfs:comment
  <rdfs:domain rdf:resource="#Connector"/>
  <rdfs:range rdf:resource="#Predicate"/>
  <rdfs:range rdf:resource="#Connector"/>
  <!-- this last range definition enables recursion -->
</Predicate>

<rdfs:Class ID="Connector" />

<rdfs:Class ID="And">
  <rdfs:subClassOf rdf:resource="#Connector" />
</rdfs:Class>

<rdfs:Class ID="Or">
  <rdfs:subClassOf rdf:resource="#Connector" />
</rdfs:Class>

<rdfs:Class ID="Not">
  <rdfs:subClassOf rdf:resource="#Connector" />
</rdfs:Class>

<rdfs:Class ID="Predicate" />

<Predicate ID="Variable">
  <rdfs:comment xml:lang="en">Variable within a predicate</rdfs:comment>
  <rdfs:domain rdf:resource="#Predicate"/>
  <rdfs:range
rdf:resource="http://www.w3.org/FictionalSchemas/useful_types#String"/>
</Predicate>

<Predicate ID="Constant">
  <rdfs:comment xml:lang="en">Constant within a predicate</rdfs:comment>
  <rdfs:domain rdf:resource="#Predicate"/>
  <rdfs:range
rdf:resource="http://www.w3.org/FictionalSchemas/useful_types#String"/>
```

```
</Predicate>
</RDF>
```
