As XDuce is to XML, so ? is to RDF

Programming Languages for the Semantic Web

Alan Jeffrey Peter F. Patel-Schneider
Alcatel-Lucent Bell Labs
{ajeffrey,pfps}@bell-labs.com

Abstract. *In this position paper, we briefly introduce the foundations of the Semantic Web for a programming languages research audience, summarize recent work by the authors on validation, and discuss issues in programming languages for the Semantic Web.*

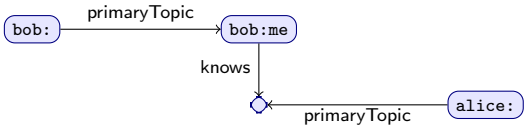
Introduction

The Semantic Web. The aim of the Semantic Web is to provide machine-readable representations of information which support mechanized reasoning. These representations describe a labeled graph of entities and their relationships, serialized in a format compatible with the *Resource Description Framework* (*RDF*) [9].

Entities in an RDF graph are named using URIs. A common case of RDF is *Linked Data* [3], where the entity names are HTTP URIs which respond to a `GET` request with an *authoritative* RDF representation. Such hyperlinked datasets form dependency graphs such as Figure 1. For example, a `GET` on Bob's home page `bob:` (using abbreviations such as `bob:` for URI prefixes such as `http://example.com/bob#`) might return HTML:

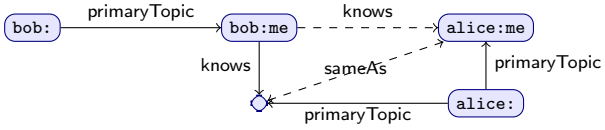
```
<html><head><base href="bob:"/></head>
  <body rel="foaf:primaryTopic" resource="bob:me">
    <p rel="foaf:knows">
      I know <a rev="foaf:primaryTopic"
        href="alice:">Alice</a>.</p></body></html>
```

As well as the human-readable text, this page contains RDFa annotations, from which an RDF-aware robot can extract the graph:



that is, "Bob's home page describes Bob, who knows someone who is described by Alice's home page." Now, if the same robot visits Alice's home page, and extracts the graph:

then a reasoner can deduce "Bob knows Alice":



These deductions are supported by an *ontology* defined in a language such as the *W3C Web Ontology Language* (*OWL*) [5], whose building blocks are ones familiar to a programming languages audience, including classes, subclasses, and types and arities of relationships.

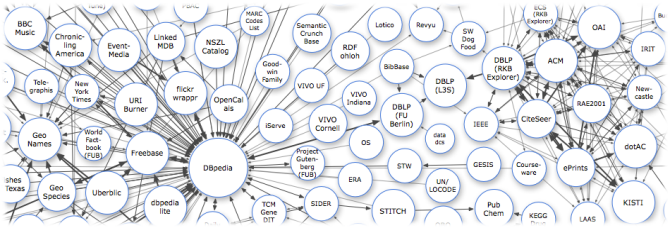


Figure 1. Linking Open Data cloud diagram (detail)

Programming languages for XML. Starting in 2001, there was a burst of interest in specialized languages for processing XML data, notably XDuce [6] and CDuce [2]. These languages have many features in common, notably:

- they are strongly typed functional languages,
- they include schemas such as DTDs or XML Schemas in their type systems, with subtyping for language inclusion, and
- given valid input (with respect to a schema), well-typed programs produce valid output.

Programming languages for RDF? At first glance, RDF seems that it should be a more natural fit for a programming language than XML, since its concerns (entities, relationships, classes and subclasses) appear to be a more natural fit than those of XML (nodes, children, schemas, regular languages, and language inclusion). Indeed, such a language might help alleviate the impedance mismatch between XML and OO languages, which results in complex serialization frameworks such as JAXB.

However, there are some features of ontology languages which mean that there are some research challenges in designing an RDF-aware programming language:

- *Rich type system*: ontology languages typically have a very rich language of type expressions, including intersection and union types, type recursion, and modal types (discussed below).
- *Classical foundations*: the logical foundations of programming languages are typically constructive, given by a Curry–Howard isomorphism. In contrast, ontology languages are typically classical, and include features such as type negation and excluded middle (every object is a member of the type $T \sqcup \neg T$).
- *Arbitrary subtype constraints*: in most programming languages, subtype constraints are of a limited form, for example Java only supports subtypes of the form $\mathsf{c}\langle\vec{X}\rangle \sqsubseteq \mathsf{d}\langle\vec{T}\rangle$. In contrast, ontology languages allow subtype constraints $T \sqsubseteq U$ for arbitrarily complex type expressions T and U.
- *Incomplete information*: in OO languages, there is a canonical source of information about an object, which is the object itself.

Description Logic	Hennessy-Milner Logic
Concept	Proposition
Concept name	Atomic proposition
Role name	Action
Interpretation	Labeled transition system / Kripke structure

Figure 2. Description Logic cheat-sheet

In contrast, RDF allows assertions of new relationships about existing entities. For example, Bob's home page might assert that Alice's name is "Alice", even though Bob is not authoritative for Alice.

- *Data validation*: in the XML case, the safety condition is clear – valid input should result in valid output. Data validity for ontologies is an area of current research, so it is not clear what the victory conditions are, never mind how to achieve them.
- *Nominal data*: in languages such as XDuce, trees are immutable, so node identity is unimportant. In contrast, RDF data forms a graph in which node identity is important. As a result, languages for processing RDF must be aware of entity names, and must be at least as powerful as the ν-calculus [10].

Towards a programming language for RDF

Description logics. The formal basis of ontology languages is *description logic* (*DL*) [1], which is strongly related to modal logic. A simple but illustrative DL is $\mathcal{ALCF}$, where *concepts* are defined by grammar (where r and c are drawn from sets of atomic role names Rol and concept names Con):

$$C \quad ::= \quad c \mid \neg C \mid \bot \mid C_1 \sqcap C_2 \mid \forall r \,.\, C \mid \leq\!1\, r$$

This logic (up to syntax) is an extension of *Hennessy–Milner Logic* (*HML*) [4] with a uniqueness modality $\leq\!1\, r$. An *interpretation* is a labeled transition systems with Kripke structure, that is a triple $(S, \rightarrow, L)$ for $\rightarrow \subseteq (S \times \mathsf{Rol} \times S)$ and $L \subseteq (S \times \mathsf{Con})$. Satisfaction is defined as usual for HML, but with one new clause:

$$\mathcal{M}, s \vDash \leq\!1\, r \text{ whenever } s \xrightarrow{r} t \text{ and } s \xrightarrow{r} u \text{ implies } t = u$$

A summary of the correspondence between DL and HML is given in Figure 2. Note that DLs are classical, so De Morgan duality characterizes properties such as sub-concept ($C \sqsubseteq D$ whenever $\neg(C \sqcap \neg D)$. In RDF, concepts are used as types for entities, for example the type for Bob's home page is $\mathsf{Document}$, which satisfies $\mathsf{Document} \sqsubseteq \leq\!1\, \mathsf{primaryTopic}$.

"Just" data. The simplest case of a program for RDF is a function which immediately returns an RDF graph. This is equivalent to an RDF graph with free variables, or (by analogy to module systems) an RDF graph which distinguishes between imported and exported entities. In a recent paper [8], we investigated validity for such *partitioned* RDF graphs. For example, Bob's home page validates with imports (`alice:` : $\mathsf{Document}$) and exports (`bob:` : $\mathsf{Document}$).

Models for programming languages are often categorical in nature, so we hope to find categorical structure in the data language. In the case of partitioned RDF graphs, they form a symmetric monoidal category, where the objects are RDF graphs (thought of as interfaces) and the morphisms from A to B are validated RDF graphs which import A and export B. The category is only symmetric monoidal, not cartesian, for the usual reason in languages which include name generation (generating a name then copying it is not the same as generating two names).

Since the category is symmetric monoidal, it supports a diagrammatic presentation (see, for example [11]), and so provides a formal basis for dependency graphs such as Figure 1.

First order programming. There are first-order languages for querying RDF, notably the SPARQL query language. Such languages are not intended as general purpose programming languages, so only provide limited support for features such as branching and recursive functions. They do not support static typing to ensure validity of generated RDF.

A general-purpose first-order programming language for RDF could be a first-order lambda-calculus extended with records and name generation [10]. A first cut type system would just add DL concepts as base types to the simply typed λ-calculus, and inherit subtyping from concept inclusion. Such a type system would use an algorithm for determining concept inclusion in the base case, but would otherwise be structural. The type system would still be non-trivial, for reasons discussed in the introduction.

Categorically, we would expect to see an extension of the category of RDF graphs to support coproducts (for branching) and a form of partial trace (for recursion).

Higher order programming. In the higher-order case, the mix of RDF and functions becomes more interesting, with the possibility of supporting objects with methods as well as fields. The type system for such a language could be based on semantic subtyping [2], and so would no longer have a concept inclusion algorithm as an isolated component. Since the categorical model of RDF is symmetric monoidal (with a cartesian center), we expect that the model for higher order programming would be monoidal closed (with a cartesian closed center).

Implementation. We have mechanized a description logic and its semantics in Agda, and used it to prove the categorical structure of partitioned RDF [7]. It may be that such a mechanization could form the basis of a semantic web library, using Agda's dependent types to embed subtyping, rather than having to define a new domain-specific language.

References

[1] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2nd edition, 2007.

[2] V. Benzaken, G. Castagna, and A. Frisch. CDuce: An XML-centric general-purpose language. In *Proc. ACM Int. Conf. Functional Programming*, 2003.

[3] T. Berners-Lee. Linked data, 2006. `http://www.w3.org/DesignIssues/LinkedData.html`.

[4] M. C. B. Hennessy and R. Milner. Algebraic laws for non-determinism and concurrency. *J. ACM*, 32(1):137–161, 1985.

[5] P. Hitzler, M. Krötzsch, B. Parsia, P.F. Patel-Schneider, and S. Rudolph. OWL 2 web ontology language primer. W3C Recommendation, 2009. http://www.w3.org/TR/owl2-primer/.

[6] H. Hosoya and B.C. Pierce. Regular expression pattern matching for XML. *J. Functional Programming*, 13(6):961–1004, 2002.

[7] A. S. A. Jeffrey. Agda libraries for the semantic web. `https://github.com/agda/agda-web-semantic/`, 2011.

[8] A.S.A. Jeffrey and P.F. Patel-Schneider. Integrity constraints for linked data. In *Proc. Int. Workshop on Description Logics*, 2011.

[9] F. Manola and E. Miller. RDF primer. W3C Recommendation, 2004. http://www.w3.org/TR/rdf-primer/.

[10] A. M. Pitts and I. Stark. Observable properties of higher order functions that dynamically create local names, or: What's new? In *Proc. Math. Foundations of Computer Science*, pages 122–141, 1993.

[11] P. Selinger. A survey of graphical languages for monoidal categories. In B. Coecke, editor, *New Structures for Physics*, volume 813 of *Lecture Notes in Physics*, chapter 4, pages 289–356. Springer, 2011.