Deep Integration of Ruby with Semantic Web Ontologies

Obie Fernandez

ThoughtWorks, Inc.

Goals of this Presentation

- Brief introduction to Semantic Web technology such as RDF and OWL
- Introduce idea and benefits of using Ruby deeply integrated with RDF and Ontologies
- Propose a Ruby library that realizes those benefits

Resource Description Framework

- The basic building block in RDF is an subject-predicate-object *triple*
- Any object from one triple can play the role of a subject in another triple
- Any RDF statement itself can be subject or object
- Plain RDF makes no data modeling commitments
- RDF itself provides no mechanisms for declaring vocabulary with semantic meaning

RDF Expressed in XML

```
<rdf:Description
   rdf:about="http://www.famouswriters.org/twain/mark">
   <s:hasName>Mark Twain</s:hasName>
   <s:hasWritten
        rdf:resource="http://books.org/ISBN0001047582"/>
</rdf:Description>

<rdf:Description rdf:about="http://www.books.org/ISBN0001047582">
   <s:title>The Adventures of Tom Sawyer</s:title>
   <rdf:type rdf:resource="http://desc.org/schema#Book"/>
</rdf:Description>
```

Example from http://www.openrdf.org/doc/papers/Sesame-ISWC2002.pdf

RDF Schema

RDF Vocabulary Description Language 1.0

- Used for simple hierarchy of concepts and properties
- Enhances plain RDF with terms like
  - Class, Property, Literal
  - subClassOf, subPropertyOf
  - range, domain
- In other words, provides a **type** system
- RDF Schema is expressed using RDF syntax

About Ranges and Domains

- RDF Schema describes how properties and classes are intended to be used together
- The most important information of this kind is supplied by using the RDF Schema properties **rdfs:range** and **rdfs:domain** to further describe application-specific properties
- Seems similar to normal *type* system but isn't

Property Ranges (Example)

Defining a range for hasMother puts a limitation on what values are allowed for that property:

- hasMother **rdf:type** Property
- hasMother **rdfs:range** Female
- hasMother **rdfs:range** Person

Example RDF statement:

- frank **hasMother** maria

In order for the statement to be semantically correct, maria must be a Female and a Person...

Domain Ranges (Example)

Defining a domain for weight puts a limitation on what classes may use that property:

- weighs **rdf:type** DatatypeProperty
- weighs **rdfs:range** xsd:integer
- weighs **rdfs:domain** Vehicle
- weighs **rdfs:domain** Book

Example RDF statement:

- myCar **weighs** 2500 !?

Care is needed when defining domains!

This example implies that any resource with a weighs property is both a Vehicle *and* a Book.

Description Logic (DL)

- In DL properties have their own semantic identity and are applied to specific classes of resources
- In OO programming languages properties are defined as attributes of their class
- Contrast these DL class definitions with typical OO programming...
  - TeachersKid: All instances that are Human and hasMother who is a Teacher
  - Father: All instances that are not instances of Mother and is in the domain of hasChildren

OWL Web Ontology Language

- Extends RDF Schema with richer descriptions of classes
- Can include descriptions of classes, properties plus instances
- OWL *formal semantics*

  Derive logical consequences, meaning facts that are not literally present in the ontology
- OWL *mechanisms*

  Derive consequences based on combining multiple ontology documents (including publicly available ontologies off the web)

Java Doesn’t Work for Ontologies

- Java is a *frame language:* an object (or frame) is the main unit of structuring data
- Polymorphism in RDF and OWL is very different than in Java which makes for admittedly clumsy API
- Properties are defined separately from classes and can be applied at will to any resource
- Jena is a Java open-source framework for building semantic web apps…

```
Resource r = myModel.getResource( myNS + "DigitalCamera" );
OntClass cls = (OntClass) r.as( OntClass.class );
Restriction rest = (Restriction) cls.as( Restriction.class );
```

Deep Integration of OWL with Dynamic Languages

- Importing ontologies directly into the programming context so that its classes are usable alongside classes defined normally
- Support *necessary and complete* class definitions of DL, where the attributes of an object determines its *Class*
- Behavioral code added only to class definitions where it's most sensible and realistic

Deep Integration Encourages Separation of Concerns

- Splits declarative and procedural aspects cleanly
- Introduces language independence for data while retaining semantic meaning
- Declarative aspects can be authored by non-technical domain experts with ontology editors such as Protégé

Description Logic in Ruby

- Ruby is a dynamic language (similar to Python)
- Can be described as intersection of Smalltalk and PERL
- Method calls are messages, like in Smalltalk
- Unknown messages can be handled by overriding method_missing of base Object class
- Instances whose classes are defined in ontology can use inference engine or reasoner for help in determining if/how to respond to messages

Deep Integration Ruby Examples

```
# declarative examples

class Human < OWL::Thing

  defined_with_xml
      “http://example.org/humans.owl”

end

class TeachersChild < OWL::Thing

      defined_with_query “TeachersChild? kindOf
      Human ^ TeachersChild? hasMother (kindOf
      Teacher)”

      def misbehave
        raise “Teacher’s kids never misbehave!”
      end

end
```

```
# inline declaration and usages

> TeachersChild = Thing.new(“TeachersChild?
      kindOf Human ^ TeachersChild? hasMother
      {kindOf Teacher}”)

> suzy = TeachersChild.new

> all_teachers_kids = TeachersChild.find_all

> suzy.kind_of? Human true

> billy = Human.new

> billy.kind_of? TeachersChild unknown

> billy.hasMother = suzy.hasMother

> billy.kind_of? TeachersChild true

> billy.hasSister? Suzy true
```

Why Deep Integration Enhances Programming with Ruby

- Definition of problem domain and data done with powerful graphical tools such as Protégé instead of text
- OWL ontologies are inherently shareable as part of the semantic web family
- Semantically-rich RDFS becomes the serialization format for any given object graph
- Gives Ruby programmer powerful inferencing capabilities against the underlying RDF database

Treating Ontologies as Programming Libraries not Data

- The programmer imports ontologies rather than *loading* them
- Programmer works directly with ontology classes, not via API interfaces such as Resource, OntClass, Restriction, etc...
- Exposes the power of description logics transparently
- Represents a significant paradigm-shift

Impact on Lookup Data and Enumerations

- Programmers typically have to manage quantities of lookup data as constants and enumerations specified directly inside the implementation code or in database tables
- Deep integration means that lookup data can be authored and managed with ontology tools
- Instances of lookup data are imported automatically along with ontology

Open World Assumption

- Descriptions of resources are not confined to single scope
- Simply because something is not stated as a fact does not imply that it isn't true, therefore we cannot assume to know everything, only what we know to be factual
- Methods that would traditionally be thought of as having boolean return types might also need to return **Unknown**

Example (Can a spoon be played as a musical instrument?)

```
spoon = KitchenUtensil.new

spoon kind_of? MusicalInstrument unknown
```

To be continued…

The author is working on open-source implementation of the ideas described above to be made available on RubyForge

Updates available at http://enterpriseagile.blogspot.com