Semantic web concepts and tools in Bioconductor

©2006 VJ Carey

July 28, 2006

Contents

1   Introduction

Semantic web activities of W3C include the formulation of standards to support data interchange via WWW. Formal models for metadata are a major concern, and the Resource Description Framework (RDF) model is the primary product in this area.

A useful resource is Shelly Powers' monograph *Practical RDF* (pub. O'Reilly, 2003). She notes:

> ... the RDF specification is about metadata – data about data. This is a key RDF concept; by creating a domain-neutral specification to describe resources, the same specification can then be used with many different domains but still be processed by the same RDF agents or parsed by the same RDF parsers. (p. 84)

She also cites the W3C OWL comment on ontologies:

> An ontology formally defines a common set of terms that are used to describe and represent a domain. Ontologies can be used by automated tools to power advanced services such as more accurate web search, intelligent software agents, and knowledge management. (p. 228)

Because bioinformatics involves so many problems of data integration from heterogeneous and distributed source, tools for remote metadata access and interpretation are in high demand. A W3C working group on Semantic Web for Life Sciences has been formed and has had several meetings and active discussion on a public web mailing list.

This lab focuses on the use of R to work with RDF models and their serializations. RDF models can be used for gene ontology and related ontologies in biology. RDF is used to encode models in a richer metadata modeling system called OWL (for Web Ontology Language). Using Bioconductor tools one can parse and transform documents in OWL and/or RDF to work with emerging metadata standards for life sciences data.

There is no pretense in this lab to provide a comprehensive view of work in semantic web research for life sciences. The *Journal of Web Semantics* has some relevant resources. Much of the work in this area should be considered high risk.

2   The RDF model in brief

RDF prescribes a graphical model for information. Information takes the form of a directed graph. An arc in the graph consists of a subject node, a directed predicate edge, and an object node. We will use the term "blank node" to refer to a non-specific, but uniquely identifiable, element of the graph.

- In the original specification, the subject node was required to be a blank node or a Uniform Resource Identifier reference (URI reference, defined by Berners-Lee et al., see appendix section 7.) [This is now updated to be an Internationalized Resource Identifier (IRI).]

- The predicate node is required to be a URI reference

- The subject node is required to be a URI reference, a literal, or a blank node.

For concreteness, we will examine an serialization of such a graph in an R data frame. The R object `gordb` is a representation of the entire GO ontology (see section 3.2.1) in subject-predicate-object form.

```
> library(Rredland)
> data(gordb)
> gordb[1:3, ]

                     subject                                     predicate
NA     (r1154007138r8790r1)    http://www.w3.org/2002/07/owl#someValuesFrom
102884 (r1154007138r8790r1) http://www.w3.org/1999/02/22-rdf-syntax-ns#type
NA.1   (r1154007138r8790r2)         http://www.w3.org/2002/07/owl#onProperty
                                         object
NA      http://www.geneontology.org/owl/#GO_0006310
102884    http://www.w3.org/2002/07/owl#Restriction
NA.1      http://www.geneontology.org/owl/#part_of
```

These rows involve blank nodes for subjects; to motivate this representation more straightforwardly, we consider how to understand the structure related to the term `metaphase`. The following code exploits knowledge of the fact that a predicate with substring "label" is used to link the subject [GO term label] to the actual string literal for the term:

```
> ac = as.character
> tag = gordb[intersect(grep("\"metaphase\"", ac(gordb[, 3])),
+     grep("label", ac(gordb[, 2]))), 1]
> tag

[1] http://www.geneontology.org/owl/#GO_0051323
24467 Levels: (r1154007138r8790r1) ... http://www.geneontology.org/owl/#part_of
```

We have discovered the GO symbolic tag used for the simple term "metaphase". Now let us look at all RDF arcs involving this tag as subject. (We define a chomp function to deal with the fact that some comments are very long and inhibit printing of neighboring strings):

```
> chomp = function(x) {
+     data.frame(lapply(x, function(x) substring(x, 1, 60)))
+ }
> tmp <- chomp(gordb[grep(tag, gordb[, 1]), ])
> tmp
```

```
                                 subject
1 http://www.geneontology.org/owl/#GO_0051323
2 http://www.geneontology.org/owl/#GO_0051323
3 http://www.geneontology.org/owl/#GO_0051323
4 http://www.geneontology.org/owl/#GO_0051323
                                              predicate
1      http://www.w3.org/2000/01/rdf-schema#label
2    http://www.w3.org/2000/01/rdf-schema#comment
3 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
4 http://www.w3.org/2000/01/rdf-schema#subClassOf
                                                        object
1                                                  "metaphase"
2 "Progression through metaphase, the second stage of chromoso
3                           http://www.w3.org/2002/07/owl#Class
4                                     (r1154007138r8790r3402)
```

Now we see that the metaphase term is said to be a subclass of an entity identified as a blank node (r1154007138r8790r3402).

```
> theblank = as.character(tmp[grep("subClassOf", ac(tmp[, 2])),
+     3])
> theblank

[1] "(r1154007138r8790r3402)"
```

Let's look at the arcs for which this blank node is subject:

```
> chomp(gordb[grep(theblank, ac(gordb[, 1])), ])

              subject                                      predicate
1 (r1154007138r8790r3402)          http://www.w3.org/2002/07/owl#onProperty
2 (r1154007138r8790r3402)     http://www.w3.org/2002/07/owl#someValuesFrom
3 (r1154007138r8790r3402) http://www.w3.org/1999/02/22-rdf-syntax-ns#type
                            object
1    http://www.geneontology.org/owl/#part_of
2 http://www.geneontology.org/owl/#GO_0000279
3   http://www.w3.org/2002/07/owl#Restriction
```

Now we see that the relationship of GO:0051323 to GO:0000279 is not a simple "is-a" or subclass; it is a part-of constituting "some values from" the process named by 279. What is that?

```
> chomp(gordb[grep("0000279", ac(gordb[, 1])), ])
```

```
                                    subject
1 http://www.geneontology.org/owl/#GO_0000279
2 http://www.geneontology.org/owl/#GO_0000279
3 http://www.geneontology.org/owl/#GO_0000279
4 http://www.geneontology.org/owl/#GO_0000279
                                              predicate
1        http://www.w3.org/2000/01/rdf-schema#label
2      http://www.w3.org/2000/01/rdf-schema#comment
3 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
4 http://www.w3.org/2000/01/rdf-schema#subClassOf
                                                        object
1                                                      "M phase"
2 "Progression through M phase, the part of the cell cycle com
3                             http://www.w3.org/2002/07/owl#Class
4                                       (r1154007138r8790r71)
```

We have now learned that metaphase is a part of M phase. Elaborating the objects of the comment predicates would teach us more.

In summary, the graphical model we have just explored using R is:

```
                 rdfs:label
GO:00051323 ------------------> "metaphase"
             owl:subClassOf
           ------------------> (b1)


                 rdf:type
(b1)         ------------------> owl:restriction
             owl:onProperty
           ------------------> go:part-of
           owl:someValuesFrom
           ------------------> GO:0000279


                 rdfs:label
GO:0000279  ------------------> "M phase"
```

We are using `tok:` prefixes to denote the provenance of a metadata concept. We really are not supposed to make any assumptions about the meanings of "label", "type" and so on – the meanings are fixed by the authorities identified in these namespace qualification prefixes. We see that some of the metadata terms are coming from RDF schema (rdfs), some from RDF syntax (rdf) some from owl, some from GO.

There seems to be some artificiality here. Is metaphase properly thought of as a subclass of M phase? No. The subclass relationship is used to denote a generic hierarchical subsumption. The details of the relationship are complex and require multiple

predicates, so the blank node is used to allow emanation of multiple arcs to specify the properties defining the relationship.

3   Starting out with Rredland

3.1   Getting acquainted

The *Rredland* package is a prototype interface to the C-based `librdf.org` resources for parsing and modeling information in RDF. `librdf` (Redland) has a very rich API and only a small portion of it is exposed at present.

```
> library(Rredland)
```

Functions currently available include:

```
> objects("package:Rredland")
```

```
 [1] "cleanXSDT"              "freeRedl"
 [3] "getArcsWith"            "getClassElements"
 [5] "getClassGraph"          "getDatatypeProperties"
 [7] "getOWLClasses"          "getOWLProperties"
 [9] "getOWLSubclasses"       "getObjectProperties"
[11] "getPropertiesWithDomain" "getPropertyRange"
[13] "getStatus"              "makeRedlURI"
[15] "nodeFromURIString"      "openRedlWorld"
[17] "readRDF"                "ref"
[19] "restoreBDB"             "setStatus"
[21] "size"                   "world"
```

A fundamental tool is the import function:

```
> args(readRDF)
```

```
function (uri, storageType = c("internal", "bdb")[1], storageName = "test",
    world = ..GredlWorld, stoHash = NULL)
NULL
```
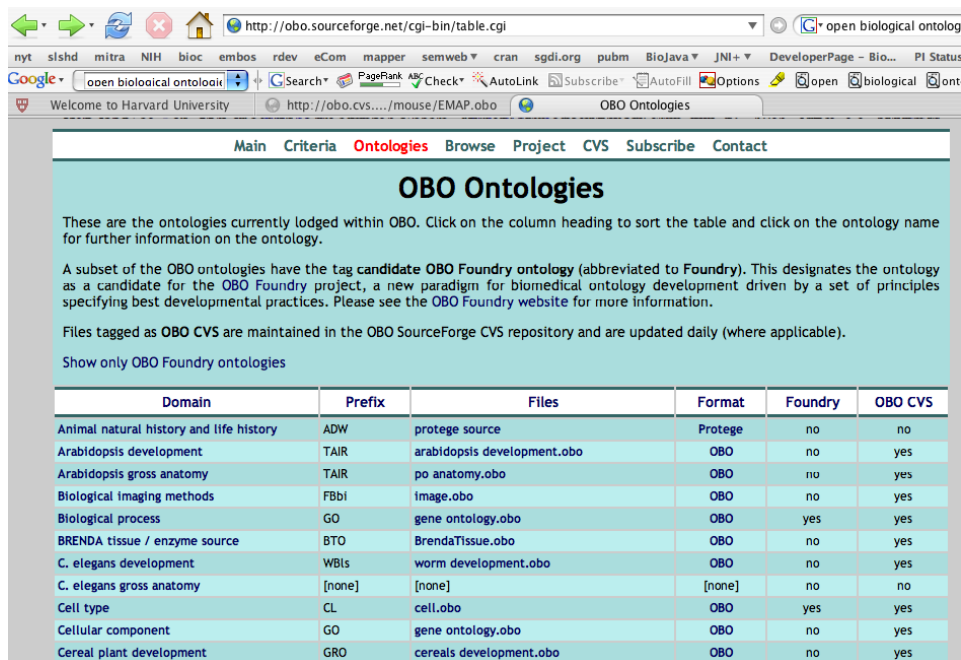
There are two possible ways to identify an RDF resource for importing. The simplest approach uses a string URI (e.g., ■`file:some.rdf`■). Alternatively, one can construct a librdf URI object and pass that to the import function `readRDF`. You can do

```
example(readRDF)
```

after attaching the library to see some demonstrations.

3.2 The Open Biological Ontologies

The OBO project collects a variety of vocabularies with the aim of supporting shared use of standard terminologies.

3.2.1 Gene ontology

One of the most familiar resources to bioinformaticians is Gene Ontology (GO). GO is serialized and distributed by OBO in their OBO format. A small excerpt from the markup is given here:

```
format-version: 1.0
date: 24:07:2006 10:24
saved-by: gwg
auto-generated-by: OBO-Edit 1.002
subsetdef: goslim_generic "Generic GO slim"
subsetdef: goslim_goa "GOA and proteome slim"
subsetdef: goslim_plant "Plant GO slim"
subsetdef: goslim_yeast "Yeast GO slim"
subsetdef: gosubset_prok "Prokaryotic GO subset"
default-namespace: gene_ontology
remark: geneontology.org version: Revision: 4.30

[Term]
id: GO:0000001
name: mitochondrion inheritance
```

```
namespace: biological_process
def: "The distribution of mitochondria, including the mitochondrial
    genome, into daughter cells after mitosis or meiosis, mediated
    by interactions between mitochondria and the cytoskeleton." [GOC:mcc,
    PMID:10873824, PMID:11389764]
exact_synonym: "mitochondrial inheritance" []
is_a: GO:0048308 ! organelle inheritance
is_a: GO:0048311 ! mitochondrion distribution
```

This markup is fairly easy for a human to read and understand. However, programmatic translation requires some effort or special tools. The XML standard for markup is advantageous, in that very generic tools for parsing and isolating structures can be deployed. RDF/XML goes beyond XML in that the tag set and informatin structure is fully specified by the RDF standard.

A translator from OBO markup to OWL/RDF/XML is available. The translated excerpt is:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.geneontology.org/owl/#"
    xml:base="http://www.geneontology.org/owl/">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="GO_0000001">
    <rdfs:label>mitochondrion inheritance</rdfs:label>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">The
      distribution of mitochondria, including the mitochondrial genome,
      into daughter cells after mitosis or meiosis, mediated by interactions
      between mitochondria and the cytoskeleton.</rdfs:comment>
<!-- organelle inheritance -->
    <rdfs:subClassOf rdf:resource="#GO_0048308"/>
<!-- mitochondrion distribution -->
    <rdfs:subClassOf rdf:resource="#GO_0048311"/>
  </owl:Class>
```

There is extensive use of namespace qualification. You can find the definition of the OWL vocabulary defined in RDF at

    http://www.w3.org/2002/07/owl

and see the appendix section 6 for an excerpt.

3.2.2 Exercises 1: Inspecting the triples representation for GO

1. Your copy of the Rredland library includes a data.frame `gordb` that represents a parsed and translated version of the RDF model for GO. Here again are the first three records:

```
> data(gordb)
> gordb[1:3, ]
```

```
                      subject                                    predicate
NA     (r1154007138r8790r1)    http://www.w3.org/2002/07/owl#someValuesFrom
102884 (r1154007138r8790r1) http://www.w3.org/1999/02/22-rdf-syntax-ns#type
NA.1   (r1154007138r8790r2)          http://www.w3.org/2002/07/owl#onProperty
                                    object
NA     http://www.geneontology.org/owl/#GO_0006310
102884   http://www.w3.org/2002/07/owl#Restriction
NA.1       http://www.geneontology.org/owl/#part_of
```

Note that the order of rows is completely arbitrary. In this case, we see a number of blank nodes; section 2 above illustrates interpretation of such structures.

Emulate the programming in section 2 to identify the immediate ancestor of apoptosis in the GO graph. You can use the functions

```
> arcsWsubj = function(subj) gordb[grep(subj, ac(gordb[, 1])),
+     ]
> term2tag = function(term) {
+     tmp = gordb[grep(term, ac(gordb[, 3])), ]
+     tmp[grep("label", ac(tmp[, 2])), 1]
+ }
```

to find this tag. Note that these are very specialized functions that only work if there is a `gordb` defined.

2. How can we find the children of the apoptosis term in the GO graph?

3.2.3 Mouse anatomy

The EMAP.obo ontology for mouse developmental anatomy is self describing, with the following remark near the top of the document:

> The Anatomical Dictionary for Mouse Development has been developed at the Department of Anatomy, University of Edinburgh, Scotland and the MRC Human Genetics Unit, Edinburgh as part of the Edinburgh Mouse Atlas project (EMAP), in collaboration with the Gene Expression (GXD)

project at MGI, The Jackson Laboratory, Bar Harbor, ME. Copyright 1998-2002 University of Edinburgh (UK) and MRC (UK). Questions and comments should be sent to Jonathan Bard (J.Bard@ed.ac.uk) The file

'Mouse_anatomy_by_time_xproduct'

contains the stage-specific anatomical structures. The anatomical structures for each developmental stage are listed hierarchically in the form of a tree structure, using part-of relationships. The developmental stage (Theiler stage = TS) is displayed followed by the 'print name' for the anatomical structure. The print names are generated by GXD to unambiguously identify anatomical structures for the user. If the name for a node is unambiguous, only the node name itself is displayed. Otherwise, the node's name is followed by a minimal number of parent node's names to provide sufficient context. The EMAP id refers to the respective stage-specific anatomical structure.

3.2.4 Exercises 2: Assessing anatomical change in mouse by Theiler stage

Again, a data frame has been cooked for you providing the EMAP ontology in triples form.

```
> data(EMAPdf)
> EMAPdf[1:3, ]

                     subject                                     predicate
NA     (r1154000089r7521r1)     http://www.w3.org/2002/07/owl#someValuesFrom
82385 (r1154000089r7521r1) http://www.w3.org/1999/02/22-rdf-syntax-ns#type
NA.1  (r1154000089r7521r2)          http://www.w3.org/2002/07/owl#onProperty
                                  object
NA       http://www.geneontology.org/owl/#EMAP_0
82385 http://www.w3.org/2002/07/owl#Restriction
NA.1   http://www.geneontology.org/owl/#part_of
```

1. At what TS stage does Reichert's membrane emerge, and when does it disappear?

2. How many anatomical features are static between TS18 and TS19? Identify some features present in TS18 that are gone at TS19.

3. At what Theiler stage does pigmented retinal epithelium emerge?

4. Emulate the graph at the end of section 2 to illustrate the relationship between EMAP_1904 and EMAP_1905.

3.2.5 Exercises 3: Parsing and inspecting the MGED OWL ontology

You can find the MGEDOntology (version 1) in OWL format as follows:

```
> mgfi = system.file("RDF/MGEDOntology.owl", package = "Rredland")
> fiu = paste("file:", mgfi, sep = "")
> substr(readLines(mgfi, n = 15), 1, 70)

 [1] "<?xml version=\"1.0\"?>"
 [2] "<rdf:RDF"
 [3] "    xmlns:protege=\"http://protege.stanford.edu/plugins/owl/protege#\""
 [4] "    xmlns:rdf=\"http://www.w3.org/1999/02/22-rdf-syntax-ns#\""
 [5] "    xmlns:xsd=\"http://www.w3.org/2001/XMLSchema#\""
 [6] "    xmlns:rdfs=\"http://www.w3.org/2000/01/rdf-schema#\""
 [7] "    xmlns:owl=\"http://www.w3.org/2002/07/owl#\""
 [8] "    xmlns:daml=\"http://www.daml.org/2001/03/daml+oil#\""
 [9] "    xmlns:dc=\"http://purl.org/dc/elements/1.1/\""
[10] "    xmlns=\"http://mged.sourceforge.net/ontologies/MGEDOntology.owl#\""
[11] "  xml:base=\"http://mged.sourceforge.net/ontologies/MGEDOntology.owl\">"
[12] "  <owl:Ontology rdf:about=\"\">"
[13] "    <dc:title rdf:datatype=\"http://www.w3.org/2001/XMLSchema#string\""
[14] "    >The MGED Ontology</dc:title>"
[15] "    <dc:creator rdf:datatype=\"http://www.w3.org/2001/XMLSchema#string\""
```

1. Use readRDF to import the OWL model. How many statements are there?

2. Create the associated data.frame. This employs the `as(x, "data.frame")` pattern. Emulate the graph diagram at the end of section 2 to illustrate the relationship between the DiseaseLocation class and primary_site. There are four blank nodes involved.

4   Intact.rdf and protein-protein interaction

A fragment of the intact database, serialized to RDF by Eric Jain of ISB-CH, is provided.

```
> infi = system.file("RDF/partIntact.rdf", package = "Rredland")
> fii = paste("file:", infi, sep = "")
> substr(readLines(infi, n = 15), 1, 70)

[1] "<?xml version='1.0' encoding='UTF-8'?>"
[2] "<rdf:RDF xmlns=\"urn:lsid:uniprot.org:ontology:\" xmlns:rdf=\"http://www."
[3] "<rdf:Description rdf:about=\"urn:lsid:uniprot.org:intact:EBI-300303\">"
[4] "<rdf:type rdf:resource=\"urn:lsid:uniprot.org:ontology:Interaction\"/>"
[5] "<rdfs:label>aret-apt-1</rdfs:label>"
[6] "<rdfs:comment>Interaction detected by coIP</rdfs:comment>"
[7] "<participant rdf:resource=\"urn:lsid:uniprot.org:uniprot:O16114\"/>"
[8] "<participant rdf:resource=\"urn:lsid:uniprot.org:uniprot:O18409\"/>"
```

```
 [9] "</rdf:Description>"
[10] "<rdf:Description rdf:about=\"urn:lsid:uniprot.org:intact:EBI-297969\">"
[11] "<rdf:type rdf:resource=\"urn:lsid:uniprot.org:ontology:Interaction\"/>"
[12] "<rdfs:label>hoxb1-pbx1-1</rdfs:label>"
[13] "<rdfs:comment>Interaction detected by X-ray crystallography.</rdfs:com"
[14] "<participant rdf:resource=\"urn:lsid:uniprot.org:uniprot:P14653\"/>"
[15] "<participant rdf:resource=\"urn:lsid:uniprot.org:uniprot:P40424\"/>"
```

We can get a feel for the character of the resource as follows, after parsing and converting to a triples data frame:

```
> dint[1:5,]
                                            subject
1 urn:lsid:uniprot.org:intact:EBI-300303
2 urn:lsid:uniprot.org:intact:EBI-300303
3 urn:lsid:uniprot.org:intact:EBI-300303
4 urn:lsid:uniprot.org:intact:EBI-300303
5 urn:lsid:uniprot.org:intact:EBI-300303
                                          predicate
1 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
2       http://www.w3.org/2000/01/rdf-schema#label
3    http://www.w3.org/2000/01/rdf-schema#comment
4        urn:lsid:uniprot.org:ontology:participant
5        urn:lsid:uniprot.org:ontology:participant
                                             object
1 urn:lsid:uniprot.org:ontology:Interaction
2                               "aret-apt-1"
3           "Interaction detected by coIP"
4        urn:lsid:uniprot.org:uniprot:O16114
5        urn:lsid:uniprot.org:uniprot:O18409
```

To study the protein-protein interaction graph accumulated over the experiments, we focus attention on the triples with participant predicates, split them by the interaction label values, and join participants when interaction is declared.

4.1 Exercises with Intact data

1. Examine the comment fields. How would you add additional structure to reflect specific experimental approaches, distinguishing for example Y2H and GST pull-down? Does the existing RDF data need to be altered in any way?

2. Create the interaction graph. Use the graph package to assess the degree distribution. Use Rgraphviz to plot.

3. Read the data using the BDB hash representation. Look at the sizes of the constructed hashes and compare to the size of the saved data frame.

5 Further work

On suitable hardware, you could analyze the entire intact database. Some work to analyze the comment fields could lead to added value by distinguishing experiment types systematically.

The BioPAX ontologies (level1 and 2) are provided in OWL format in the Rredland package. Explore them. Obtain the Reactome serializations based on BioPAX ontologies, and evaluate their utility.

Berners-Lee's closed world machine (google on that phrase) is an interesting simple resource for rule-based inference on RDF models.

Creating OWL/RDF models de novo is accomplished conveniently using protege (protege.stanford.edu).

6 Appendix: Excerpt from the formal definition of the OWL vocabulary

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
     <!ENTITY rdf  "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
     <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
     <!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#" >
     <!ENTITY owl  "http://www.w3.org/2002/07/owl#" >
  ]>

<rdf:RDF
  xmlns     ="&owl;"
  xmlns:owl ="&owl;"
  xml:base  ="http://www.w3.org/2002/07/owl"
  xmlns:rdf ="&rdf;"
  xmlns:rdfs="&rdfs;"
>

<Ontology rdf:about="">
  <imports rdf:resource="http://www.w3.org/2000/01/rdf-schema"/>
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/TR/2004/REC-owl-semantics-20040210/" />
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/TR/2004/REC-owl-test-20040210/" />
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/TR/2004/REC-owl-features-20040210/" />
  <rdfs:comment>This file specifies in RDF Schema format the
    built-in classes and properties that together form the basis of
```

```
      the RDF/XML syntax of OWL Full, OWL DL and OWL Lite.
      We do not expect people to import this file
      explicitly into their ontology. People that do import this file
      should expect their ontology to be an OWL Full ontology.
    </rdfs:comment>
    <versionInfo>10 February 2004, revised $Date: 2006/07/28 05:34:42 $</versionInfo>
    <priorVersion rdf:resource="http://www.daml.org/2001/03/daml+oil"/>
</Ontology>

<rdfs:Class rdf:ID="Class">
  <rdfs:label>Class</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdfs;Class"/>
</rdfs:Class>

<Class rdf:ID="Thing">
  <rdfs:label>Thing</rdfs:label>
  <unionOf rdf:parseType="Collection">
    <Class rdf:about="#Nothing"/>
    <Class>
      <complementOf rdf:resource="#Nothing"/>
    </Class>
  </unionOf>
</Class>

<Class rdf:ID="Nothing">
  <rdfs:label>Nothing</rdfs:label>
  <complementOf rdf:resource="#Thing"/>
</Class>

<rdf:Property rdf:ID="equivalentClass">
  <rdfs:label>equivalentClass</rdfs:label>
  <rdfs:subPropertyOf rdf:resource="&rdfs;subClassOf"/>
  <rdfs:domain rdf:resource="#Class"/>
  <rdfs:range rdf:resource="#Class"/>
</rdf:Property>

<rdf:Property rdf:ID="disjointWith">
  <rdfs:label>disjointWith</rdfs:label>
  <rdfs:domain rdf:resource="#Class"/>
  <rdfs:range rdf:resource="#Class"/>
</rdf:Property>

<rdf:Property rdf:ID="equivalentProperty">
  <rdfs:label>equivalentProperty</rdfs:label>
  <rdfs:subPropertyOf rdf:resource="&rdfs;subPropertyOf"/>
```

```
</rdf:Property>

<-- ... -->
</rdf:RDF>
```

7 Appendix: URI protocol excerpt

RFC 2396   URI Generic Syntax   August 1998

3. URI Syntactic Components

   The URI syntax is dependent upon the scheme. In general, absolute
   URI are written as follows:

      <scheme>:<scheme-specific-part>

   An absolute URI contains the name of the scheme being used (<scheme>)
   followed by a colon (":") and then a string (the <scheme-specific-
   part>) whose interpretation depends on the scheme.

   The URI syntax does not require that the scheme-specific-part have
   any general structure or set of semantics which is common among all
   URI. However, a subset of URI do share a common syntax for
   representing hierarchical relationships within the namespace. This
   "generic URI" syntax consists of a sequence of four main components:

      <scheme>://<authority><path>?<query>

   each of which, except <scheme>, may be absent from a particular URI.
   For example, some URI schemes do not allow an <authority> component,
   and others do not use a <query> component.

      absoluteURI   = scheme ":" (hier_part | opaque_part)

   URI that are hierarchical in nature use the slash "/" character for
   separating hierarchical components. For some file systems, a "/"
   character (used to denote the hierarchical structure of a URI) is the
   delimiter used to construct a file name hierarchy, and thus the URI
   path will look similar to a file pathname. This does NOT imply that

the resource is a file or that the URI maps to an actual filesystem
pathname.

```
hier_part     = ( net_path | abs_path ) [ "?" query ]

net_path      = "//" authority [ abs_path ]

abs_path      = "/"  path_segments
```

URI that do not make use of the slash "/" character for separating
hierarchical components are considered opaque by the generic URI
parser.

```
opaque_part   = uric_no_slash *uric

uric_no_slash = unreserved | escaped | ";" | "?" | ":" | "@" |
                "&" | "=" | "+" | "$" | ","
```

We use the term <path> to refer to both the <abs_path> and
<opaque_part> constructs, since they are mutually exclusive for any
given URI and can be parsed as a single component.

3.1. Scheme Component

   Just as there are many different methods of access to resources,
   there are a variety of schemes for identifying such resources. The
   URI syntax consists of a sequence of components separated by reserved
   characters, with the first component defining the semantics for the
   remainder of the URI string.

   Scheme names consist of a sequence of characters beginning with a
   lower case letter and followed by any combination of lower case
   letters, digits, plus ("+"), period ("."), or hyphen ("-"). For
   resiliency, programs interpreting URI should treat upper case letters
   as equivalent to lower case in scheme names (e.g., allow "HTTP" as
   well as "http").

```
scheme        = alpha *( alpha | digit | "+" | "-" | "." )
```

   Relative URI references are distinguished from absolute URI in that
   they do not begin with a scheme name. Instead, the scheme is
   inherited from the base URI, as described in Section 5.2.

3.2. Authority Component

   Many URI schemes include a top hierarchical element for a naming
   authority, such that the namespace defined by the remainder of the
   URI is governed by that authority. This authority component is
   typically defined by an Internet-based server or a scheme-specific
   registry of naming authorities.

      authority   = server | reg_name

   The authority component is preceded by a double slash "//" and is
   terminated by the next slash "/", question-mark "?", or by the end of
   the URI. Within the authority component, the characters ";", ":",
   "@", "?", and "/" are reserved.

   An authority component is not required for a URI scheme to make use
   of relative references. A base URI without an authority component
   implies that any relative reference will also be without an authority
   component.

3.2.1. Registry-based Naming Authority

   The structure of a registry-based naming authority is specific to the
   URI scheme, but constrained to the allowed characters for an
   authority component.

      reg_name   = 1*(unreserved | escaped | "$" | "," |
                          ";" | ":" | "@" | "&" | "=" | "+")

3.2.2. Server-based Naming Authority

   URL schemes that involve the direct use of an IP-based protocol to a
   specified server on the Internet use a common syntax for the server
   component of the URI's scheme-specific data:

      <userinfo>@<host>:<port>

   where <userinfo> may consist of a user name and, optionally, scheme-
   specific information about how to gain authorization to access the
   server. The parts "<userinfo>@" and ":<port>" may be omitted.

      server   = [[userinfo "@"] hostport]

   The user information, if present, is followed by a commercial at-sign

"@".

```
   userinfo       = *( unreserved | escaped |
                       ";" | ":" | "&" | "=" | "+" | "$" | "," )
```

Some URL schemes use the format "user:password" in the userinfo
field. This practice is NOT RECOMMENDED, because the passing of
authentication information in clear text (such as URI) has proven to
be a security risk in almost every case where it has been used.

The host is a domain name of a network host, or its IPv4 address as a
set of four decimal digit groups separated by ".". Literal IPv6
addresses are not supported.

```
   hostport       = host [ ":" port ]
   host           = hostname | IPv4address
   hostname       = *( domainlabel "." ) toplabel [ "." ]
   domainlabel    = alphanum | alphanum *( alphanum | "-" ) alphanum
   toplabel       = alpha | alpha *( alphanum | "-" ) alphanum
   IPv4address    = 1*digit "." 1*digit "." 1*digit "." 1*digit
   port           = *digit
```

Hostnames take the form described in Section 3 of [RFC1034] and
Section 2.1 of [RFC1123]: a sequence of domain labels separated by
".", each domain label starting and ending with an alphanumeric
character and possibly also containing "-" characters. The rightmost
domain label of a fully qualified domain name will never start with a
digit, thus syntactically distinguishing domain names from IPv4
addresses, and may be followed by a single "." if it is necessary to
distinguish between the complete domain name and any local domain.
To actually be "Uniform" as a resource locator, a URL hostname should
be a fully qualified domain name. In practice, however, the host
component may be a local domain literal.

   Note: A suitable representation for including a literal IPv6
   address as the host part of a URL is desired, but has not yet been
   determined or implemented in practice.

The port is the network port number for the server. Most schemes
designate protocols that have a default port number. Another port
number may optionally be supplied, in decimal, separated from the
host by a colon. If the port is omitted, the default port number is
assumed.

3.3. Path Component

The path component contains data, specific to the authority (or the scheme if there is no authority component), identifying the resource within the scope of that scheme and authority.