

Cross-repository Integration with RDF and OWL

James Howison
and Kevin Crowston



Hi everyone,
hope the workshop goes well.
Wish I could be there.
Think of me, though, I'm at home
trying to finish this dissertation :)
Thanks to Kevin for presenting.

Cheers,
James

Same subject; different representations

- Repositories of Repositories (RoRs)
 - FLOSSmole, FLOSSmetrics, SRDA (Notre Dame), DOAPspace
- All contain data about same real-world entities
- Divergent representations and access formats
- Local namespaces (SQL tables and columns)

Approaches to linking

- Replicate and convert
 - Gather data from different locations
 - Convert to unified format for querying
- Federate and relate
 - Data stays at current location; in current vocabularies
 - Assess and declare relationships between the data

60 second intro to RDF

- Resource Description Format
- Data composed of Statements:
 - Object, Key, Value (Subject, Predicate, Object)
 - James doesPresentation crossRepositoryLinking .
 - FLOSSmole hasOldName OSSmole .
- Each element in a statement is a URI
 - URI as *name*, URL as *locator*
 - URIs great for FLOSS data as it is collected on the web
 - URIs abbreviated for readability:
 - @prefix rdf: <<http://www.w3.org/1999/02/22-rdf-syntax-ns#>>

- Data-like statements (about Resources/Individuals)
 - Classes of Resources and Properties to link Resources

```
ex:myCar rdf:type      ex:Car ;  
          ex:hasColor  color:red .
```

- Schema-like statements (RDF-Schema & OWL)
 - Relationships between Classes
 - ex:Car rdfs:isSubClassOf ex:Vehicle .
 - Definition of Classes based on restrictions

```
ex:RedThings rdf:type owl:Class ;  
              owl:equivalentClass [ rdf:type owl:Restriction ;  
                                     owl:onProperty :hasColor ;  
                                     owl:hasValue color:red . ] .
```

Meaning: **“All things that have color:red as the value of their :hasColor property are RedThings”**

Reasoning

- Logical *entailments* from Schema-like and Data-like statements available through a Reasoner.
 - `ex:myCar rdf:type ex:Vehicle .`
 - `ex:myCar rdf:type ex:RedThings .`
- Reasoners have capabilities; essentially a set of rdf, rdfs and owl vocabulary which they understand and can draw conclusions from.

Statements for data linking

- URIs are only handles; the same thing can have multiple handles

```
<http://gaim.sf.net> rdf:type my:Project ;  
    my:has_unixname "gaim" .
```

```
<http://sf.net/projects/gaim> your:foundedAt "2001-08-17T02:20:34Z"^^xsd:dateTime .
```

- owl:sameAs

```
<http://gaim.sf.net> owl:sameAs <http://sf.net/projects/gaim> .  
==>
```

```
<http://gaim.sf.net> your:foundedAt "2001-08-17T02:20:34Z"^^xsd:dateTime ;  
    rdf:type my:Project ;  
    my:has_unixname "gaim" .
```

- owl:sameAs works both ways

```
<http://sf.net/projects/gaim> your:foundedAt "2001-08-17T02:20:34Z"^^xsd:dateTime ;  
    rdf:type my:Project ;  
    my:has_unixname "gaim" .
```

- owl:sameAs has the *entailment* that both names now access all the properties of each name; they are synonyms.

Can also link vocabulary-type statements

- **owl:equivalentClass**

```
ex:someProject rdf:type your:Project .  
your:Project owl:equivalentClass my:Project .  
==>  
ex:someProject rdf:type my:Project .
```

- **Same available as owl:equivalentProperty**

- **rdfs:subClassOf**

```
your:Project rdfs:subClassOf doap:Project .  
my:Project rdfs:subClassOf doap:Project .  
ex:someProject rdf:type your:Project .  
==>  
ex:someProject rdf:type doap:Project .
```

but NOT: ex:someProject rdf:type my:Project .

In short, that's it

- Express semantic relationships between different repositories
 - Individual and Vocabulary Equality, class membership etc
- Reasoners go to work and produce the implied statements
- Then one can query across repositories, using the same vocabulary

Storing and Querying RDF

- Numerous serialization formats
 - XML, Turtle/N3 (plain text), Manchester Abstract Syntax
 - Equivalent for our purposes
- Triple-stores
 - In-Memory, DB backed, Btree backed

Querying RDF

- SPARQL
 - Querying via pattern matching

```
SELECT ?var
WHERE {
    ?var rdf:type my:Project .
}
```
- Protocol for access via HTTP GET with query encoded into URL; returns XML formatted results
- FILTER (eg ?time <= "2004-01-01T00:00:00Z"^^xsd:dateTime)
- OPTIONAL, UNION
- Extensions for COUNT etc (although not usually owl:sameAs aware)
- SPARQL standardized in Jan 2008; >5 different servers with client libraries in Java, Python, Perl, Ruby, .NET ...

From Relational DBs to RDF

- Procedural:
 - Scripts that query the DB and produce RDF
 - One-time conversion
- Declarative:
 - Mapping language: D2RQ
 - Dynamic conversion
 - Offers SPARQL frontend with RDB backend
 - Can also 'dump' RDF for improved performance

Three approaches to linking

- Namespace as origin metadata
 1. Retain local namespace; use equivalence statements between them
 2. Create meta-namespace; use equivalence statements between local and meta namespaces
- Create meta-namespace; map directly from RDBs to meta-namespace
 3. Build explicit provenance statements
 - Reification (Statement as object with attributes)
 - Named Graph (Set of Statements as object with attributes)
 - OWL 2.0 Annotation properties

Ground vs Theory-laden

- Much that we agree on (ground data):
 - A project name is a project name
 - An email is an email
 - A founding date is a founding date
- Many (interesting) statements are theory-laden
 - A Developer is one that has CVS access
 - A Developer is one that is listed on homepage
 - A Successful project has X releases and X downloads
 - This set of emails depicts Collaboration
- RDF allows layering on top of shared ground data. Theory-laden definitions are shareable and alterable.

Flosscomms (fc:)

- Studying of communications of FLOSS projects
 - Mailing Lists & Forums from FLOSSmole
 - Trackers & Releases from SRDA (Notre Dame)
 - SVN logs directly from SVN XML (CVSAnalY didn't store log message content)
- Direct Procedural conversion to meta-vocabulary
- Layering of theory-laden statements about document genre attributes, including Roles
- Archives converted to *Events* (of different kinds), occurring in *Venues*, associated with *Identifiers* which identify *Participants*.
- <<http://floss.syr.edu/ontologies/2008/flosscomms-sketch.pdf>>

Summary Stats

How many communications in **any** archived Gaim forums in 2002?

```
SELECT COUNT(DISTINCT ?event)
WHERE {
  ?event fc:isEventOf <http://sf.net/projects/gaim> ;
        fc:hasTime ?time .
  FILTER (?time >= "2002-01-01T00:00:00Z"^^xsd:dateTime &&
          ?time < "2003-01-01T00:00:00Z"^^xsd:dateTime )
}
==> "9316"^^xsd:integer
```

How many communications in archived Gaim **mailing lists** in 2002?

```
SELECT COUNT(DISTINCT ?event)
WHERE {
  ?event fc:isEventOf <http://sf.net/projects/gaim> ;
        fc:hasTime ?time ;
        rdf:type fc:MailingListEvent .
  FILTER (?time >= "2002-01-01T00:00:00Z"^^xsd:dateTime &&
          ?time < "2003-01-01T00:00:00Z"^^xsd:dateTime )
}
==> "520"^^xsd:integer
```

- **Fire & Gaim**

- Total Events: 152,362 (Fire: 20,482 , Gaim: 131,879)
- Total Venues: 24 of 5 different types (Fire: 10, Gaim: 14)
 - MailingLists, SVN, Forums, ReleaseNotes, Trackers
- Total Identifiers: 322,408, with 57,541 distinct strings
 - RealNames, SfUserName, SfUserID, SfAbbrevEmailAddress
- Total Releases: 189 (Fire: 39, Gaim: 150)
 - Multiple packages for each project

Classing Developers by how many Development Events

- After Entity Resolution (matching RealNames, Emails etc) to generate Participants
 - fc:DevelopmentEvent owl:unionOf { fc:SvnCommitEvent, fc:SoftwareReleaseEvent, fc:TrackerAdminEvent }
 - fc:Developer rdfs:subClassOf fc:Participant .
- Create restriction that says that a Participant is a fc:Developer if they didPerform $> X$ fc:DevelopmentEvents
- Potentially limit to “in last 6 months” (may require procedural intervention)

Challenges

- Mapping between vocabularies is (relatively) easy; Mapping between differing data models is harder (D2RQ helps here)
- Reasoner performance, especially owl:sameAs is problematic (EC2 helps here)
- Federation querying technologies are in infancy; replication still a necessary approach
- Understanding, using and extending 'well-known' vocabularies, like DOAP, FOAF, EvoOnt and Event Ontology

Conclusions

- Linking various repositories is hard but achievable
- RDF integration offers a way for Repositories to collaborate without enforced standardization
- Expose your data to RDF; setup a SPARQL frontend
- Take, and extend the flosscomms vocab, together with DOAP and SIOC.
- <http://floss.syr.edu/ontologies/2008/>