Cross-repository data linking with RDF and OWL

Towards common ontologies for representing FLOSS data[*]

James Howison
School of Information Studies
Syracuse University, NY, USA
jhowison@syr.edu

ABSTRACT

This paper provides an approach to the problem of integrating data from multiple research repositories for FLOSS data. It introduces semantic web technologies (RDF, OWL, OWL-DL reasoners and SPARQL) to argue that these are useful for building shared research infrastructure. The paper illustrates its point by describing parts of an ontology developed for the integration and analysis of project communications drawn from FLOSSmole, the Notre Dame archive and direct collection of data. RDF vocabularies provide a way to agree on things we agree about as well as a way to be clearer about ways in which we disagree.

1. INTRODUCTION

Work in the last few years has produced large and growing public repositories for research data on free and open source software development, usually called Repositories of Repositories [1]. Repositories such as the Notre Dame Sourceforge repository, FLOSSmetrics (which succeeds CVSanalY) and FLOSSmole have adapted the archives of open source development for the use of researchers. They have been relatively useful for saving researchers time (by avoiding redundant collection) and protecting open source projects against the pernicious effects of automated collection, and current work on sharing analyses ought to improve their usefulness (eg [3]).

However these data are side effects of activity; they are not organized in ways that assist researchers. The Notre Dame database dumps, for example, are designed to run a high volume website, while the FLOSSmole datasets are limited to public facing data. Similarly the repositories, especially FLOSSmole and FLOSSmetrics, are increasingly collecting data from multiple repositories, as research interest (and the open source community) extends away from Sourceforge. The data contained in the repositories are expressed in the terms of the particular archive in which they were stored and little or no documentation about the common semantics of the various fields in the databases, or, just as importantly, semantic differences between similarly named fields. The recent NSF Workshop on Free/Open Source Software Repositories and Research Infrastructures (FOSSRRI 2008) explicitly called for ways of integrating separate repositories, while acknowledging that the repositories will and most likely should remain institutionally separate, for reasons including different funding sources and academic credit.

So far FLOSS repositories have taken two approaches to this issue. The first is documentation, either through commented SQL schema (FLOSSmole) or through a wiki, as well as through recurrent discussions on mailing lists (such as ossmole-discuss). These efforts have been ad-hoc, free-text and, if we are to be honest and judge by the traffic on our mailing lists, not very successful.

This short paper describes a set of technologies and research work which promise improved semantic understanding of commonalities and differences in data held in public repositories. Formal, machine-readable mappings which describe the semantics of the data available in RoRs are needed, and the set of technologies to be described below are now mature enough to provide them. Once the technologies are introduced the paper turns to describing an initial, partial effort to produce a useful vocabulary and some of the operations it supports.

2. RDF/OWL BACKGROUND

This section introduces a set of technologies best known as "semantic web technologies". They include a data representation format (RDF), logical languages for describing types and relations between types (RDFS and OWL), automated reasoning facilities (reasoners) and a query language (SPARQL). The space here is obviously too limited (and our understanding too preliminary) to provide a comprehensive introduction, rather we attempt to whet the appetite.

2.1 What is RDF?

RDF is a way of representing knowledge. Knowledge in RDF is represented by a set of statements, known as triples, which can be thought of as subject, key and value, but are more commonly referred to as subject, predicate and object.[1].

[*]Thanks to Andrea Wiggins and Kevin Crowston for their comments and support. This work was supported by NSF Grant 07–08437

[1]It is crucial to realize that RDF is separate from its serialization, which can be in a number of different XML formats or in plain text formats, known as Turtle, n3 or N-Triple. The statements in this paper are written in a very limited

WoPDaSD 2008 IFIP 2.13, Milano

However, before one can talk about things in a formal way, one needs a way to refer to those things. Things in RDF are identified by URIs, which are convenient due to their dual functions as locally controlled unique namespaces and their function as locators. For example one could write the knowledge that my name is James Howison (using my personal website address as a handle for me as a concept) [2]

```
<http://james.howison.name/> foaf:name "James Howison" .
```

A major primary advantage of RDF over relational databases is the use of URIs, which are a great combination of a globally recognized namespace while being locally controlled. For example many databases may have a column in a table named `project_name` but there is no built in way to differentiate similarly named fields from each other, or to be clear on how one should interpret the contents. A concrete example is that FLOSSmole stores Sourceforge project names in a column called `projects.unixname`, while the Notre Dame Sourceforge dumps store Sourceforge project names in a table named `groups.unix_group_name`. By using URIs to name fields one would at least be able to tell that different organizations defined these, and, using the technologies below, find a formal, machine readable way to map between them. The point is not necessarily to force either project to alter their data representations, but to gain a way to formally describe and therefore use these fields.

Global namespacing is extremely valuable when it comes to merging and combining datasets; sharply reducing the potential for namespace collisions (URIs are still completely different, even if they have similar endings). The use of URIs also provides a ready made way to identify and aggregate statements about particular entities (because they will use the same URI). This makes RDF data ideal for uniting knowledge from diverse sources, such as across different repositories.

The promise of semantic web technologies for cross-repository research, therefore, is to elaborate a vocabulary appropriate to representing the many different types and properties inherent in the FLOSS datasets, and to describe appropriate mappings between them. First, however, this paper provides a brief introduction to the technologies, since they are not yet well known.

2.2 What do RDFS and OWL add?

A very important and useful type of statement in RDF is the rdf:type statement. This allows ontologies to specify that a particular resource (a URI) is of a particular type, where that type is also referred to as a URI. Thus one can make the statement that

```
sf_proj:gaim rdf:type ex:OpenSourceProject .
```

Statements that particular URIs (often called 'Individuals' or 'Resources') are of a type opens up the possibility of making inferences about that URI based on its type. This is the realm of RDFS (RDF Schema) and OWL (Web Ontology Language). These are vocabularies, themselves expressed using RDF, which provide terms with specific semantic meanings. These meanings are understood by reasoners, which are software programs which use the statements to generate additional statements ('entailments' or 'inferences'), based on a set of given statements (often called 'ground' data).

For example using RDFS one can state that an OpenSourceDeveloper is a type of Person and therefore infer that any URI described as an OpenSourceDeveloper must also be a Person. Similarly one can infer knowledge about properties, such as an inverse relationship (other property relationships include transitivity):

```
ex_data:someParent  ex:isParentOf  ex_data:someChild .
ex:isParentOf       owl:inverseOf  ex:isChildOf .
```

gives

```
ex_data:someChild  ex:isChildOf  ex_data:someParent .
```

OWL takes this type of reasoning further and allows one to infer knowledge using a powerful set of semantics such as stating that a Person is an OpenSourceDeveloper if—and only if—they have checked code into the repository of an OpenSourceProject. The point, of course, is not to make this debatable concrete statement, but to have a way to formally describe and refer to such definitions. Finally SWRL (Semantic Web Rule Language) offers a declarative way to increase the expressivity of inference beyond the class-based reasoning of OWL.

RDF, together with RDFS and OWL and their reasoners, therefore provides a way to build a knowledge base, statement by statement. In the RDF world there is no formal distinction between the schema and data and both together are called ontologies (or graphs), but as we transitioned from relational databases to RDF we found it a useful distinction. A Vocabulary, built from statements made using the RDF, RDFS and OWL terms, is like a schema. This vocabulary can be used to make statements about URIs, which can be outside the vocabulary namespace, whether they are resolvable actual web addresses or not. These statements are more like data and the URIs are known as Individuals (sometimes Resources or Instances). To avoid this confusion this paper will refer to the schema-like statements as a *vocabulary*, and when data-like statements are included the whole set of statements will be called a *knowledge base*.

2.3 Federating datasets with RDF and OWL

Data about things comes from many places, and those that publish the data use many different sets of terms to talk about it. Some of these terms are exactly equivalent, most are different, sometimes obviously, sometimes subtly. The tools of global namespaces, together with some of the vocabulary from RDF, RDFS and OWL provide formal ways to talk about these relationships, even when the terms are from different namespaces. For example, consider two terms to talk about people's names: "last name" and "family name". These are related concepts, but they aren't identical because there are cultures where one's family name is usually presented first. Nonetheless they are both types of names, and knowing that might sometimes be enough.

To express those relationships formally we can imagine two different totally made up namespaces (foo: and bar:):

subset of Turtle (which is itself a subset of n3), eschewing convenient nested notations for reasons of introductory clarity. See a simple Syntax Primer as an appendix

[2] `foaf:name` is an abbreviation of http://xmlns.com/foaf/-0.1/name, which has formally defined semantics (those semantics can, incidentally, be read about by using the URI as a URL).

```
@prefix foo: <http://example.foo.com/foo#>
@prefix bar: <http://example.bar.com/bar#>
```

Each namespace has a term for referring to “last name” and “family name”:

```
foo:lastname    rdf:type  rdf:Property .
bar:familyname  rdf:type  rdf:Property .
```

We can express a particular semantic relationship between these terms using the vocabulary from RDFS. Here we’ll assert that when people speaking with the bar vocabulary use the term “last name”, and when people speaking the foo vocabulary use familyname, we can understand that they are equivalent; which is to say that all `foo:lastname` are the same as `bar:familyname`, and vice versa.

```
foo:lastname owl:equivalentProperty bar:familyname .
```

With the right reasoner, one would now be able to get all of the names by using a query expressed in either `foo:` or `bar:` vocabulary.

Sometimes, however, one actually wants to retain the original semantics while conveying information about the relatedness of terms. For example one can formalize the idea that these are both types of names, without rendering them equivalent in every way. Here we introduce a third totally different namespace, which we’ll call `ex_vocab` and declare a term in that namespace called `name`. Now we can write:

```
foo:lastname   rdfs:subPropertyOf ex_vocab:name .
bar:familyname rdfs:subPropertyOf ex_vocab:name .
```

Now queries for `bar:familyname` or `foo:lastname` will only return names explicitly declared as such, but queries for ex:name will return names declared with either of those terms.

In such ways RDF provides the ability to federate datasources, either directly or indirectly through higher level concepts. These are, of course, only a few simple examples of ways in which terms might be related.

2.4 Querying using pattern matching: SPARQL

Of course representing and storing data is not much use without the ability to search or query the resulting knowledge bases. The semantic web community, through the w3c, has recently standardized a very useful query language (and protocol) for searching Knowledge Bases. This is called SPARQL. A SPARQL query is a template with some invariant and some variant elements, against which a Knowledge Base can be tested and any matching results returned. For example a list of open source projects could be obtained by finding a set of matches (bindings) for the variable ?project in:

```
SELECT *
WHERE {
 ?project rdf:type ex:OpenSourceProject .
}
```

and assuming more properties, such a query can be extended:

```
SELECT *
WHERE {
  ?project  rdf:type            ex_vocab:OpenSourceProject .
  ?project  ex_vocab:foundedAt  ?founding_date .
  FILTER (
    ?founding_date <= "2000-01-01T00:00:00Z"^^xsd:dateTime
  )
}
```

which would return only those projects whose ex:founding_date was before the start of the year 2000. For a set of statements to match, the ?project variable has to be the exact same URI in each place it is mentioned. Unlike SQL there is no need to specify and manage table joins; there are only statements (triples) and SPARQL implementations offer extremely scalable querying (there are implementations capable of handling 10 billion triples, although see below for scalability issues and solutions.)

2.5 Advantages for FLOSS representation

We believe that the semantic web technologies offer a good way forward to increasing the compatibility and documentation of the public data repositories. Our data already naturally has well-known URIs and our repository projects have domain names and thus natural namespaces for building their vocabularies.

These technologies can also support the layering of meaning that is crucial to FLOSS research. For example the FLOSS data landscape is fortunate to have in it entities with very concrete existence, such as a SVN commit. There is little scope for disagreement that such an event took place (although one might argue about what timezone the timestamp was in).

On the other hand, researchers develop and use theories which go beyond such bland statements. They may want to state that the email—or patterns of email exchanges—represent higher level constructs, such as collaboration. Such further statements could be expressed using the same URI identifiers for the ‘ground’ but more theoretically informed, separate, vocabularies for the higher level constructs.

In this way RDF vocabularies provide a way to agree on things we agree on as well as a way to be clear about ways in which we disagree (or to put it more politically support clear discussions of multiple interpretations).

2.6 Mapping between FLOSSmole and the Notre Dame dataset

Both FLOSSmole and the Notre Dame archive store information about projects in Sourceforge, such as their project unixname (eg gaim or fire). Using the RDF concepts above it is possible to define a formal mapping which captures the understanding that these names refer to the same project, and thereby link any data that each repository has about that project.

The first thing needed is to map each local representation to RDF. This requires a URI scheme for each field in the databases. Since the projects already have functional domain names, these are natural places to begin. While URIs, as names, don’t have to resolve, it is extremely convenient if they do resolve and ideally as close to a page describing their semantics as possible.

FLOSSmole does not yet have individual pages describing its data fields, so the URIs used here do not yet resolve. The database that holds the unixname information is called “ossmole_merged”, the table is “projects” and the field (or column) is “unixname”. Thus we can say that FLOSSmole defines a rdf:Property called:

```
<http://ossmole.sf.net/dataset/ossmole_merged/projects#unixname>
  or
fm:unixname
```

By contrast the Notre Dame archive holds data in different databases and tables. By inspection one can determine that in the database schema called sf0208 (February 2008) there is a table called groups and a field called unix_group_name. Notre Dame's archive does have pages which provide basic descriptions of the table schemas, so we can use those direct URLs (even though they are password protected).

```
<https://zerlot.cse.nd.edu/cgi-bin/treq.pl?uschema=sf0208&utable=groups#unix_group_name>
  or
nd:unix_group_name
```

Using such URIs one is able to convert the databases into a set of RDF statements. For convenience we will use the Sourceforge project home page as the handle for the Project, abbreviated according to the prefix in the Syntax Primer to sf_proj:gaim. We can now show a statement from both FLOSSmole and the Notre Dame archive:

```
sf_proj:gaim   fm:unixname        "gaim"^^xsd:string .
sf_proj:gaim   nd:unix_group_name "gaim"^^xsd:string .
```

Once the data is represented in RDF we are able to formally describe the mapping. Here we will use a 'meta' namespace, which endeavors to describe the concepts at play in a repository neutral language, thus preserving but enhancing the original semantics. We will use the fc: namespace (see Syntax Primer):

```
fc:sf_unixname rdf:type rdf:Property .
```

And then declare that our two existing properties are subproperties of this:

```
fm:unixname        rdfs:subPropertyOf  fc:sf_unixname .
nd:unix_group_name rdfs:subPropertyOf  fc:sf_unixname .
```

All queries for fc:sf_unixname will now return any projects from either FLOSSmole or NotreDame. It is also possible, using a combination of OWL statements, like owl:InverseFunctionalProperty (which means that there can only be a single value for a property) to discover that the project entities are the same, that they are owl:sameAs, such that all the statements about either are now statements about each.

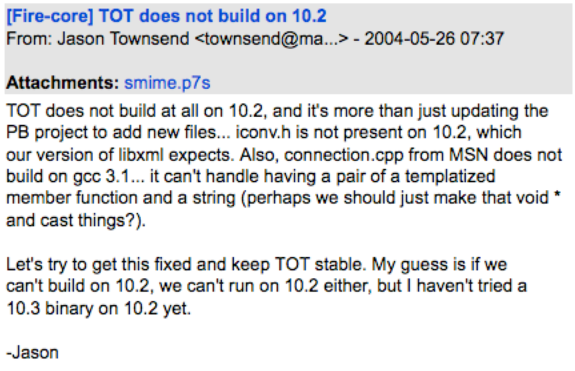
3. THE FLOSSCOMMS VOCABULAR

The previous section introduced the idea of a meta-ontology able to express semantic similarity between data in different repositories. We have begun building such an ontology to integrate communications data from FLOSSmole, the Notre Dame repository and directly collected SVN log messages (unfortunately CVSanalY did not store the content of the log message, or it would have been used). This vocabulary is called flosscomms (floss communications).

The research project is to discover genres of communication and to observe the manner in which community work wends its way through different communication venues and forms; data from multiple different communication venues is thus crucial. The ontology introduced here allows the representation of communication data from Mailing lists, Forums, SVN log messages, Release Notes and Issue Trackers. This is accomplished without losing the important typing information that the medium provides. For those familiar with Object oriented programming the task is similar to ensuring that each object can respond to the same set of interfaces; however the semantic web technologies offer a separation of such semantics from a particular implementation (and global namespaces).

The ontology allows the simple execution of queries to answer questions such as "What is the relative media use over time?", "Do some venues have faster communication than others?", "Do some venues have longer discussions than others?". When additional entity resolution work is done to identify individual participants, one can ask questions like "Do core developers have consistent, regular participation, regardless of venue, or do they cycle in and out?".

Below we outline an example representation of an email to a mailing list and provide some comments about other communication types; however there is not sufficient space to cover the entire ontology. The ontology, its documentation and some sample RDF data are available online[3]



[Fire-core] TOT does not build on 10.2
From: Jason Townsend <townsend@ma...> - 2004-05-26 07:37

Attachments: smime.p7s

TOT does not build at all on 10.2, and it's more than just updating the PB project to add new files... iconv.h is not present on 10.2, which our version of libxml expects. Also, connection.cpp from MSN does not build on gcc 3.1... it can't handle having a pair of a templatized member function and a string (perhaps we should just make that void * and cast things?).

Let's try to get this fixed and keep TOT stable. My guess is if we can't build on 10.2, we can't run on 10.2 either, but I haven't tried a 10.3 binary on 10.2 yet.

-Jason

Figure 1: An example email from Sourceforge

Figure 1 shows an email from a Sourceforge mailing list (which has been collected by FLOSSmole). There is a lot of semantic information here, but we will only consider a few elements: firstly the email comes with a ready-made, if long, URI as a handle in RDF (the destination of the linked subject). We have data about time and date at which it was sent, as well as two identifiers referring to the person that sent the email: a real name and an abbreviated email address. We also know that the email was the start of a thread and it was sent to a particular mailing list (fire-core). The flosscomms ontology allows us to represent this data in RDF. The full URI for the email is very long (it uses Message ID header from the original email), so we will abbreviate it as sf_email:1. Note that we can create unique URIs for things on this page by using the # separator at the end of the URI; these are purely for convenience in reading related URIs, adding something in that way creates a completely different URI.

The core concept in the ontology is that of an Event, which is simply something that occurs at a particular time (unlogged in date times on sourceforge are in the PST timezone, hence -0800).

```
sf_email:1 rdf:type    fc:MailingListEvent ;
           fc:hasTime "2004-05-26T07:37:00-0800"^^xsd:dateTime .
```

Events are associated with an Identifier, such as a username or email address, using the property fc:hasPerformerIdentifier.

```
sf_email:1 fc:hasPerformerIdentifier sf_email:1#name ;
                                     sf_email:2#email .
```

[3] http://floss.syr.edu/ontologies/2008/ (note how the ontologies URI becomes a convenient location to publish it, and documentation about it).

```
sf_email:1#name  rdf:type      fc:RealNameIdentifier ;
                 fc:hasContent "Jason Townsend"^^xsd:string .

sf_email:2#email rdf:type      fc:SfAbbrevEmailAddress ;
                 fc:hasContent "townsend@ma..."^^xsd:string .
```

where both fc:RealNameIdentifier and fc:SfAbbrevEmailAddress are sub-classes of fc:Identifier.

Note here that by using a specific Class for the abbreviated email addresses, we make possible intelligent reasoning which takes into account the special abbreviation style that Sourceforge Mailing lists use (which limits certain kinds of entity matching).

Some Events—those we were particularly interested in while builing our model—include textual content that is communicated; we model the content as a Document that is associated with an Event; some Events, such as Release Notes, have two Documents associated with them (Change List and Release Note), similarly one might argue that an SVN commit has at least two Documents: the patchset and the log message. For our email example:

```
sf_email:1 fc:hasDocument sf_email:1#doc .

sf_email:1#doc rdf:type fc:EmailAddressContent ;
  fc:hasSubject "TOT does not build on 10.2"^^xsd:string ;
  fc:hasContent "TOT does not build at all ..."^^xsd:string .
```

where fc:EmailAddressContent is a sub-class of fc:Document.

Events which include Documents are defined as a special sub-type of Event called a CommunicationEvent. Here we use a statement drawing on the OWL vocabulary (owl:someValuesFrom) that says ”Any Event that has any associated Documents is also a CommunicationEvent”. That, when used with an appropriate reasoner allows us to infer (in addition to being a fc:Event and a fc:MailingListEvent) that sf_email:1 is also a fc:CommunicationEvent.

Events are accessible to us because they ended up in an archive of some type, usually because they were conveyed in a particular Venue, such as the developer's mailing list, or even in a Venue such as a SourceCodeRepository, specifically a SvnRepository, or even the a FileReleaseSystem like the Sourceforge release system. In our example the mailing list for the message is the fire-core mailing list. Sourceforge provides a useful URI for this mailing list (which we will abbreviate as sf_email:fire-core):

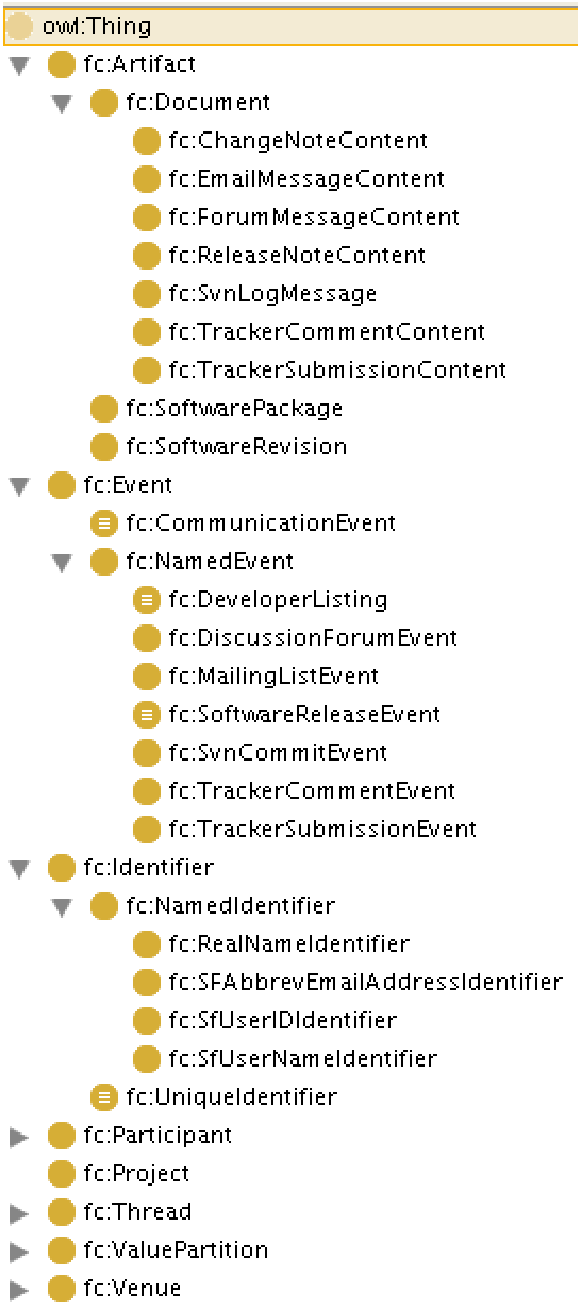
```
<http://sourceforge.net/mailarchive/forum.php?forum_name=fire-core>
  rdf:type
    fc:MailingList .
```

Some Venues are threaded venues, meaning that there is an intermediate level of encapsulation. In our example the email is the first in a thread, so we can name the thread after it (Sourceforge actually has full URIs to refer to threads, which do use the Message ID of the first message in the thread)

```
sf_email:1#thread  rdf:type       fc:Thread ;
                   fc:isThreadOf  sf_email:fire-core .
```

And we can state that the particular email is part of this thread, using the property fc:hasEvent.

```
sf_email:1#thread fc:hasEvent sf_email:1 .
```



```
owl:Thing
▼ fc:Artifact
    ▼ fc:Document
        fc:ChangeNoteContent
        fc:EmailMessageContent
        fc:ForumMessageContent
        fc:ReleaseNoteContent
        fc:SvnLogMessage
        fc:TrackerCommentContent
        fc:TrackerSubmissionContent
      fc:SoftwarePackage
      fc:SoftwareRevision
▼ fc:Event
      fc:CommunicationEvent
    ▼ fc:NamedEvent
        fc:DeveloperListing
        fc:DiscussionForumEvent
        fc:MailingListEvent
        fc:SoftwareReleaseEvent
        fc:SvnCommitEvent
        fc:TrackerCommentEvent
        fc:TrackerSubmissionEvent
▼ fc:Identifier
    ▼ fc:NamedIdentifier
        fc:RealNameIdentifier
        fc:SFAbbrevEmailAddressIdentifier
        fc:SfUserIDIdentifier
        fc:SfUserNameIdentifier
      fc:UniqueIdentifier
▶ fc:Participant
  fc:Project
▶ fc:Thread
▶ fc:ValuePartition
▶ fc:Venue
```

Figure 2: This shows part of the full class hierarchy for flosscomms, focusing on Events, Documents and Identifiers. It is a screenshot from Protege, an ontology editor.

Of course the flosscomms vocabulary defines inverse relationships, so if we use an appropriate reasoner, we will infer inverse the reverse relationships too:

```
sf_email:1 fc:isEventOf sf_email:1#thread .
sf_email:1#thread fc:isThreadOf sf_email:fire-core .
```

Finally we wish to state that this mailing list belongs to a particular project, which has as its unixname, ”fire”:

```
sf_proj:fire  rdf:type        fc:Project ;
              fc:sf_unixname  "fire"^^xsd:string ;
              fc:hasVenue     sf_email:fire-core .
```

So far this knowledge base is nice and accurate, but is a little clumsy for querying. For example to get all the Events associated with the fire project one would have to query using a SELECT clause like:

```
SELECT ?event
WHERE {
  ?event  fc:isEventOf  ?thread .
  ?thread fc:isThreadOf ?venue .
  ?venue  fc:isVenueOf  sf_proj:fire .
}
```

when it would be much more convenient (and totally semantically true) to query for every event, regardless of its venue (or thread) using:

```
SELECT ?event
WHERE {
  ?event fc:isEventOf sf_proj:fire .
}
```

This can be achieved in a number of ways. Currently an extension to the flosscomms ontology uses the SWRL (Semantic Web Rule Language) but one could also use OWL 2.0’s Role Chains to state that “Any Venue with a Thread has all the Events of its Threads” and “Any Project with a Venue has all the Events of its Venues”; and with an appropriate reasoner additional RDF statements would be generated for each event such that the shorter query above would work. Since some types of Events, such as Release Notes, are related to their projects by fairly long links (a Project produces a Package which has a Revision which is released in a ReleaseEvent, which has an associated ChangeNote (phew!)), declaring such relationships is very convenient and is a real payoff for the modeling work.

3.1 A note on Entity Resolution

Conceptually Events are (usually) performed by a person, and that person can be modeled as a Participant in the project. Yet when we encounter the archives we do not directly known who performed the event; rather we find an Identifier, such as an email address (or perhaps a set of identifiers such as an SfUserId or SfUserName). However usually we want to know which actual person performed a set of activities, so we want to link the Identifiers to a Participant, using fc:isIdentifiedBy, and thereby Participants to Events, using fc:didPerform.

That is simple enough so far. However some types of Identifiers are unique, while others are not. For example it is legitimate to assume that two events, taken from a Sourceforge archive, associated with the same Sourceforge user name, were in fact performed by the same person. On the other hand it would be foolhardy to assume that two events associated with the same real name part of an email address (such as ”John” or ”Pradeep”) were necessarily the same person. We simply don’t know enough to make that assertion. This knowledge can be expreseed in a number of ways, including creating sub-types of Identifier:

```
fc:SfUserNameIdentifier rdfs:subClassOf  sf:UniqueIdentifier .
fc:RealNameIdentifier   rdfs:subClassOf  sf:NonUniqueIdentifier .

ex:someIdent rdf:type fc:SfUserNameIdentifier ;
             fc:hasContent "johnuser"^^xsd:string .

ex_data:someOtherIdent  rdf:type fc:RealNameIdentifier ;
                        fc:hasContent "John"^^xsd:string .
```

This type of knowledge is very useful for managing what type of assertions can be made about whether or not the same Participant can be associated with a set of Identifiers, and could therefore be understood to be the same Entity. For example RealNameIdentifiers, while not UniqueIdentifiers, are not devoid of semantics and there are algorithms for matching (although we’ve found that the open source domain throws up particular challenges in that regard). In any case all that can be said here is that entity matching can leverage such semantics when they are presented using RDF typing.

However entity matching is undertaken—whether using UniqueIdentifiers or through fuzzy matching of NonUniqueIdentifiers—when Participants are found to be linked this can be expressed can be expressed in RDF using owl:sameAs, which was introduced above:

```
ex_data:partFromEmail  owl:sameAs ex_data:partFromSVN .
```

This is a very powerful statement. If that is included in a set of statements passed to a reasoner, the reasoner will generate statements such that anything that is true of ex_data:partFromEmail is also true of ex_data:partFromSVN (and vice versa). In this manner one can accomplish Entity Resolution using RDF statements and link Participants across differing Venues.

Such entity matching enables really interesting things like classifying Participants into Core, Peripheral or Active User, based on the types of Events they have undertaken (which is similar to the achievements reported in [2]). Of course such definitions would be entirely debatable but by declaring new namespaces for them and publishing the definitions or workflows that create the classifications we would better know where we agree and be clearer about where we do not. The point is that an RDF ontology allows the representation of underlying ground data from different sources using the same vocabulary and thus supports clear definitions of research concepts, as research proceeds.

3.2 Future additions

The flosscomms vocabulary is a work in progress but is particularly tailored to Communication Events. Clearly there are other types of events which are of great interest, such as when a project is founded and periodic developer listings, or even leadership changes (however theoretically defined), to name just a few. One would also want to develop vocabularies to represent data about project performance, such as the ubiquitous measures of downloads and pageviews. There are many different sources of such data, but using a full URI to name the datapoint enables researchers to be clear about what technique they are using (for example figures scraped from Sourceforge vs analysis of the logs of a Linux distribution).

3.2.1 Integration with existing ontologies

Close readers will have noted that all the Classes and Properties discussed in this paper are expressed in the same namespace (fc). This was done mainly because we were learning while doing and wanted to have predictability in reasoning (without simultaneously learning the implications of other vocabularies, especially their interaction). However, there are a large number of generic and specific available vocabularies that ought to be integrated with the flosscomms and its successors vocabularies (after all open source projects are not the only things with email lists or names!).

There are generic ontologies, such as Friend of a Friend (foaf), which provide vocabulary for people and relationships, as well as ontologies for things such as temporal periods (temporal) from Instants to Intervals. As researchers work with more theoretically informed concepts they may wish to use statements from the SKOS vocabulary (this is particularly apposite for hierarchies of Codes for content analysis).

More specifically there are existing domain ontologies that ought to be used. Description of a Project (DOAP) is already supporting a range of statements specifically about open source projects (such as the location of their bug tracker and release system as well as old names etc). DOAP increasingly supports collaboration amongst systems designed to help discover open source projects and by using real-world URIs where-ever possible our data repositories can become part of this semantic web.

Similarly there are existing ontologies for describing Issue Tracking systems (such as EvoOnt). There is also the SIOC ontology, which provides vocabulary for describing online communities and includes quite similar things to flosscomms. Finally researchers other than us have already developed OWL ontologies for describing open source artifacts, such as the padme.owl[4] ontology, developed by Chris Jensen.

The beauty of RDF and OWL, however, is that the terms in the flosscomms vocabulary, or future vocabularies, do not necessarily have to be changed. Rather as one discovers the semantic relationships between the vocabularies one simply adds statements conveying that understanding and allows reasoners to use those to draw conclusions. For example if one believes that the flosscomms concept of a Project is identical to the DOAP concept of a project (ie that all fc:Project are also doap:Project) then one can transform a Knowledge Base simply by adding the single statement:

```
fc:Project owl:equivalentClass doap:Project .
```

And with the appropriate reasoner any RDF data expressed in either vocabulary will be accessible.

3.3 Performance Issues

The picture painted above, especially the use of reasoners and SWRL rules, perhaps paints too rosy a picture of the state of the art in Semantic Web technologies, especially using open source components.

As a data point, we have collected all the CommunicationEvents from FLOSSMole (Mailing Lists), Notre Dame (Tracker Items, Forums, Release Notes) and SVN logs for two FLOSS projects (gaim and fire) and represented the data using the ontology described in this paper. In total there are over 158,000 Events, which before reasoning generate about 2.5 million RDF statements. After reasoning this expands to 9.5 million RDF statements. This is just the data from two open source projects.

Currently we store the realized (ie after reasoning) ontology using the SDB store from the open source Jena project, using a MySQL backing store. Querying this using SPARQL is fine, performance wise (on a Macbook with 3G ram), returning results such as the Events for a month (that is using a FILTER) in sub 10 seconds and almost instantly using a 1G query cache for MySQL.

However we did face significant performance issues when applying a reasoner to our knowledge base. This is because the reasoners available (such as Pellet or the Jena set of reasoners) are very RAM hungry; they hold the entire graph (set of statements) in memory while generating the inferences. Needless to say this was too much for a laptop, even with 3G of RAM. After much experimentation we shifted this component of the work to Amazon's EC2, which provides Linux instances with 15G for 80 cents an hour. Of course others could and will use supercomputing resources provided by groups such as NSF's TeraGrid; but the EC2 option is a low overhead, quick start, approach. Once the reasoning is complete, the database can be transferred back to local operation (or made available for community querying). In any case this type of reasoning is a run-once, query-many type of operation. That said as the community experiments with new definitions, based on this ground data, substantial memory resources will be required.

Scalability of the semantic web technologies to "web-scale" (ie able to encompass data available throughout the world wide web) is a very active area of research. To give some idea of the current capabilities of commercial products, the AllegoGraph system is free of charge for users up to 50 million statements, and they claim high performance with up to 10 billion triples.

4. CONCLUSIONS

The significant progress made in collecting and making available Repositories of Repositories for research on FLOSS and its development can be extended through the use of semantic web technologies. Specifically it is possible to represent data drawn from multiple RoR (and directly from the field) using an aligned vocabulary. Such vocabularies can then form the 'ground' data for analyses such as Entity Resolution. Semantic Web technologies are well suited for FLOSS data, which already draws on URIs. Importantly they do not demand pre-agreement between researchers, since relationships between vocabularies can be expressed and they offer integration of multiple sources of data.

APPENDIX

A. SYNTAX PRIMER

The example RDF in this paper are written with a simple syntax that is valid Turtle (and n3). Each element is a full URI, but for readability one is able to define a prefix abbreviation. Abbreviated URIs are written without angle brackets, while full URIs are always written with angle brackets.

```
@prefix fc:
 <http://floss.syr.edu/ontologies/2008/flosscomms-basic.owl#> .
```

Thus fc:Event stands for the full URI:

[4] http://rotterdam.ics.uci.edu/development/ontologies/padme.owl

```
<http://floss.syr.edu/ontologies/2008/flosscomms-basic.owl#Event>
```

Similarly the ex_vocab: prefix is just a made up prefix to denote example vocabulary (schema-like) statements, and ex_data: is a made up prefix to denote example data-like statements.

The example statements should be read as including these prefix definitions, as well as as well as the 'standard' prefixes and the final special prefix for typing the literal data (ie the content inside quote characters), which makes use of the data types defined by xml schema (which is a good example of the re-use that RDF makes possible).

```
@prefix ex_vocab: <http://example.com/vocabulary#> .
@prefix ex_data:  <http://example.com/dataset#> .
@prefix sf_proj:  <http://sourceforge.net/projects/> .

@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl:  <http://www.w3.org/2002/07/owl#> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
```

Each statement ends with a single period, but the subject (ie the first part of the triple) can be 're-used' by using a semi-colon after the object (but this is still two, or more, separate statements). Hopefully the indenting in the layout makes this clear.

By convention Classes are capitalized and CamelCased (ex:SomeClass), Object Properties are CamelCased but written with a small first letter (ex:someObjProperty), and Data Properties (those with literal subjects) are written with all lower letters, with words separated by underscores (ex:some_data_property).

B. TOOLS AND RESOURCES

There are many useful tools and resources available for learning more about ontologies. We developed our Vocabulary in Protege (using both 3.4 and 4.0), which is a Java-based GUI editor with built in reasoners and visualization capability. The Protege Pizza Tutorial is an excellent way to come to grips with the capabilities of OWL. The best thing we ever did was abandon attempts to read the RDF/XML representations and learn Turtle/N3 (which are plain text formats).

As far as tools go, we have found the open source Jena project to be the most capable API, drawing on the Java API to read, build and query our Vocabularies and Knowledge Bases. Using Jena, we wrote adapters to turn the FLOSSmole mailing list tables, the Notre Dame XML and the SVN log XML into RDF statements. For reasoning we have primarily used the Jena provided OWL_MEM_MICRO_RULE_INF reasoner, which performs well with large numbers of individuals (we also used the Pellet complete DL reasoner). For data storage we have primarily used the Jena SDB engine, which is a database backed store optimized for SPARQL queries. We have also explored the AllegroGraph commercial offering (which has a gratis version), which appears to have very interesting SNA and temporal reasoning capabilities built in.

C. REFERENCES

[1] I. Antoniades, I. Samoladas, S. K. Sowe, G. Robles, S. Koch, K. Fraczek, and A. Hadzisalihovic. D1.1 study of available tools. EU Framework deliverable, FLOSSmetrics, 2007.

[2] S. Christley and G. Madey. Global and temporal analysis of social positions at sourceforge.net. In *The Third International Conference on Open Source Systems (OSS 2007), IFIP WG 2.13*, Limerick, Ireland, June 2007.

[3] J. Howison, A. Wiggins, and K. Crowston. eResearch workflows for studying free and open source software development. In *Proceedings of the Fourth International Conference on Open Source Software (IFIP 2.13)*, 2008.