Google Summer of Code 2008 Semantic Web

Title: Moving DSpace into the age of the Semantic Web

Student: Peter Coetzee

Mentor: Mark Robert Diggory

Co-Mentors: James Rutherford, Scott Phillips, ???

Abstract

The current architecture of DSpace uses the DSpace Intermediate Metadata (DIM) format to store information about the items in its archive. It is presently crafted with a view to being concealed from outer view. This proposal is targeted at translating this private metadata format into an accepted metadata publishing standard, and integrating it into the web of data in accordance with Linked Data best practices (http://linkeddata.org/). By creating de-referencable URIs (as RDF/XML, N3, HTML etc.) for everything in the metadata store, it becomes trivial for a user to browse the data store based around related concepts ("This Author", "This Subject" and the like) simply by clicking on the metadata item in the Manakin display.

Furthermore, a semantic web client may use these de-referencable URIs as the starting point for information discovery, as well as formulating arbitrarily complex queries over its SPARQL endpoint. This work could directly feed into stated future plans DSpace has outlined for a federated query system over disparate data sources (LDAP, JDBC, DNS, XCat/OCLC/Barton, Google Scholar etc.)

This work would firmly begin to move DSpace into the age of the Semantic Web, providing compelling utility for users, and helping to boot-strap the web of data through the wealth of existing DSpace instances.

Broad Goals

There are four primary and heavily inter-related components to this proposal:

- Conversion of DIM to a publishable, linked RDF metadata format,
- Embedding of this RDF metadata as RDFa and/or Microformats in Manakin,
- Enabling clickable metadata display in Manakin to de-reference the linked data representations of this metadata, and
- A semantic-web queryable SPARQL endpoint of this metadata, to enable rich and expressive queries to be made across a DSpace repository.

Goal Breakdown

DIM Conversion

- Allow metadata to be attached to any DSpace Object (not just items)
- Maintain a triple store for ease of RDF serialisation + SPARQL queries (see below)
    - Use a single DB schema, offer views over it for triple store and legacy format?
    - Maintain a totally separate triple store in parallel with existing system?
- Add abstract superclass to handle metadata-related functions (factored out of Item)
- Alter XSLTCrosswalks to disseminate and ingest using RDF/XML rather than DIM
- Will need to mint URIs for all things
    - Use UUID for this?
    - Would be 'nice' to have readable URIs...better for SEO too, for those who care
- Use new URI as ObjectIdentifier / ExternalIdentifier, presumably

Embedded RDF

- Most likely Manakin-only, as this appears to be favoured over the older JSPUI for new development. Can always be backported, should the need arise
- Choice of preference between RDFa and microformats? Use both, if possible?
- GRDDL links to dereferencable RDF resource

Metadata Dereferencing

- New minted ExternalIdentifier URIs mentioned above should all be dereferencable using 303 redirects etc
- Build this as a servlet filter, so that it can be plugged into the XMLUI and JSPUI as-is?

SPARQL Endpoint

- Again, a servlet filter for integration into both UIs?

Open Issues

- Choice of persistent storage
  - Parallel stores
  - SQL Views over intermediate store
  - Pure RDF store
  - **Resolution**: Map existing SQL into RDF and persist in a triple store along side old relational store
- Choice of RDF framework
  - Performance comparison between Jena and Sesame needed?
  - Integrate Sesame into Jena? How does this compare?
  - Other stores / mappings (Virtuoso? D2RQ?)
  - **Resolution**: Initially backed by Jena, but framework abstracted away from as much of codebase as possible using a new MetadataManager service
- Where to inject the RDF
  - Change the adapters to convert RDF into METS / DIM etc?
  - Change the XSLT to transform RDF/XML?
- Decide how the DSpace data model fits into RDF, and vice versa

RDF Serialisation

- Must be XSLT'able
- Ideally fast; streaming straight from the database would be preferable
- TriX has some promise (http://www.hpl.hp.com/techreports/2004/HPL-2004-56.html)
- **Possible solution**: "R3X", a subset of RDF/XML with no grouping or nesting: http://www.wasab.dk/morten/blog/archives/2004/05/30/transforming-rdfxml-with-xslt
  - Initial experiments (using Jena) suggest that the built in RDF/XML-ABBREV serializer is slower than the built in RDF/XML serializer, as expected
  - An R3XWrter has been written to write out any Jena Model, based on the standard RDF/XML writer
  - StreamingStatementModel implementation written, to stream Triples in the base Graph using an iterator (only good for a single use, then the iterator is consumed)
  - Working from memory, this seems a small (but consistent) percentage slower than the RDF/XML writer. Profiling shows this to be a result of the extra calls to write out the extra rdf:Description for each statement. This is despite the standard Model construction time (from a List<Triple>) being factored in.
  - Streaming an SQL ResultSet from MySQL one row at a time appears to vary in speed; neither technique appears to have a statistically significant speed advantage
  - **Conclusion**: There appears to be no conclusive evidence upon which to base a recommendation of the R3X serialization format, as it is more verbose and thus a larger amount of data to work with. The overhead involved in grouping statements of the same subject appears to be minimal (thanks to Jena's internal architecture). The only compelling reason to opt for R3X would appear to be memory usage; should so many statements be used in a single model as to require streaming, this may be required to reduce memory usage. However, were this the case it should be feasible to use

    ```
    ORDER BY
    ```

    in the SQL and group statements in the DBMS. A variation on this streaming code could easily be made to have similar output to the RDF /XML Writer, but to stream results.
  - **Potential ToDos**: Investigate if there is there something to be gained by re-attempting this experiment with a different RDF API / DBMS. Group streamed results in the R3XWriter?
- **Possible solution**: Stream grouped results, using

  ```
  ORDER BY
  ```

  in SQL
  - Assume grouped ordering in R3XWriter in order to group results about resources in output, thus minimising the repetitive use of <rdf: description /> tags
  - This appears to be slower than the standard Jena RDF/XML writer for models over 10,000 triples. For smaller models, it affords a 2-3x speedup (this is counting the cost of retrieving results from the database) Most likely cause for slowdown is the SQL ordering in database.
  - Updated StreamingStatementModel to cache results from iterator in memory, and to deliver a union of these graphs
  - StreamingStatementModel can now be used for more than one iteration over its contents (but with added memory overhead, once the iterator has been consumed, as no further statements are streamed)
  - Once iterator is consumed, behaves exactly as if it were a normal in-memory Model - after partial iterations, it is a hybrid (half cached results, half streamed
  - This causes the R3XWriter to perform a small margin slower on writes, but allows the StreamingStatementModel to be plugged in to the standard Jena writers
  - New StreamingStatement model outperforms Jena's standard in-memory model within the RDF/XML writer by a factor of 1.5-2x for larger models (10,000 triples and up)
  - **Conclusion**: The new Model implementation suffers a slight penalty in object creation, but is probably an improvement for its flexibility. If models to be worked with are relatively small (< 10,000 triples), the new grouping R3XWriter offers an improvement. Otherwise, a StreamingStatementModel works best with the standard Jena Writer.

Integration With Existing DSpace Code Base

Reworked Metadata System

A new Metadata system is to be used (see org.dspace.metadata). This will represent an RDF graph as triples (or 'MetadataItems') of {URIResource, Predicate, Value}, where "Value" can be a URIResource or a LiteralValue. DSpaceObject is transformed into an Interface, with the existing abstract implementation factored into DSpaceObjectCore. The DSpaceObject interface extends URIResource. The graph will all be handled by the MetadataManager service, thus helping to keep the DSpace model as anemic as possible. This will permit a user to pass a DSpaceObject to the Manager, and retrieve a set of MetadataItems pertaining to the object. More specific searches will be possible by implementing the Selector interface. The onus of authorisation is on the MetadataManager, to ensure the current user has permission to act on the metadata in the requested fashion.

Content Integration

A StackableDAO is used to connect D2RQ to the triple store, such that each time a modification for the given DSpaceObject occurs, it is replicated in the triple store. The mapping is thus still controlled by the single D2RQ mapping file, without the performance penalties associated with exposing the D2RQ model publicly.

Data Publishing

This is handled in a set of Servlets, which will respond to SPARQL queries or conduct Linked Data content negotiation. They can simply be plugged into whichever UI is desired (or a separate webapp, if the instance is only to serve RDF).

- ContentNegotiationServlet pulls its configuration from the semweb extensions configuration file (currently config/mapping.n3), and will conduct content negotiation, and pass through to the SPARQL or Describe servlets.
- DescribeMetadataServlet handles de-referencing of URIs. It describes the "thing" requested using the MetadataManager to handle authorisation.
- SparqlServlet is slightly more complex. It supports the standard

  ```
  /sparql?query=EncodedQuery
  ```

  URL syntax, as well as an

  ```
  &format=FormatType
  ```

  option, to select SPARQL result serialisation. The single endpoint is used for all queries (SELECT, CONSTRUCT, ASK, DESCRIBE). ASK requires no special handling. For SELECT and CONSTRUCT the query form is analysed, and property functions are inserted for each triple match which results in output in order to authorise the Subject and Predicate of the triple. For example, the query

  ```
  SELECT ?s ?p
  WHERE
    {
      ?s ?p  ?o  ;
         ?p2 ?o2 .
    }
  ```

  will be expanded into

  ```
  SELECT ?s ?p
  WHERE
    {
      ?s ?p  ?o  .
      ?s ?p2 ?o2 .
      ?p <pf> rdf:Predicate .
      ?s <pf> rdf:Subject .
    }
  ```

  where

  ```
  &lt;pf&gt;
  ```

  is the URI of the property function. In this way, the authorisation is handled transparently. It is important to note here that the decision was taken to permit pattern matching without authorisation for READ on that DSO. Authorisation is only required for data which is output from the query.

DESCRIBE queries are handled using a custom DESCRIBE handler. This again uses the MetadataManager to get its data and handle authorisation for the resource requested. The query body is not changed in this case.

Embedded RDF will have to be implemented separately for each UI that it is desired for. An implementation exists for the JSPUI, which takes the form of a new tag, dspace:metadata. Example usages:

```
<%
    Community community = (Community) request.getAttribute("community");
%>
<!-- Display metadata about a single DSpaceObject -->
<dspace:metadata resource="<%= community %>" />


<!-- Display metadata about a set of DSpaceObjects -->
<dspace:metadata resource="<%= community.getCollections() %>" />
```

This tag will print out a block of RDFa metadata into the page, by default hidden from view. If you wish for this block of metadata to be displayed, add an attribute '

```
display="true"
```

' to the tag.

Remaining Tasks

- Some performance issues around authorisation of large blocks of metadata need to be cleaned up.
- Manakin is presently not working in trunk, so no RDFa has been integrated into it as yet. Similarly, DIM is still used for its XSLT as any changes could not be tested inside the UI.
- Code documentation
- De-referencable metadata links in UI (see above)

Orthogonal Issues

Cleaning up Qualified Dublin Core with DCTERMS/RDF

- Replace Metadata Registry file with dcterms.rdf
- Replace element names with qualifiers in new namespace where appropriate
- Move custom qualifiers to new namespace
- Example: dc:description.abstract --> dcterms:abstract

References and Other Research

References

http://www.tomjewett.com/dbdesign/dbdesign.php?page=intro.html Tom Jewett provides an excellent beginners overview of relational modeling in this tutorial, I've found that the patterns located here manifest themselves throughout the the DSpace database. --Mark Diggory 18:32, 26 May 2008 (EDT)

http://hcs.science.uva.nl/usr/Schreiber/docs/owl-uml/owl-uml.html Some alignment details between UML and OWL Lite.

Other Research

http://simile.mit.edu/reports/stores/ and http://esw.w3.org/topic/RdfStoreBenchmarking seem well executed, if potentially out of date - is there any value in repeating this work with DSpace specific instance data? – Peter Coetzee

D2RQ

http://www.w3.org/2007/03/RdfRDB/papers/d2rq-positionpaper/

- Example of D2RQ mapping rendered for the dspace.mit.edu database. Mapping_n3.mht

OpenLink Virtuoso

http://virtuoso.openlinksw.com/wiki/main/Main/VOSSQLRDF