**Descriptions of current RDF practice**

This document describes current (as of October 2013) practices of Hydra adopters who use RDF (beyond the context of RELS-EXT/-INT), including vocabularies, ontologies, infrastructure, and tools used. In addition, it surfaces challenges and gaps they have encountered.

**Sufia**

Sufia uses RDF for descriptive metadata and for controlled vocabularies. The terminology for descriptive metadata within Sufia is largely based on the Dublin Core Terms vocabulary, drawing other elements from FOAF (Friend of a Friend), and RDFS (RDF Schema). Descriptive metadata is by default stored in Fedora as a managed datastream and may be serialized in the plain-text NTriples format or in RDF/XML. (Other serializations such as Turtle may soon be added.) Sufia provides tools ("rake" tasks) for harvesting and caching RDF-based controlled vocabularies, such as LCSH (Library of Congress Subject Headings), Lexvo (languages and language families), and Library of Congress Genres. Once harvested, these vocabularies may be incorporated into descriptive metadata as values and may be hooked to web-based form fields to provide authority control or "authority suggestion."

**Gaps**

Implementers should have an option to store either or both the URIs and terms from controlled vocabularies in their metadata, and if URIs are chosen, there should be tools for resolving those URIs into terms for display purposes.

- The tools for harvesting, indexing, and caching vocabularies are rudimentary and inefficient and do not offer implementers much flexibility.

**Curate**

Curate extends Sufia to include... look here: [https://github.com/ndlib/curate/tree/master/app/repository_datastreams](https://github.com/ndlib/curate/tree/master/app/repository_datastreams)

**Vocabs**

- Primarily DC expressed NTriples
- FOAF - person/profile

**Tools**

- Curate - Rails Engine
- CurateND - Install of Curate for Notre Dame
- ActiveFedora Registered Attributes gem
- Questioning Authority gem
- Sufia Models gem (extraction from Sufia)

**Infrastructure**

- **Gaps**
  - As Sufia

**Oregon Digital**
100 collections across corpus

**Vocabs**
Dublin Core primarily
Using DarwinCore, MARC relators, etc. No limit is our hope.

**Tools**
Ingest forms

**Infrastructure**
Triplestore for vocab/namespace

**Gaps**
Similar to Sufia
Ontologies that don’t exist

**UCSD DAMS**

**Vocabs**

**Tools**
- ActiveFedora supports :class_name option for predicate mapping.

**Infrastructure**
Postgres “triple store”.

**Gaps**

References:
- GH Repo: https://github.com/ucsdlib/dams
- About our Data Model: https://libraries.ucsd.edu/blogs/dams/about-our-data-model/
- DAMS.owl https://github.com/ucsdlib/dams/blob/master/ontology/dams.owl

**RDF Samples**
- Sufia: bit.ly/sufia-rdf
- Oregons: bit.ly/oregon-rdf
- UCSD: https://github.com/ucsdlib/dams/wiki/REST-API-Response-Samples
- A graph https://dl.dropboxusercontent.com/u/6923768/Work/DAMS/DAMS%20object%20rdf%20graph.png
Patterns (Monday 2pm slot)

Sketch variety of current patterns and ideal pattern (if possible) for using RDF for metadata management in Hydra-based repositories.

Current Patterns

UCSD

Challenges

- modeling level (I missed this one. feel free to add this)
- collections and how they were dealt with in Solr
- implementation level
  - things that mean the same thing have a different predicate
  - inconsistent usage over time
- Complex objects exposed the problems even more

Oregon Digital has the same principles as UCSD but wants to use “Open World” and gain properties of an external term without having to integrate into data model.

“Ideal” Patterns

Maybe we’re not shooting for an ideal pattern here but rather some patterns that have broader appeal. For instance:

- At the model level, if you have “simple” metadata, use the Sufia or Curate recipe. If you have “complex” metadata, use the UCSD or Oregon Digital recipe.
- At the model level, use the module Notre Dame developed (active-fedora-attributes ?) to allow easier generation of views/forms
- At the view level, hook into your lower-level attributes in such-and-such way (see Notre Dame’s work on Curate and active-fedora-attributes
- … this is just an example of how we might tackle this: as a set of recipes that are already in use in the Hydrasphere

Input & Store Patterns

1. simple Sufia/Curate web form input
2. complex assembly plan / legacy collection ingest
3. lookup/store reference to external authority
4. define the pattern for the RDF data stream.

How do you know what code to use? How do you know what the data stream looks like?

Need docs, reference implementations, etc.

Simple web form input (Penn State, Notre Dame, WGBH)

Proposal - https://gist.github.com/jeremyf/7086692

User requests the web page for creating a new item (they’ve already told us what they’re going to create--I’m a book!, an ETD! etc.). Item’s ruby model is inspected, and reveals its fields that should be displayed based on that, as well as some of the validation that need to happen.

Render form.
User does entry / edits

Apply client-side JS validation for simple things (valid email format, eg.)

Client-side JS for controlled vocabulary look up / validation

User hits submit.

A service object handles the submission. This entails...

- I’m going to validate
- I’m going to persist
  - to the descMetadata datastream of a Fedora object, a managed datastream living on a filesystem (per Fedora/Akubra config)
  - to file datastreams (payload)
  - indexing happens
- I’m going to transform the payload to create derivatives
  - and persist those
- I’m going to offload work

It then renders a display view.
- This can then be inspected to figure out what to show.
- You can optionally start to get fancy here.

Prerequisites

- for each item type ("book"), need to define:
  - a terminology in the definition of your descMetadata (e.g.) datastream, that defines your metadata element set, what those elements map to, and what elements get indexed how
    - how do I define a flat terminology?
      - talk to your stakeholders about their metadata needs
      - enumerate a set of elements that are needed by stakeholders
      - map those elements to RDF predicates within your terminology
      - decide which elements get indexed how
      - link to example terminologies
      - tutorial: https://github.com/projecthydra/active_fedora/wiki/Lesson%3A-Define-a-Complex-Network-of-Related-RDF-Types
    - how do I define a nested terminology?
      - talk to your stakeholders about their metadata needs
      - enumerate a set of elements that are needed by stakeholders
      - map those elements to RDF predicates, classes, and so forth within your terminology
      - decide which elements get indexed how
      - link to example terminologies
      - tutorial: https://github.com/projecthydra/active_fedora/wiki/Lesson%3A-Define-a-Complex-Network-of-Related-RDF-Types
  - a set of delegations (maybe?) in your ActiveFedora model — these say, “if you call the title method of an instance of this class, it knows to reach into this datastream at this location”
  - a list of Active Fedora (AF) Registered Attributes which enables more programmatic form generation at the view layer

The above describes Jeremy’s pattern: Have the web form generation lean on the object via inspecting its model. An alternative is to have the web form generation lean on the vocab for the predicate (which lives at some externalizable URI) <- Tom J's pattern

Note: need to keep in mind that we will live in a bifurcated world of Hydra heads--some RDF-based, some XML-based (some F3, some F4).

Complex assembly plan/legacy collection ingest (UCSD, Oregon Digital)

- get an XLS from a data provider, one row per thing
  - typically collection based
- and/or get XML from a dump (from ContentDM, MARC XML, EAD extract, TEI, e.g.)
- have a LOT of conversation with metadata analysts and data provider (really a tremendous amount of talking here)
- produce an “assembly plan” from MD analysts
  - at Oregon: create a YAML mapping file that describes the mapping b/t the source and the RDF. Field by field
    - Mapping provides machine actionable ingest
    - Either CURIE or method for mapping
    - Returns BagIt bags for ingest into Hydra
  - at UCSD: METS profile that does the mapping
    - Provides guidance but not necessarily completely machine actionable
    - process currently in flux pending migration of data into new dams
- Can ArchivesSpace provide direct RDF output repository
- Analysts transform spreadsheet data into METS-like form
- METS-like form transformed into triples
  - DAMS manager has mapping language
- UCSD go-forward plan - get Metadacticians to talk about metadata in ‘rational’ ways
  - Offload preparatory work to platform (like ArchivesSpace) that could directly export into repository

To do:
- identify where is the code?
- what is the model?
  - how do you arrive at one?

Gap:
- XLS -> Hydra metadata converter (mappings and tools)
- agreement on a common data model
- vocab & ontologies for descMD that work in practice
- crosswalks to other
- transforms to popular serializations, exchange formats (MODS, DC, etc.)

- mappings
- look at Karma the tool. Visual mapping of a spreadsheet header to an ontology / element in a vocab

Lookup/store reference to external authority

https://github.com/projecthydra/questioning_authority

Mountable Rails engine that can be embedded in a Hydra Head

Three primary functions:

- **Lookup**
  - Questioning Authority gem provides a uniform endpoint for apps to query.
  - Need an adapter for each vocab of interest.
  - Must identify adapters in advance.
  - Gives option to bring down a localized copy of the URI:Term map.

- **Store**
  - If you decide to store URI:Term map locally, then you have a localized way to store key:value pair. Store only the ones looked up, or the whole vocab.
  - Persistence options:
    - Store just the individual key:value pair in Questioning Authority for display label retrieval.
    - Keep the whole vocab locally, cached in Questioning Authority. Do look ups from the engine for performance reasons.
    - Persist both URI and label to underlying data store (repository) for preservation reasons.

- cross-reference URI to get label
  - Expose a method to get a label out from the Questioning Authority gem to feed the Hydra app. Reverse look up.

Persistence patterns:

- store just the label (Sufia currently)
• store the URI and the label in DescMD
• store the URI only in DescMD, do a look up on retrieval to a vocab to get a fresh label

Potential Tooling:
• see (4) Gaps in RDF Tooling for more information  https://docs.google.com/document/d/1UHo5MxgHZJ5kuKs3m-IBJQd6PXLOhUEPwpQYeIm37D8/edit?usp=sharing
  • Apache Stanbol has an “entity hub” component
  • Questioning Authority is the cache and does look ups
  • ingest the URI as its own (authority) object into your core repository. Look it up.

Questioning Authority could use a “customer” to guide the rest of the implementation. Questioning Authority was built at Sept 2013 HydraPartners meeting at State College.


Oregons volunteer to provide feature input;

1. wants all three Persistence Patterns supported. (label, URI, label & URI)