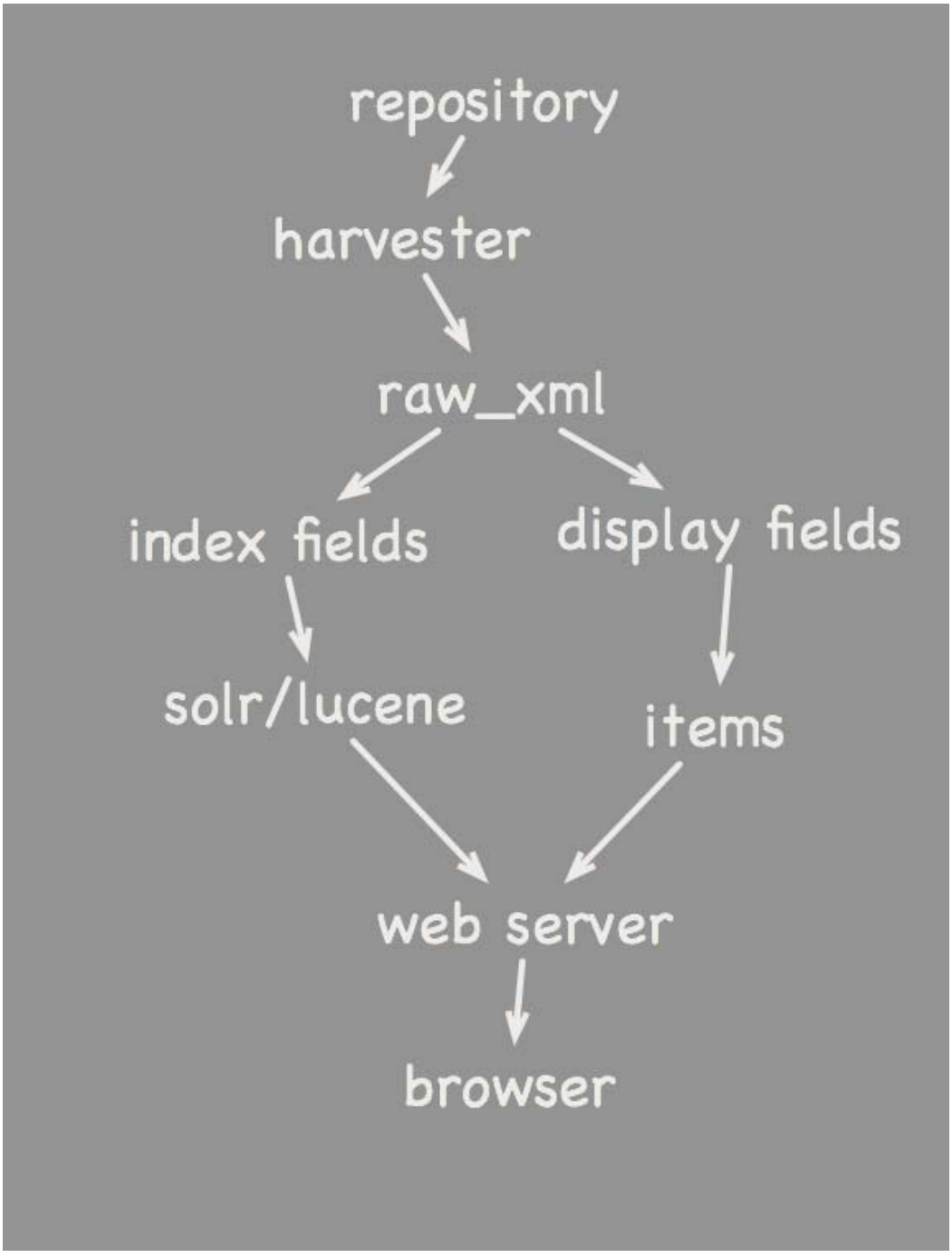
Digital Library Federation: Aquifer Project

Portal Architecture Overview

Chick Markley, Systems Architect

February 1, 2008

draft



Architecture

This document is an attempt to make plain the design and construction of the DLF Aquifer Portal. The project development theme is agility. The project is organic and at times names and constructions are bound by history and the trail of trial and error. While sometimes we have taken the time to correct the resulting inconsistencies, often the demands of keeping the system running and continuing to advance take precedence over devoting the time to going back and cleaning things up. The source code for this project is available at http://sourceforge.net/projects/dlf-aquifer/.

The rationale for caching a near displayable form of the xml is simple. Records are expected to be viewed and rendered many times before they are updated. Simple tests of the retrieving randomly selected records for display point to at least a two-fold improvement in display time.
We rejected

Environment

The architecture of the Aquifer Portal is built upon Ruby on Rails version 1.2.6 (RoR). The RoR architecture follows the Model/View/Controller (MVC) design pattern and we follow that whenever possible. This development and production of this application lives lightly on the underlying operating systems and accesses it directly absolutely as little as possible. Alternative technical strategies such as Java, PHP, DLXS, Perl were considered but Ruby was settled on both for its purported promise of short development time and engineering team's feelings that the other platforms, while serviceable, were complex and less productive than modern programming should be. The team continues to be satisfied and impressed with Ruby's suitability for a project of this scope and scale.

Project Highlights

- Simple flat view of metadata, wisely mapped from complex MODS format
- SOLR/Lucene works great for digital collection metadata
- AssetAction support provides new avenues for contributor control over digital object usage
- Zotero support through unAPI server
- System exposes data sources and transforms to interested users (digital librarians)
- *Distinctive* headings allow better machine generated differentiation of collections
- Unseen only filter for registered users presents only new objects across users searches
- eXist parallel database allows full XPATH ad hoc queries of database
- User history features help users and statistics
- Bots and spiders are treated as registered users for statistics purposes
- Ruby on Rails an excellent choice for Websites devoted to digital objects
- Community contributions
    - CQL/SRU OSS ruby plugin produced based on Java OSS libraries
    - TEMPER/CDL date normalization OSS ruby plugin produced from Java OSS libraries
    - acts_as_xml OSS ruby plugin for simplifying library metadata manipulation and integration with Rails ActiveRecord

Primary Records

The primary function of the portal is to aggregate and make searchable library metadata encoded in the MODS XML format. This is accomplished via two tables. Incoming MODS data is stored as is in the table *raw_xmls*, which has a text column that receives the xml text. The MODS data is converted into a displayable form via a batch program named raw_xml_to_item that uses an XSL transform to create an hash of an array of strings. The keys to the hash are a simple list of field tags, the value associated with each key is an array of one or more strings, i.e. fields for that particular tag.
This structure is serialized via JSON into a text column, data in the *items* table.

The rationale for caching a near displayable form of the xml is simple. Records are expected to be viewed and rendered many times before they are updated. Simple tests of the retrieving randomly selected records for display point to at least a two-fold improvement in display time.
We rejected the idea of storing a full rendered version of the MODS record in HTML in order to allow greater flexibility in the configuration of display of fields. Administrators can edit the field displays without having to regenerate the rendered forms. Ordering of fields in displays is controlled by the *tags* table which contains displayable labels for the field tags and ordering information for the brief and full record displays. The XSL file represents an enormous body of work and distilled wisdom from our Metadata Working Group.
(http://wiki.dlib.indiana.edu/confluence/display/DLFAquiferMeta/Data+Processing)

Note: There are enhancement requests to allow users to configure their own renderings. The structure is in place to support this but this development is out of scope for the American Social History Online project.

Indexing

The records are indexed and searched via the apache SOLR/Lucene software. Indices are generated by the batch program ItemIndexer from the MODS data with another XSL transform. Fields are generated using the basic text, string, facet and date types available through SOLR. Many MODS elements are indexed both in a text and facet field. There is a single field 'key_date_d' generated for searching that is used for user sorting purposes. There is a complex specification of cascading element and attribute priorities. This ordered list is scanned with a date parser built on the CDL date normalization utility/TEMPER tools. The first successfully parsed date (there are many unparsable dates such as n.d.) is then used as a sort date, additional dates indices are also generated including range aware decade and year facets.

SOLR's faceting technology allows users to break down search results into useful subsets, facet filters can be iteratively applied and removed within a search. Combining Lucene with the user item history allows a user to filter out records that a user has seen recently. This feature can operate across the different searches a user tries, so if a particular search does not turn up anything useful a new query will not show the same records over again. There is more work to be done with this feature, but it is very interesting.

Indexing is presumed to be the primary vehicle for user content discovery.

Headings

The headings subsystem contains a subset of the fields added to SOLR indexes. Counts of the number of times a given heading occurs overall and by collection are kept here. The tables here allow for a browsing interface in either list form or the popular tag cloud visual metaphor. Either format can rank headings alphabetically, by frequency, rarity and by their **distinctiveness**, i.e. how headings in a particular collection differ from the aggregate collections. Distinctiveness, unlike the other formats ranks headings by the ratio of the headings occurrences within a collection to the occurrences of the heading in the database of a whole. The headings sorted this way are very useful in presenting the interesting and differentiating content of each collection.

Heading access is presumed to be the primary vehicle of search engine content discovery.

User

Site users may optionally register themselves. Registration allows users to configure a few features of the systems; number of records returned on a page, save search history, etc. The user login uses OpenID, simplifying the login code. Tables track users' searches and records retrieved and viewed. A known list of bots are identified via the USER_AGENT request element and treated as registered users. The data accumulated in the item_histories table for these users can provide metrics for search engine coverage of the system.

Collections

The collections table contains MODS records, in both their original xml text and in a JSON serialization. The serialization used is different than the items table. The approach here preserves the full MODS hierarchy. It represents an alternate technical approach to rendering MODS records that might later be incorporated in the items table.

Machine Access

The portal supports access via the unAPI and SRU and AssetAction protocols. unAPI is used to support the Zotero citation management tool. SRU allows Sakaibrary and other external clients to search and retrieve records, the CQL parser is a ruby translation of the standard Java CQL implementation, and includes a first draft of an attempt to provide a generic query mapping from CQL to SOLR. AssetActions provide a mechanism for contributors to define and control additional functions for a digital resource. Currently AssetAction support is provided for the Collectus image management tool.

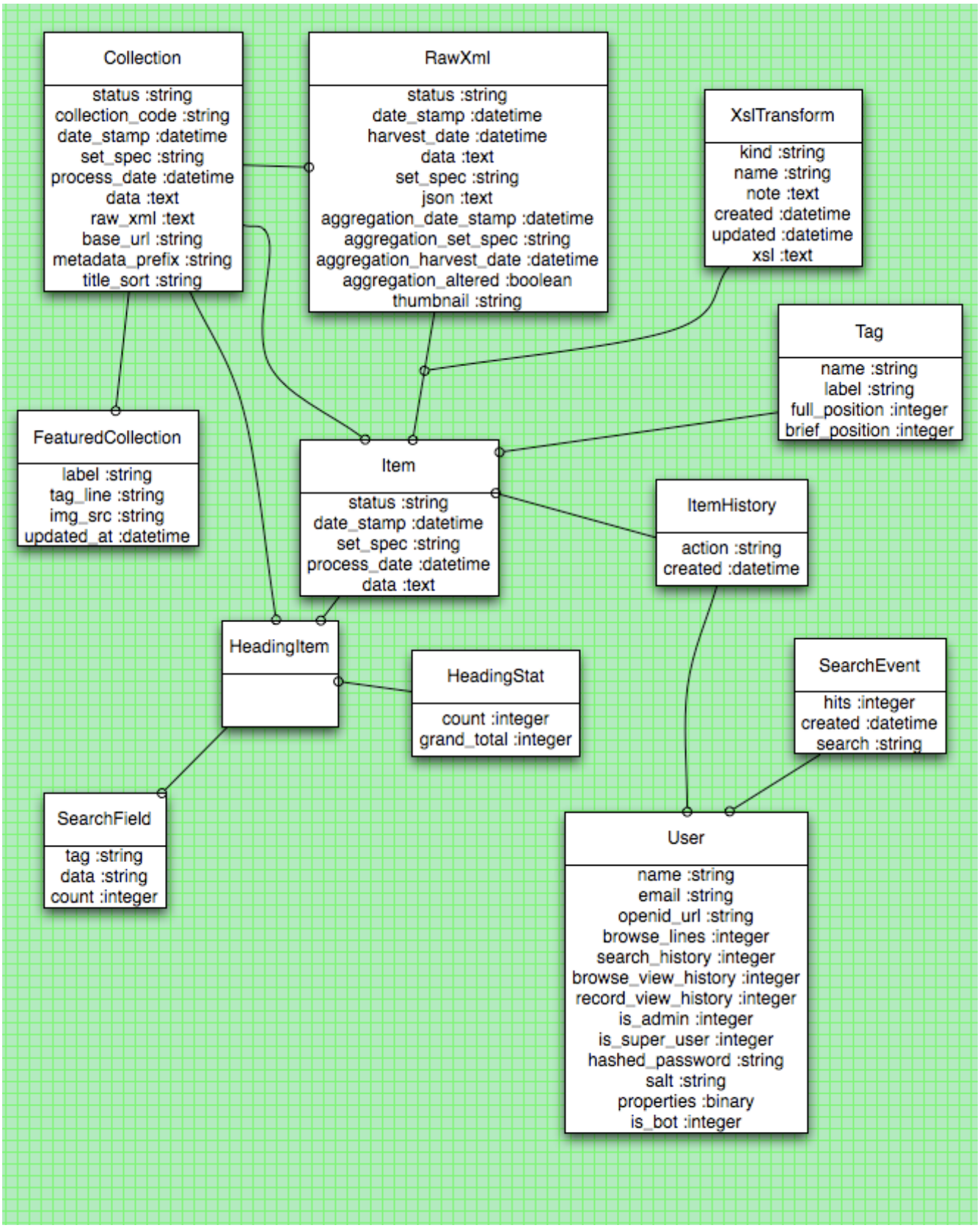
Data and Process Exposure

Whenever possible the portal attempts to pull back the curtains on its inner workings. XML source to original MODS record is available wherever records are presented. Additionally the XSL transforms themselves are presented including a simple interface that allows interested users to apply them to individual records and look at the resulting output. Non-binding edits can be made to the XSL transforms to check proposed changes.

The portal also maintains a parallel database based on the eXist OSS XML database platform. All contributed MODS records are added to this database which allows for full add hoc XPATH queries. This tool proved invaluable during the MetaData Working Group's data processing standards development. Programmers and Librarians could examine the database from a strict XML viewpoint, identifying standard usage patterns and problematic coding practices. It was not felt that using the

eXist system was a suitable replacement for the MySQL database, but that could change in the future. The barriers are about performance concerns and limited development time and not about any technical programming limitations.

Schema


Collection
status :string
collection_code :string
date_stamp :datetime
set_spec :string
process_date :datetime
data :text
raw_xml :text
base_url :string
metadata_prefix :string
title_sort :string
RawXml
status :string
date_stamp :datetime
harvest_date :datetime
data :text
set_spec :string
json :text
aggregation_date_stamp :datetime
aggregation_set_spec :string
aggregation_harvest_date :datetime
aggregation_altered :boolean
thumbnail :string
XslTransform
kind :string
name :string
note :text
created :datetime
updated :datetime
xsl :text
Tag
name :string
label :string
full_position :integer
brief_position :integer
FeaturedCollection
label :string
tag_line :string
img_src :string
updated_at :datetime
Item
status :string
date_stamp :datetime
set_spec :string
process_date :datetime
data :text
ItemHistory
action :string
created :datetime
HeadingItem
HeadingStat
count :integer
grand_total :integer
SearchEvent
hits :integer
created :datetime
search :string
SearchField
tag :string
data :string
count :integer
User
name :string
email :string
openid_url :string
browse_lines :integer
search_history :integer
browse_view_history :integer
record_view_history :integer
is_admin :integer
is_super_user :integer
hashed_password :string
salt :string
properties :binary
is_bot :integer

Controller/Action Summary

- collection
  - index
  - list
  - show
  - heading
  - group
  - index
  - list
  - sets
- home
  - about
  - conditions_of_use
  - contact_us
  - faq
- help
  - index
  - tools
- my_profile
  - edit
  - index
  - records_selected
  - record_viewed
  - search_history
- search
  - advanced
  - full_record
  - heading_list
  - images
  - index
  - item
  - mods_record
  - random_record
  - timeline
  - transform_record
- sru
  - diagnostics
  - index
  - explain
  - search_retrieve
- tag_admin
  - index
  - list
- unapi
  - index
- user_admin
  - edit
  - index
  - list
  - show
  - update

A controller plus an action forms the base section of the url used for the site. For example to display the about page the url is /home/about, to display a particular item found in a search the url is /search/item additional sub-elements and arguments may further target the url to a particular page or resource.