Architectural overview and customizations

VitroLib extends Vitro, the open source ontology and instance editor that provides the ontology-agnostic semantic application underpinning VIVO, the researcher profiling system. VitroLib generates content display and content editing interfaces based on BIBFRAME, Bibliotek-o which extends BIBFRAME, and related ontologies.

What is Vitro?

Vitro is a general-purpose web-based ontology and instance editor with customizable public browsing. Vitro was originally developed at Cornell University, and is used as the core of the popular research and scholarship portal, VIVO. Vitro is an integrated ontology editor and semantic web application implemented as a Java web application that runs in a Tomcat servlet container. With Vitro, you can: create or load ontologies in the Web Ontology Language (OWL) format; edit instances and relationships; build a public web site to display your data; and search your data with Apache Solr.

How does VitroLib extend or customize Vitro?

Similar to how VIVO adds VIVO ontology-specific customizations for information display and entry, VitroLib adds customizations for the display and addition of BIBFRAME and related ontology information. Being ontology-agnostic, Vitro can support the display and editing of BIBFRAME data, but we opted to explore more usable design that could support catalogers in their workflow than that provided by Vitro which is better suited for an audience that has greater familiarity with ontology editing tools. Furthermore, we refactored code responsible for integrating vocabulary lookups from VIVO into the Vitro layer and wrote implementations for looking up Questioning Authority sources. We also explored how to enable more client-side configuration of property-specific customizations. The work we did to translate and integrate SHACL is defined (here).

Examples of content-specific or workflow-specific customizations

We used Vitro's built in configurations or display and editing customization support to modify the default Vitro behavior to better suit cataloging workflows or provide content-specific behavior. We included a link on the home page to provide quick access to catalogers for cataloging a work/instance/item. This link used a custom editing form that generated three related entities: work, instance, and item, and then redirected the user to the resulting work entity profile page.

We used "faux properties" which provide configurable display of labels, display, and links to custom editing forms to map from SHACL to provide properties and property groupings that could help catalogers with viewing and navigating the system. We included work, instance, item specific templates. These templates defined the broad category, i.e. work, instance, item, and not just the specific ontology class applicable to the entity. We did not want both a faux property and the underlying ontology property to show up on a page, as it would be confusing to users to see, for instance, both the faux property "agent activity and role" as well as the ontology property "hasActivity" when the first is what we want the user to use. We updated content-specific templates to hide the ontology property if a faux property masking that property is present. We also include custom editing forms for several different properties to help provide a more usable interface that didn't require catalogers to know all the details of the ontology.

Integrating Questioning Authority and other lookups

Lynette Rayle and Dave Eichmann have worked on creating an infrastructure and code for enabling a configurable approach to integrating "Questioning Authority" or QA lookups. Lynette has worked on creating configurable linked data lookups using the QA Ruby on Rails gem. We refactored the VIVO lookup code to provide JAVA interfaces in the Vitro layer that could then be implemented in VitroLib to enable lookups to be available to the VitroLib interface. We added QA implementations for Library of Congress Name Authority File, Library of Congress Subject Headings, and the Library of Congress Genre Form Terms QA lookups that were made available on a server setup for this grant. We also incorporated some contextual information made possible by Lynette's implementation to enable more information to be displayed to the user.

Below are some screenshots showing the integration of QA lookups into the VitroLib interface. Figures 1 and 2 below the integration of lookups from the Library of Congress Name Authority File where a user can select from autocomplete results. We modified the lookups front-end to enable the "Verify this match" link to open up the Real World Object found in the result. From cataloger feedback, we know the desired behavior is to open up the authority file instead but this modification enabled some level of review of the selected result. Also, Figure 1 shows ISNI as a potential lookup source which is not implemented as a QA authority but which was implemented as a VitroLib search option.

Figure 1: LCNAF Lookup

Figure 2: Verify this match link next to selected LCNAF lookup

Figures 3 and 4 below show the integration of the Library of Congress Genre Forms lookup using additional context in the search results. The user types in "animation" when adding a Genre Form property and is able to see multiple search results which display alternate labels as well as links to broader and narrower terms where applicable. Clicking on the search result name will open up the corresponding Library of Congress Genre Form page . The design of these tables was influenced by the mockups that Lynette Rayle had designed for exploring contextual information retrieval in QA lookups.

Figure 3: Genre Form lookup with query

Figure 4: LCGFT lookup results

Figure 5 below provides an overview of the lookup process in the VitroLib implementation. The user's query is carried from the interface to code within VitroLib that implements the Vitro search interface and that parses the query and makes the request to the Questioning Authority URL that performs the search and returns results in JSON. The VitroLib code then parses the results and returns them in a format understood by the VitroLib interface for display.

blocked URL

Figure 5: Overview of QA integration with VitroLib

Client-side configurations of custom forms

We worked on creating client-side configurations for defining custom editing form behavior. Without these customizations, in Vitro or VIVO, creating custom forms for adding and editing information require the creation of the following components: (a) a JAVA class responsible for specifying the RDF to be added to the system on addition or modification as well as for defining the fields that can have values submitted from the form, (b) a Freemarker template providing the front-end display of the form fields, and (c) any related JavaScript for dynamic display or validation.

We wanted to streamline this process and enable easier configuration and quicker iterations. This work took some time and was not meant to replace the JAVA-based custom form method but provide the ability to automate custom forms on the basis of configuration for simple cases and enable client-saved configuration in combination with templates and javascript for some other cases. The new configuration is client-side including (a) a JSON-LD configuration replacing the JAVA code that specified the expected fields on the form and the generated RDF from those fields on form submission and (b) a JSON object configuring display characteristics such as ordering, labeling, dropdown contents for hard-coded select field content, and whether a field is using autocomplete searching internal VitroLib data.

The fully automated solution requires just these configurations and can generate a display which outputs fields in the specified order and maps to the specified RDF. For situations that require interactions between different fields or a different display, the client-side configurations can be used with a specified Freemarker template which in turn can include any custom JavaScript that is required for that page. The JSON-LD configuration and any Freemarker template are defined as related to a property or faux property in RDF files that are loaded as part of VitroLib. One of these properties exists in out of the box Vitro while the property associating the template with the property was created specifically for this work.

The client-side configuration enables for quicker VitroLib testing and development. That said, the configuration still requires understanding multiple moving parts: the RDF and field configuration, the display configuration (if required), any Freemarker templates that might be required, and JavaScript that may be associated with those templates.

Linked Data Notifications (or LDN) Experiments

As part of work with Harvard, we experimented how we may integrated LDN into VitroLib. An RDF file specified a triple which used the "inbox" predicate to specify the LDN inbox URL. We were able to demonstrate a servlet that read these triples in and then was able to request inbox contents from the LDN inbox URL. This servlet thus demonstrated consumption of the LDN inbox.

We also added code that would write to the LDN inbox whenever data was added or edited through the interface. Vitro comes with reasoning code that is triggered when a user adds or edits information through the interface and this code can see which triples have been added or removed. The code we added as part of this experiment took those triples that were added and then submitted them to the LDN inbox that we defined in our RDF that was part of the system. We thus were able to both read form and write to an LDN inbox in VitroLib.