Lessons learned: VitroLib and SHACL

What is SHACL and why did you use it?

Ontologies define what types of entities should be modeled and how these entities relate to each other through properties. If an ontology specifies that a property should have a specific class as a domain (i.e. what type of entity can be the subject of a statement using that property) or as a range (i.e. what type of entity can be the object of a statement using that property), then data following this ontology should follow these rules and not allow other classes to be used as domains or ranges for that property. An ontology could thus choose to not specify a domain or range, leaving possible options for domain or range classes open to particular implementations. Ontologies can define unions for domains or ranges but not all will pick this option and not all software applications will be able to interpret this option. Ontologies define rules that should be followed by data in general and often do not or cannot define context-dependent information as we describe below.

When generating an interface from an ontology, the lack of a domain assignment can be translated in two ways: either all classes can serve as a domain for a domain-less property or none of the classes can serve as a domain. The desired interface behavior is often that certain domains may be applicable while others may not be. Similarly, for a particular property, the desired interface behavior may be to present certain labels or options for display or editing in the context of a particular domain or range while displaying those labels or options differently for different domains and/or ranges. In a hypothetical example, when using the property “related” to connect a book with an author, the interface should show “authors” as the property label with the domain “book” and the range “person”. When displaying the same property with the domain “painting” and the range “person”, the interface should show “artist” as the label for the property “related” instead. In neither case, do we want the property label provided by the ontology, and in both cases, the ontology does not provide information about which domains can be applicable and which labels should be used in which context.

To implement expected ontology behavior that is dependent on context, we relied on metadata application profiles constructed in LD4P for music as part of Cornell HipHop collection and for Rare Materials and ArtFrame as part of the ARM group. We relied on SHACL specifications of these application profiles to help generate portions of the VitroLib interface.

The Shapes Constraint Language or SHACL is a “a language for validating RDF graphs against a set of conditions” (https://www.w3.org/TR/shacl/) . SHACL can also be used to represent what properties should be present on a form for RDF content submission. For VitroLib, we used SHACL for this second purpose for defining forms for adding specific content types. We used SHACL to define which ontology properties were to be grouped together, what the properties needed to be labeled, and what those properties needed to link to or what values those properties should have. The great benefit of having an RDF specification is that the specification can be queried and mapped to the software-specific configuration required for a particular software application.

Why SHACL and not ShEX?

The main reasons we picked SHACL were those of convenience in this particular project:

(a) SHACL provided definitions of property groups and (b) TopBraid had a tool we could use to experiment with SHACL creation. From the perspective of VitroLib, the choice between SHACL and ShEx was not based on deep philosophical or implementation differences but rather whatever option was available to the project to begin defining application profiles. A similar process for mapping SHACL components to VitroLib configuration options could be done for ShEx or some other application profile standard. The contribution of this work is to provide a proof of concept in VitroLib showing the translation of metadata application profiles to an interface for adding and editing RDF data.

Summary of observations

Display and editing configuration is distributed across different files and models in Vitro. Defining the appropriate configurations in VitroLib is thus already a matter of juggling various configuration components. The code we wrote to translate SHACL and generate the appropriate VitroLib configurations was able to automate some of the generation of VitroLib configuration from SHACL while other components were not automated as the process seemed more involved, required additional analysis, or was not simple to automate (such as the creation of custom editing form for a property). In spite of the complexity surrounding configuration generation and processing, we were still able to use the mapping code to generate some of the interface for three different use cases and related application profiles. In addition, merely having any kind of standardized specification was very helpful in determining how the interface was meant to behave, even if not all portions of the SHACL could be expressed or generated automatically.

We should also note that certain SHACL possibilities need further analysis with respect to VitroLib configuration. For instance, if a property can have multiple classes as a range, does this result in the interface treating that property as multiple different properties (one per range class) or by collapsing all those possibilities into one property? The VitroLib “faux property” configuration (explained below) was dependent on unique domain and range combinations for a property and thus could not collapse multiple range classes into a single property configuration.

How does VitroLib use SHACL?

We experimented with three different use cases: (a) the Cornell HipHop collection which focused on audio content, (b) the ARM workshop which focused on RareMaterials content, and (c) the ArtFrame approach which supplied or overwrote some Art-specific profiles in addition to those provided by ARM. In each of these cases, we used code that operated independently of VitroLib to query the SHACL profiles and generate corresponding VitroLib application configurations.

The editing and display configuration components we mapped to either partially or completely were property groups, “faux property” and custom editing forms. Property groups in VitroLib can show up in tabs in the interface and display the properties within that group to allow users to see what information has been added for those properties as well as to add more or edit existing information. SHACL profiles allow the definition of property groups and these map cleanly to VitroLib property groups. As an example, the following is an excerpt from the HipHop collection SHACL which defines how properties will be grouped together as well as the label and display order of these property groups.

blocked URL

The SHACL to VitroLib translation code queries this information and then maps it to property group definitions in VitroLib. The resulting RDF is then included in VitroLib in the appropriate folder. (As an implementation note, property groups in Vitro are either read in from the files the first time the application is deployed or if the corresponding graph is empty when the application runs.) Once the files have been read in, the resulting screen should display property groups as they are shown in the screenshot below.

. blocked URL

Faux properties refer to a Vitro display configuration method that enables the definition of context-specific behavior for a particular property. A “faux property” allows a property to be displayed for a given domain and a given range with a particular label, display of information, and connection to any custom editing form (usually referred to as “custom form”). The table below shows a mapping between some of the concepts that can be represented in SHACL and the corresponding part of a faux property configuration.

blocked URL

In Figure 2 above, “Agent activity or role” is a faux property that displays this label for the ontology property “has activity”. The definitions of which faux properties should show up on which pages are translated from the SHACL which can define the expected domain and range for a property. The labels and ordering of the property and their inclusion in a property group are all translated from the SHACL to the corresponding VitroLib configuration.

The faux property in Figure 2 also uses a custom display (using ‘list views’ in VitroLib) which enables the display of what role is related to the activity. The faux property also links to custom editing behavior using a custom form. These relationships were not generated from the SHACL but we integrated the generation of this RDF into the code for translating SHACL.

Although the custom form is not generated entirely from the SHACL, the dropdowns that are present on the form are generated using the SHACL specification. The figure below shows the rdf:type property links to a list of possible activity classes using the SHACL predicate “in”. Our code was able to query these lists and reference the ontology files to generate the JSON that could be used to define the dropdowns in the interface as seen in the next figure.

blocked URL

blocked URL