Semantic Web View Language Experiment Log

General Problem:
We need a language for defining custom views over reference ontologies expressed in semantic web languages. Although most such ontologies are represented in some form of OWL, we assume only that they are in RDF. For a defined view, we also require mechanisms for materializing a view based on its definition. Additionally, we would like to be able to resolve queries against an ontology view without first materializing it, but rather by composing incoming queries with the view definition in order to form queries over the original reference ontologies. Ideally the query and view languages should be the same, enabling chains of views defined as queries over other views.

Rule Languages:
Another possible basis for a VDL lies in a rule language (i.e. a Rete or LP rule language like Prolog, Jess, SWRL, Datalog, etc.). There is room in the evolving standards of the semantic web for both a query and a rule based language. Such rule languages could be used to express views and to pose "queries" over views. When considering rule languages, I started with a couple of initial questions:

1. Do rule engines produce results that are in the same data model as the source ontology (closure property)? This is necessary to enable a view to be queried with the same language as the original ontology.
2. Can rule engines be configured to NOT alter the source ontology? We don't want to alter the reference ontology, but instead to produce a new ontology based on the results of rules fired against the reference ontology.

In addition to the above rules, I had general concerns regarding the expressiveness of rule-based languages. Do such languages support all of the VDL operations that we require (see ModifyingOperations wiki page).
As a first pass at a survey of rule languages for semantic web ontologies, I looked to SWRL (Semantic Web Rule Language), a W3C candidate submittal. Implementations of SWRL are few, at present. My initial investigation included rule engines on the following 3 platforms Protégé's SWRL tab, Sesame and an experimental SWRL engine from Jing Mei (Christine Golbreich), and the rule engine built into Jena (not SWRL).
(Note: I also intend to look at Pellet at some future point, but I must check that pellet will work with non-OWL RDF docs.)

1. **Protégé SWRL tab:**
   The Protégé SWRL tab is a Protégé OWL extension which is now included with the Protégé OWL distribution. It provides a graphical interface for creating rules. The SWRL tab is visible for any ontology which includes the SWRL ontology. This is an important note. SWRL rules are created as instances of classes in the SWRL ontology. When using the SWRL tab, created rules are stored along with the ontology on which rule inference will be preformed. This is the first limitation that I uncovered; the rules are stored directly in the source ontology. This could likely be modified without too much difficulty.
   Protégé's SWRL tab does not include a native SWRL engine, but instead allows user's to export rules and relevant facts from the knowledge base as Jess rules and Jess facts respectively. The Jess engine is then used to execute the rules. Any new

facts, resulting from the inference, can then be pushed back into the ontology. One thing that I like here is that inferred facts were not automatically added back into the source ontology. For our view creation needs, we do not wish to alter the reference ontology.

While this approach worked very well for the public "family" ontology, it did not produce any inferred facts when used in conjunction with the FMA_RadLex ontology. While this is not confirmed, my suspicion here is that, since SWRL rules only work with "instances", the Protégé facts to Jess facts export might only export "simple instances", for which there are none. However, I would still expect results since the FMA_RadLex ontology contains classes that are also instances (OWL full). Also, I am unsure of how much of the ontology is exported to Jess facts, if it is everything, then this would be problematic for the entire FMA.

It should be noted that efficiency seemed reasonable, but test cases were small in size and I did not attempt to add any knowledge back into the reference ontology (which I have no need for within the context of view generation). All tests of this approach were done in main memory. I am not sure if the SWRL tab works with Protégé OWL persistent models (at present the SparQL panel does not).

Below are some screenshots illustrating Protégé SWRL tab. Figure 1 shows two rules (the 2nd and 3rd entries) that together define the hasAncestor(x,y) relationship as the transitive closure of hasParent(x,y). The first rule uses the built-in function "query" to query over the hasAncestor() relationship and order the results. Figure 2 show these 3 rules exported as Jess facts (some parts of these rules are truncated in screenshot). Figure 3 show a few of the resulting property assertions. These can be read as follows (considering the first result in Figure 3): M06 (identifier for an individual male) hasAncestor F03 (identifier for an individual female).

SWRL Rules

Enabled	Name	Expression
☑	Ancestor-query	→ hasAncestor(?x, ?y) → query:select(?x, hasAncestor, ?y) ∧ query:orderBy(?x)
☑	Def-hasAncestor-base	→ hasParent(?x, ?y) → hasAncestor(?x, ?y)
☑	Def-hasAncestor-iter	→ hasParent(?x, ?y) ∧ hasAncestor(?y, ?z) → hasAncestor(?x, ?z)

Figure 1: Two SWRL rules which together define the transitive relationship hasAncestor and a query over that relationship.

Jess Control | Rules | Classes | Properties | Individuals | Restrictions | Asserted Indivic

Jess Rules

```
(defrule Ancestor-query (hasAncestor ?x ?y)  => (invokeSWRLBuiltIn Ancestor-query query:select 0 ?x hasAncestor ?y) (invok
(defrule Def-hasAncestor-iter (hasParent ?x ?y) (hasAncestor ?y ?z)  => (assert (hasAncestor ?x ?z)) (assertOWLProperty ha
(defrule Def-hasAncestor-base (hasParent ?x ?y)  => (assert (hasAncestor ?x ?y)) (assertOWLProperty hasAncestor ?x ?y) )
```

Figure 2: SWRL rules exported to JESS rules

```
(assert (hasAncestor M06 F03))
(assert (hasAncestor M05 F04))
(assert (hasAncestor F02 F01))
(assert (hasAncestor F09 F04))
(assert (hasAncestor F06 F07))
(assert (hasAncestor M02 M01))
(assert (hasAncestor M10 F01))
(assert (hasAncestor M10 M04))
(assert (hasAncestor F09 M01))
```

Figure 3: Partial list of asserted properties.

2. **Sesame and SWRL prototype:**
   Like Jena, Sesame is also a popular RDF platform. At the present time there does not appear to be a SWRL implementation distributed with Sesame. There is a built in inference language, which I have not yet experimented with but hope to in the next round of investigation. What I did look into was Sesame in conjunction with a small prototype application written by Jing Mei while at the Free University Berlin. This prototype transforms input SWRL rules into SeRQL queries processable by Sesame. I experimented only with the in-memory database version of this prototype as the rdbms based version was not functional. I tried to modify the non-functional rdbms version, but the SeRQL queries that it issues no longer seem viable and I did not have any immediate alternative.
   This prototype was pretty fast, particularly considering that it was first converting to queries. However, as I mentioned, I was only able to test a version were the data store was in memory. However, I was able to execute SWRL queries against the FMA_RadLex ontology without the “instance” problems experienced with the Protégé-Jess bridge.
   Two things that I didn’t like about this approach were that all of the rules had to be first added to the reference ontology, and all of the inferred facts were automatically added to the reference ontology. Neither of these is desirable for my goals. However, these issues might be easily fixed at the code level as this prototype is written with only a very small amount of code.
3. **Jena rules:**
   The Jena platform provides its own, non-SWRL, rule language. It provides both forward chaining (using a Rete engine) and backwards chaining (using an LP engine). It also supports a limited hybrid rule engine with both forward and backward chaining. The expressiveness of Jena rules is similar to, but not identical to, SWRL. I have not yet completely compared the 2 in this regard.