CEID

MSc on DATA DRIVEN COMPUTING AND DECISION MAKING (DDCDM)

SWRL (Semantic Web Rule Language)

I. Hatzilygeroudis, Professor Emeritus

A general idea…

- H SWRL(Semantic Web Rule Language is a proposal for a rule-based knowledge representation language on the Semantic Web, combining the sub-languages of the OWL Web Ontology Language (OWL DL and Lite) with those of the Rule Markup Language (Unitary/Binary Datalog).

Rules

- A SWRL rule has the format:
  - $\mathbf{B_1, \ldots, B_n \rightarrow A_1, \ldots, A_m}$
  - Commas show conjunction in both sides
  - $\mathbf{A_1, \ldots, A_m, B_1, \ldots, B_n}$ can be of the form **C(x)**, **P(x,y)**, **sameAs(x,y)**, or **differentFrom(x,y)** where **C** is an OWL description, **P** an OWL property , and **x**, **y** are Datalog variables, OWL constants, or data values

SWRL Properties

- If the head of a rule has more than one atoms, the rule can be transformed into an equivalent set of rules with one atom as the head
- Expressions, like constraints, can appear in the head or body of a rule
- This feature adds significant expressive power to OWL, but at the cost of a high indecisiveness value.

SWRL Syntax

Syntax of an atomic expression is defined as follows:

Atom ← C(i) | D(v) | R(i, j) | U(i; v) | builtIn(p, v1,…, vn) | i = j | i ≠ j

- C : class
- R : object property
- i, j : variables of names of the object or individual names of variables
- v_1,… v_n : data and variables types
- p : Built-in name

- D : Data type
- U : Data type Property

Rule example

hasParent(?x1,?x2) ^ hasBrother(?x2,?x3) → hasUncle(?x1,?x3))

SWRL Syntax

has_father(?a,?b) ^ has_married(?b,?c) → female(?c) ^ has_mother(?a,?c)

It is worth noting that a rule with multiple atoms in its head is equivalent to multiple rules having the same body as the original rule and an atom as their head.

The above rule is equivalent to the following rules :

has_father(?a,?b) ^ has_married(?b,?c) → female(?c)

and

has_father(?a,?b) ^ has_married(?b,?c) → has_mother(?a,?c)

Note

- The combination of rules with ontologies provides great expressiveness, as it tries, and partly succeeds, in combining the advantages of classical logic and logic programming.
- However, some key disadvantages remain. It should be emphasized that the SWRL language is an extension of OWL and this results in working consistently with the Open-World Assumption, just like OWL.
- Thus, SWRL does not support a 'not' operator, unlike most rule environments that operate in closed-world environments.
- Finally, it should be noted that since the SWRL rules have the form of Horn-rules, they do not allow disjunction either in their body or in their head.

The verbal-logical description of rules

If an artist x has adopted a style y and has built and artifact z then artifact z is of style y.

The verbal description of the rule

The logical description of the rule

artist (x) ^ style(y) ^ artist Style (x,y) ^ creator(x,z) → artistfactStyle(z,y)

RDF Concrete Syntax

```
<swrl:Variable rdf:ID="x"/>
<swrl:Variable rdf:ID="y"/>
<swrl:Variable rdf:ID="z"/>
 <ruleml:Imp>
  <ruleml:body rdf:parseType="Collection">
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="Artist"/>
       <swrl:argument1 rdf:resource="#x" />
    </swrl:ClassAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="Style"/>
         <swrl:argument1 rdf:resource="#y" />
    </swrl:ClassAtom>
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate rdf:resource =
                                        "artistStyle"/>
       <swrl:argument1 rdf:resource="#x" />
       <swrl:argument2 rdf:resource="#y" />
    </swrl:IndividualPropertyAtom>
```

```
<swrl:IndividualPropertyAtom>
  <swrl:propertyPredicate
                        rdf:resource="creator"/>
     <swrl:argument1 rdf:resource="#x" />
     <swrl:argument2 rdf:resource="#z" />
</swrl:IndividualPropertyAtom>
</ruleml:body>
<ruleml:head rdf:parseType="Collection">
 <swrl:IndividualPropertyAtom>
   <swrl:propertyPredicate
                      rdf:resource="artifactStyle"/>
     <swrl:argument1 rdf:resource="#z" />
     <swrl:argument2 rdf:resource="#y" />
 </swrl:IndividualPropertyAtom>
</ruleml:head>
</ruleml:Imp>
```

Analytically (1)

```
<swrl:Variable rdf:ID="x"/>
<swrl:Variable rdf:ID="y"/>
<swrl:Variable rdf:ID="z"/>
```

Definitions of variables

Definition and matching body predicates Specifically:

artist (x)
style(y)
artist Style (x,y)
creator(x,z)

```
<ruleml:body rdf:parseType="Collection">
<swrl:ClassAtom>
  <swrl:classPredicate rdf:resource="Artist"/>
  <swrl:argument1 rdf:resource="#x" />
</swrl:ClassAtom>
<swrl:ClassAtom>
  <swrl:classPredicate rdf:resource="Style"/>
  <swrl:argument1 rdf:resource="#y" />
</swrl:ClassAtom>
<swrl:IndividualPropertyAtom>
  <swrl:propertyPredicate rdf:resource="artistStyle"/>
  <swrl:argument1 rdf:resource="#x" />
  <swrl:argument2 rdf:resource="#y" />
</swrl:IndividualPropertyAtom>
<swrl:IndividualPropertyAtom>
  <swrl:propertyPredicate rdf:resource="creator"/>
  <swrl:argument1 rdf:resource="#x" />
  <swrl:argument2 rdf:resource="#z" />
</swrl:IndividualPropertyAtom>
</ruleml:body>
```

Analytically (2)

```
<ruleml:head rdf:parseType="Collection">
<swrl:IndividualPropertyAtom>
  <swrl:propertyPredicate rdf:resource="artifactStyle"/>
  <swrl:argument1 rdf:resource="#z" />
  <swrl:argument2 rdf:resource="#y" />
</swrl:IndividualPropertyAtom>
</ruleml:head>
```

Definition and matching head predicates
Specifically :
artistfactStyle(z,y)

```
<ruleml:Imp>
…
</ruleml:Imp>
```

This element allows to say that any link that satisfies the body of the rule must also satisfy the head of the same rule.

XML Concrete Syntax

<swrlx:datarangeAtom>

…

</swrlx:datarangeAtom>

The description in an individual datarange can be a datatype identity, or it can be a set of literals.

Παράδειγμα:

```
<swrlx:datarangeAtom>
  <owlx:Datatype owlx:name="&xsd;int" />
  <ruleml:var>x1</ruleml:var>
</swrlx:datarangeAtom>
```

<ruleml:var>xsd:string</ruleml:var>

It simply defines the existence of a variable. This is taken from the RuleML namespace.

Παράδειγμα:

<ruleml:var>x1</ruleml:var>

XML Concrete Syntax = Combination of OWL XML synatx και RuleML XML syntax

SWRL and Querying: SQWRL

- SWRL is a rules language, not a question formulation language
- However, a rule can previously be treated as a plan matching specification, i.e., a question.
- With built-ins, language conformations of query extensions are possible.
- A SWRL-based query language called SQWRL has been developed.

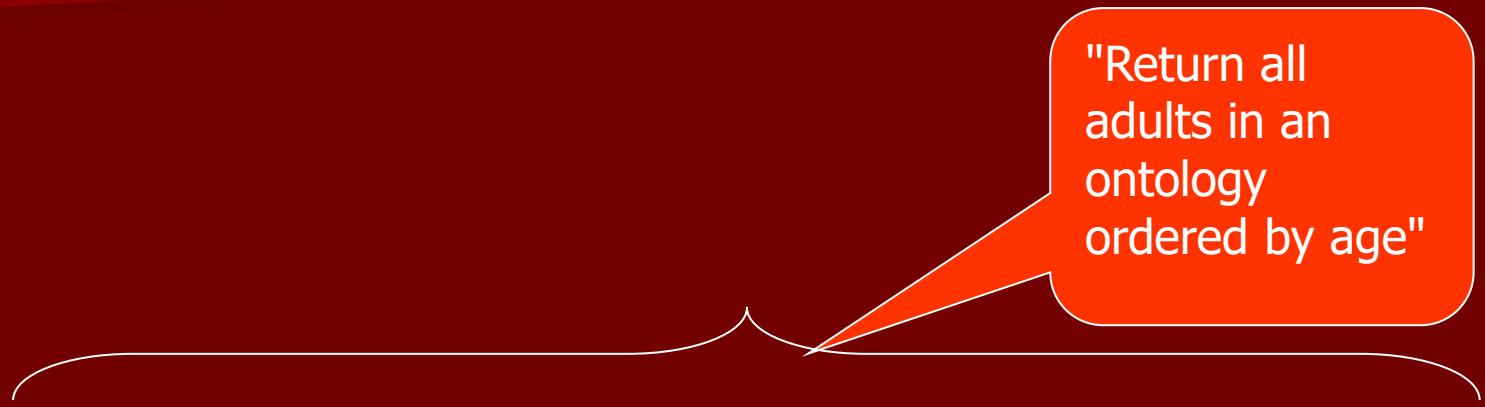
SQWRL Query Example

Query: "Return all adults in an ontology"

Person(?p) ^ hasAge(?p,?age) ^ swrlb:greaterThan(?age,17)

→ sqwrl:select(?p, ?age)

Άλλη SQWRL Query



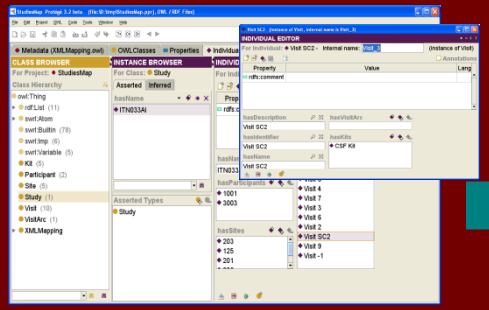
Person(?p) ^ hasAge(?p, ?age) ^ swrlb:greaterThan(?age, 17) →
sqwrl:select(?p) ^ sqwrl:orderBy(?age)

SQWRL as a query language

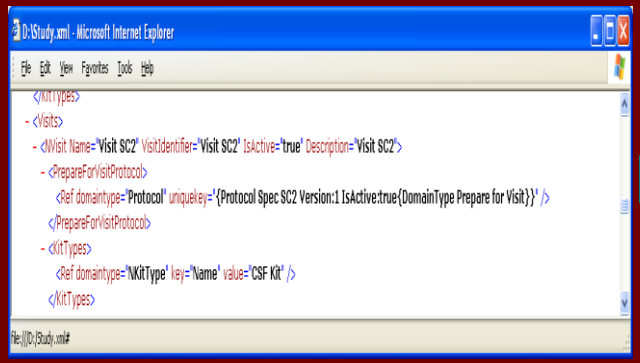
- Clearer semantics than SPARQL
- OWL-based, not RDF-based
- Very extensible through built-ins, eg temporal questions using temporal built-ins

XML Mapping with help of SWRL Mapping Rules

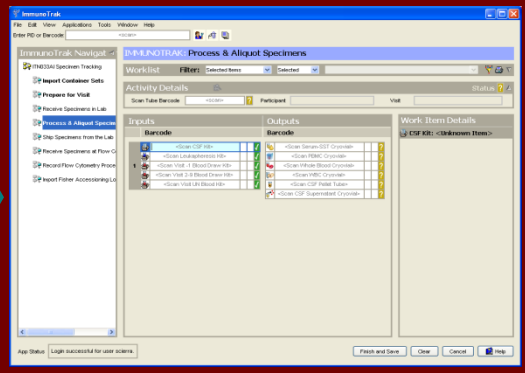
Ontology



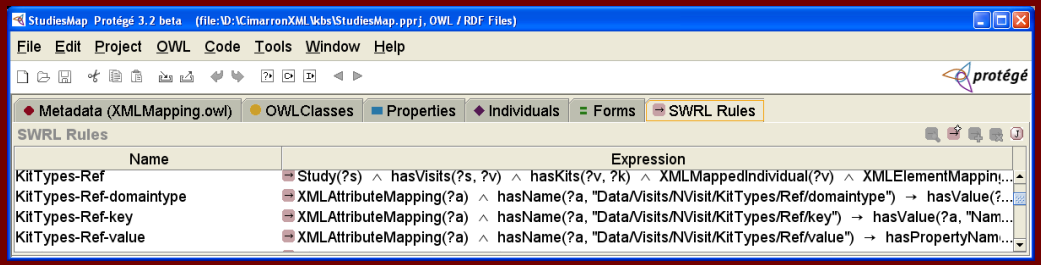
XML Document



Application



SWRL Mapping Rules



SWRL engine: SweetRules v2.0

- SweetRules v2.0 – machine for execution or SWRL rules
  http://xml.coverpages.org/ni2005-04-25-a.html
- SweetRules is a pluggable set of rule tools for RuleML and SWRL that feature interoperability between Prolog, production rules, OWL, CommonRules, Jena-2, and various other languages and inferencing with negation, priorities, and procedural bindings.

Άλλες Μηχανές

- swrl2clips
- Hoolet
- VIS use with JESS
- BaseVISor

APIs

JAXB SWRL API

(http://www.daml.org/rules/proposal/jaxb/)

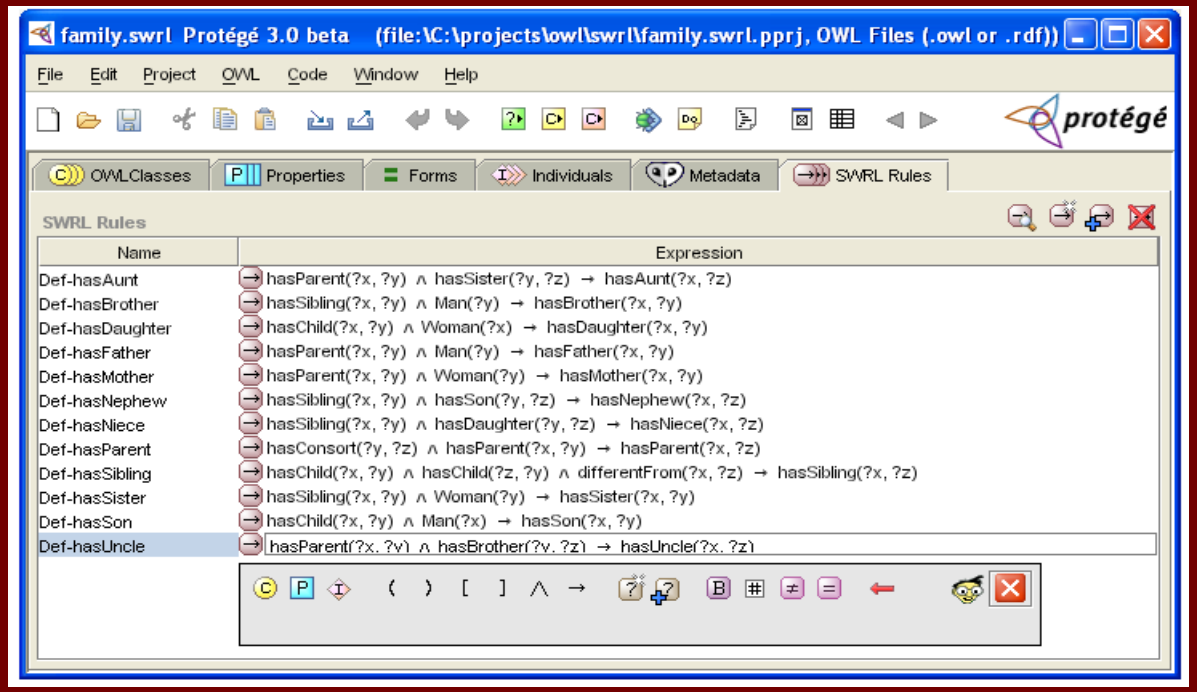
test1.swrlx: sample input for jaxbtest.java

Tools related to SWRL

- SWRLTab
- SQWRL

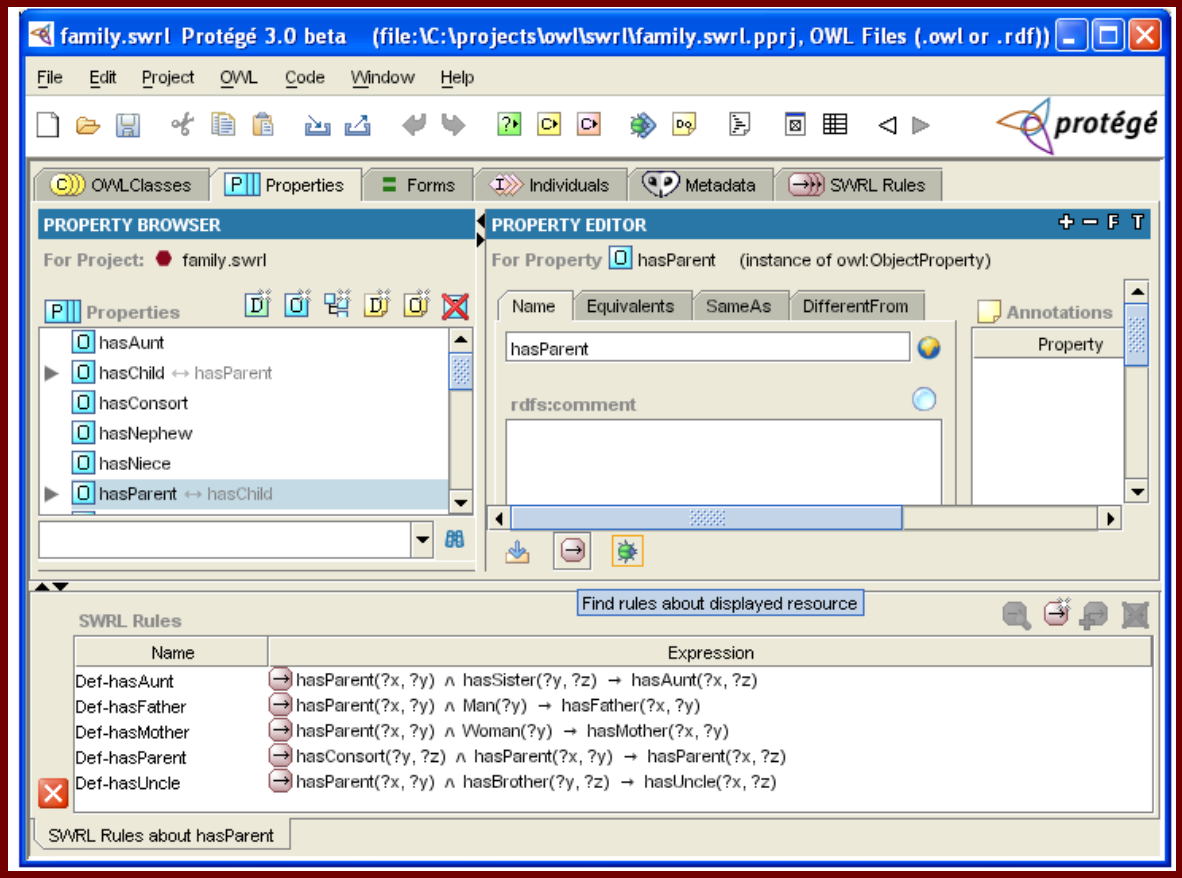
SWRL Editor

The SWRL Editor is an extension of Protege-OWL SWRLTab that supports writing SWRL rules. Can be used to create SWRL rules, open existing SWRL rules, read and write SWRL rules.



SWRL Editor

Next screenshot presents the list of rules that include a reference to hasParent property.



Some more Editors

- VIS RuleVISor
- Protege OWL Plugin, since build 215
- SWeDE