SWRLTab: A Development Environment for working with SWRL Rules In Protégé-OWL

Martin O’Connor
Stanford Medical Informatics, Stanford University
Talk Outline

- Introduction to SWRL
- Using SWRL as an OWL query language
- SWRLTab: a Protégé-OWL development environment for SWRL
Semantic Web Stack
Rule-based Systems are common in many domains

- Engineering: Diagnosis rules
- Commerce: Business rules
- Law: Legal reasoning
- Medicine: Eligibility, Compliance
- Internet: Access authentication
Rule Markup (RuleML) Initiative

- Effort to standardize inference rules.
- RuleML is a markup language for publishing and sharing rule bases on the World Wide Web.
- Focus is on rule interoperation between industry standards.
- RuleML builds a hierarchy of rule sublanguages upon XML, RDF, and OWL, e.g., SWRL
What is SWRL?

• SWRL is an acronym for Semantic Web Rule Language.
• SWRL is intended to be the rule language of the Semantic Web.
• SWRL includes a high-level abstract syntax for Horn-like rules.
• All rules are expressed in terms of OWL concepts (classes, properties, individuals).
• Language FAQ:
  – http://protege.cim3.net/cgi-bin/wiki.pl?SWRLLanguageFAQ
SWRL Characteristics

- W3C Submission in 2004: [http://www.w3.org/Submission/SWRL/](http://www.w3.org/Submission/SWRL/)
- Rules saved as part of ontology
- Increasing tool support: Bossam, R2ML, Hoolet, Pellet, KAON2, RacerPro, SWRLTab
- Can work with reasoners
Example SWRL Rule: Has uncle

\[
\text{hasParent}(\?x, \?y) \land \text{hasBrother}(\?y, \?z) \\
\rightarrow \text{hasUncle}(\?x, \?z)
\]
Example SWRL Rule with Named Individuals: Has brother

\[ \text{Person(Fred)} \land \text{hasSibling(Fred, ?s)} \land \text{Man(?s)} \rightarrow \text{hasBrother(Fred, ?s)} \]
Example SWRL Rule with Literals and Built-ins: is adult?

\[
\text{Person}(?p) \land \text{hasAge}(?p,?age) \land \text{swrlb:greaterThan}(?age,17) \\
\rightarrow \text{Adult}(?p)
\]
Example SWRL Rule with String Built-ins

\[ \text{Person}(\text{?p}) \ ^\wedge \ \text{hasNumber}(\text{?p}, \ ?\text{number}) \ ^\wedge \ \text{swrlb:startsWith}(\text{?number}, \ "+") \rightarrow \ \text{hasInternationalNumber}(\text{?p}, \ \text{true}) \]
Example SWRL Rule with Built-in Argument Binding

Person(?p) ^ hasSalaryInPounds(?p, ?pounds) ^ swrlb:multiply(\textcolor{red}{dollars}, ?pounds, 2.0) -> hasSalaryInDollars(?p, \textcolor{red}{dollars})
Example SWRL Rule with Built-in Argument Binding II

\[
\text{Person(?p)} \land \text{hasSalaryInPounds(?p, ?pounds)} \land \\
\text{swrlb:multiply(2.0, ?pounds, ?dollars)} \rightarrow \\
\text{hasSalaryInDollars(?p, ?dollars)}
\]
Example SWRL Rule with OWL Restrictions

(hasChild >= 1)(?x) → Parent(?x)
Example SWRL Rule with Inferred OWL Restrictions

Parent(?x) → (hasChild >= 1)(?x)
SWRL and Open World Semantics: sameAs, differentFrom

Publication(?p) ^ hasAuthor(?p, ?y) ^ hasAuthor(?p, ?z) ^ differentFrom(?y, ?z) → cooperatedWith(?y, ?z)
SWRL is monotonic: does not Support Negated Atoms

Person(?p) ^ not hasCar(?p, ?c) → CarlessPerson(?p)

Not possible: language does not support negation here

Potential invalidation: what if a person later gets a car?
SWRL is Monotonic: retraction (or modification) not supported

Person(?p) ^ hasAge(?p,?age) ^ swrlb:add(?newage, ?age,1) → hasAge(?p, ?newage)
SWRL is Monotonic: retraction (or modification) not supported

\[
\text{Person(?p) } \land \text{ hasAge(?p,?age) } \land \\
\text{swrlb:add(?newage, ?age,1)} \\
\rightarrow \text{hasAge(?p, ?newage)}
\]

Incorrect: will run forever and attempt to assign an infinite number of values to hasAge property

Potential invalidation: essentially attempted retraction
SWRL is Monotonic: counting not supported

Publication(?p) ^ hasAuthor(?p,?a) ^
<has exactly one hasAuthor value in current ontology>
→ SingleAuthorPublication(?p)

Not expressible: open world applies
Potential invalidation: what if author is added later?
SWRL is Monotonic: counting not supported II

\[
\text{Publication}(?p) ^ \text{(hasAuthor} = 1)(?p) \rightarrow \text{SingleAuthorPublication}(?p)
\]

Closure: though best expressed in OWL
SWRL Semantics

- Based on OWL-DL
- Has a formal semantics
- Completes OWL and fully semantically compatible
- More expressive yet at expense of decidability
- Use OWL if extra expressiveness not required (possible exception: querying)
SWRL and Querying

• SWRL is a rule language, not a query language
• However, a rule antecedent can be viewed as a pattern matching specification, i.e., a query
• With built-ins, language compliant query extensions are possible
Example SWRL Query

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

→ query:select(?p, ?age)
Ordering Query Results

Person(?p) ^ hasAge(?p, ?age) ^ swrlb:greaterThan(?age, 17)
→ query:select(?p, ?age) ^ query:orderBy(?age)
Counting Query Results

Person(?p) ^ hasCar(?p, ?car)
→ query:select(?p) ^
query:count(?car)

Important: no way of asserting count in ontology
Count all Owned Cars in Ontology

\[
\text{Person(?p) \land hasCar(?p, ?c) \rightarrow query:count(?c)}
\]
Count all Cars in Ontology

Car(?c) → query:count(?c)
Aggregation Queries: average age of persons in ontology

- Person(?p) ^ hasAge(?p, ?age) -> query:avg(?age)

Also: query:max, query:min, query:sum
Queries and Rules Can Interact

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

Adult(?a) → query:select(?a)
Example SWRL Query with OWL Restrictions

(hasChild >= 1)(?x) → query:select(?x)
All Built-ins can be used in Queries

`tbox:isDirectSubClassOf(?subClass, Person) - > query:select(?subClass)`

`tbox:isSubPropertyOf(?supProperty, hasName) - > query:select(?subProperty)`

*Note*: use of property and class names as built-in arguments in not OWL DL

*Important*: these built-ins should be used in queries only – inference with them would definitely not be OWL DL
SWRLTab

• A Protégé-OWL development environment for working with SWRL rules
• Supports editing and execution of rules
• Extension mechanisms to work with third-party rule engines
• Mechanisms for users to define built-in method libraries
• Supports querying of ontologies
The SWRLTab is a development environment for working with SWRL rules in Protege-OWL. It supports the editing and execution of SWRL rules. It also provides mechanisms to allow interoperation with a variety of rule engines and the incorporation of user-defined libraries of methods that can be used in rules. Several libraries are provided. These libraries include collections of mathematical and string manipulation routines, in addition to operators that can be used to effectively turn SWRL into a query language.

A introduction to the SWRL language can be found here.

The SWRLTab has several software components:

- **SWRL Editor** The editor supports editing and saving of SWRL rules in an OWL ontology. See the [SWRL Editor FAQ](#) for more details.

- **SWRL Factory** The factory provides high-level Java APIs that support the creation and modification of SWRL rules in an OWL ontology. This API can be used by developers who wish to work with SWRL rules in their applications. See the [SWRL Factory FAQ](#) for more details.

- **SWRL Bridge** The bridge provides the infrastructure necessary to incorporate rule engines into Protege-OWL to execute SWRL rules. See the [SWRL Rule Engine Bridge FAQ](#) for more details. The hope is that bridges for other rule engines will be developed by the Protege-OWL community and then an array of inference mechanism will become available for executing SWRL rules.

- **SWRL Jess Bridge** A bridge for the Jess rule engine is provided in the Protege-OWL distribution. A user interface called the SWRLJessTab is also provided to interact with this bridge.

- **SWRL Built-in Bridge** SWRL built-ins are predicates that accept one or more arguments. These predicates can be used in SWRL rules. The SWRLTab has a sub-component called the built-in bridge that provides a mechanism to define Java implementations of SWRL built-ins. These implementations can then be dynamically loaded by the bridge and invoked from a rule engine.

- **SWRL Built-in Libraries** A number of built-in libraries are provided by the SWRLTab. These include an implementation of the core SWRL built-ins defined in the SWRL Submission and built-ins for querying OWL ontologies. The libraries are documented here.

- **SWRL Query Tab** The query tab provides a graphical interface to display the results of SWRL rules that contain query built-ins. It is documented here.

- **SWRL Query API** This API provides a JDBC-like Java interface to retrieve the result of SWRL rules containing query built-ins. It is documented here.
What is the SWRL Editor?

- The SWRL Editor is an extension to Protégé-OWL that permits the interactive editing of SWRL rules.
- The editor can be used to create SWRL rules, edit existing SWRL rules, and read and write SWRL rules.
- It is accessible as a tab within Protégé-OWL.
<table>
<thead>
<tr>
<th>Name</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Def-hasAunt</td>
<td>→ hasParent(?x, ?y) ∧ hasSister(?y, ?z) → hasAunt(?x, ?z)</td>
</tr>
<tr>
<td>Def-hasBrother</td>
<td>→ hasSibling(?x, ?y) ∧ Man(?y) → hasBrother(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasDaughter</td>
<td>→ hasChild(?x, ?y) ∧ Woman(?y) → hasDaughter(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasFather</td>
<td>→ hasParent(?x, ?y) ∧ Man(?y) → hasFather(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasMother</td>
<td>→ hasParent(?x, ?y) ∧ Woman(?y) → hasMother(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasNephew</td>
<td>→ hasSibling(?x, ?y) ∧ hasSon(?y, ?z) → hasNephew(?x, ?z)</td>
</tr>
<tr>
<td>Def-hasNiece</td>
<td>→ hasSibling(?x, ?y) ∧ hasDaughter(?y, ?z) → hasNiece(?x, ?z)</td>
</tr>
<tr>
<td>Def-hasParent</td>
<td>→ hasConsort(?y, ?z) ∧ hasParent(?x, ?y) → hasParent(?x, ?z)</td>
</tr>
<tr>
<td>Def-hasSibling</td>
<td>→ hasChild(?y, ?x) ∧ hasChild(?y, ?z) ∧ differentFrom(?x, ?z) → hasSibling(?x, ?z)</td>
</tr>
<tr>
<td>Def-hasSister</td>
<td>→ hasSibling(?x, ?y) ∧ Woman(?y) → hasSister(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasSon</td>
<td>→ hasChild(?x, ?y) ∧ Man(?y) → hasSon(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasUncle</td>
<td>→ hasParent(?x, ?y) ∧ hasBrother(?y, ?z) → hasUncle(?x, ?z)</td>
</tr>
</tbody>
</table>
hasChild(?x, ?y) ∧
Woman(?y) → hasDaughter(?x, ?y)
hasChild(?x, ?y) & Woman(?y) → hasDaughter(?x)
hasChild(?x, ?y) ∧ Woman(?y) → hasDaughter(?x, ?y) ∧ swrlb:bind...
Executing SWRL Rules

- SWRL is a language specification
- Well-defined semantics
- Developers must implement engine
- Or map to existing rule engines
- Hence, a bridge…
SWRL Rule Engine Bridge

OWL KB + SWRL

GUI

SWRL Rule Engine Bridge

Rule Engine

Data

Knowledge
SWRL Rule Engine Bridge

• Given an OWL knowledge base it will extract SWRL rules and relevant OWL knowledge.
• Also provides an API to assert inferred knowledge.
• Knowledge (and rules) are described in non Protégé-OWL API-specific way.
• These can then be mapped to a rule-engine specific rule and knowledge format.
• This mapping is developer’s responsibility.
We used the SWRL Bridge to Integrate Jess Rule Engine with Protégé-OWL

• Jess is a Java-based rule engine.
• Jess system consists of a rule base, fact base, and an execution engine.
• Available free to academic users, for a small fee to non-academic users
• Has been used in Protégé-based tools, e.g., JessTab.
<table>
<thead>
<tr>
<th>Name</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Def-hasAunt</td>
<td>hasParent(?x, ?y) ∧ hasSister(?y, ?z) → hasAunt(?x, ?z)</td>
</tr>
<tr>
<td>Def-hasBrother</td>
<td>hasSibling(?x, ?y) ∧ Man(?y) → hasBrother(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasDaughter</td>
<td>hasChild(?x, ?y) ∧ Woman(?y) → hasDaughter(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasFather</td>
<td>hasParent(?x, ?y) ∧ Man(?y) → hasFather(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasMother</td>
<td>hasParent(?x, ?y) ∧ Woman(?y) → hasMother(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasNephew</td>
<td>hasSibling(?x, ?y) ∧ hasSon(?y, ?z) → hasNephew(?x, ?z)</td>
</tr>
<tr>
<td>Def-hasNiece</td>
<td>hasSibling(?x, ?y) ∧ hasDaughter(?y, ?z) → hasNiece(?x, ?z)</td>
</tr>
<tr>
<td>Def-hasParent</td>
<td>hasConsort (?y, ?z) ∧ hasParent(?x, ?y) → hasParent(?x, ?z)</td>
</tr>
<tr>
<td>Def-hasSibling</td>
<td>hasChild(?x, ?y) ∧ hasChild(?y, ?z) → differentFrom(?x, ?z) → hasSibling(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasSister</td>
<td>hasSibling(?x, ?y) ∧ Woman(?y) → hasSister(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasSon</td>
<td>hasChild(?x, ?y) ∧ Man(?y) → hasSon(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasUncle</td>
<td>hasParent(?x, ?y) ∧ hasBrother(?y, ?z) → hasUncle(?x, ?z)</td>
</tr>
</tbody>
</table>

**SVRLJessTab**

See [http://protege.cim3.net/cgi-bin/wiki.pl?SVRLJessTab](http://protege.cim3.net/cgi-bin/wiki.pl?SVRLJessTab) for SVRLJessTab documentation.

Press the "OWL+SWRL->Jess" button to transfer SWRL rules and relevant OWL knowledge to Jess.

Press the "Run Jess" button to run the Jess rule engine.

Press the "Jess->OWL" button to transfer the inferred Jess knowledge to OWL knowledge.

**IMPORTANT.** With the exception of owl:sameAs, owl:differentFrom and owl:DefaultValue, owl:equivalentProperty, and owl:equivalentClass, the Jess rule engine is currently ignoring OWL restrictions. To ensure consistency, a reasoner should be run on an OWL knowledge base before SWRL rules and OWL knowledge are transferred to Jess. Also, if inferred knowledge from Jess is inserted back into an OWL knowledge base, a reasoner should again be executed to ensure that the new knowledge does not conflict with OWL restrictions in that knowledge base.

<table>
<thead>
<tr>
<th>Name</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Def-hasAunt</td>
<td>hasParent(?x, ?y) ∧ hasSister(?y, ?z) → hasAunt(?x, ?z)</td>
</tr>
<tr>
<td>Def-hasBrother</td>
<td>hasSibling(?x, ?y) ∧ Man(?y) → hasBrother(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasDaughter</td>
<td>hasChild(?x, ?y) ∧ Woman(?y) → hasDaughter(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasFather</td>
<td>hasParent(?x, ?y) ∧ Woman(?y) → hasMother(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasHusband</td>
<td>hasSibling(?x, ?y) ∧ Man(?y) → hasHusband(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasNephew</td>
<td>hasSibling(?x, ?y) ∧ hasSon(?y, ?z) → hasNephew(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasNiece</td>
<td>hasSibling(?x, ?y) ∧ hasDaughter(?y, ?z) → hasNiece(?x, ?z)</td>
</tr>
<tr>
<td>Def-hasParent</td>
<td>hasConsort(?y, ?z) ∧ hasParent(?x, ?y) → hasParent(?x, ?z)</td>
</tr>
<tr>
<td>Def-hasSibling</td>
<td>hasChild(?y, ?x) ∧ hasChild(?y, ?z) ∧ differentFrom(?x, ?z) → hasSibling(?x, ?z)</td>
</tr>
<tr>
<td>Def-hasSister</td>
<td>hasSibling(?x, ?y) ∧ Woman(?y) → hasSister(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasSon</td>
<td>hasChild(?x, ?y) ∧ Man(?y) → hasSon(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasUncle</td>
<td>hasParent(?x, ?y) ∧ hasBrother(?y, ?z) → hasUncle(?x, ?z)</td>
</tr>
<tr>
<td>Name</td>
<td>Expression</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Def-hasAunt</td>
<td>hasParent(?x, ?y) ∧ hasSister(?y, ?z) → hasAunt(?x, ?z)</td>
</tr>
<tr>
<td>Def-hasBrother</td>
<td>hasSibling(?x, ?y) ∧ Man(?y) → hasBrother(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasDaughter</td>
<td>hasChild(?x, ?y) ∧ Woman(?y) → hasDaughter(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasFather</td>
<td>hasParent(?x, ?y) ∧ Man(?y) → hasFather(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasMother</td>
<td>hasParent(?x, ?y) ∧ Woman(?y) → hasMother(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasNephew</td>
<td>hasSibling(?x, ?y) ∧ hasSon(?y, ?z) → hasNephew(?x, ?z)</td>
</tr>
<tr>
<td>Def-hasNiece</td>
<td>hasSibling(?x, ?y) ∧ hasDaughter(?y, ?z) → hasNiece(?x, ?z)</td>
</tr>
<tr>
<td>Def-hasParent</td>
<td>hasConsort(?y, ?z) ∧ hasParent(?x, ?y) → hasParent(?x, ?z)</td>
</tr>
<tr>
<td>Def-hasSibling</td>
<td>hasChild(?y, ?x) ∧ hasChild(?y, ?z) ∧ differentFrom(?x, ?z) → hasSibling(?x, ?z)</td>
</tr>
<tr>
<td>Def-hasSister</td>
<td>hasSibling(?x, ?y) ∧ Woman(?y) → hasSister(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasSon</td>
<td>hasChild(?x, ?y) ∧ Man(?y) → hasSon(?x, ?y)</td>
</tr>
<tr>
<td>Def-hasUncle</td>
<td>hasParent(?x, ?y) ∧ hasBrother(?y, ?z) → hasUncle(?x, ?z)</td>
</tr>
</tbody>
</table>

---

(deftemplate Nephew extends Relative)
(deftemplate Son extends Child)
(deftemplate owl:Thing (slot name))
(deftemplate Relative extends Person)
(deftemplate Sibling extends Person)
(deftemplate Aunt extends Relative)
(deftemplate Person extends owl:Thing)
(deftemplate Mother extends Relative)
(deftemplate Niece extends Relative)
(deftemplate Daughter extends Child)
(deftemplate Father extends Parent)
(deftemplate Parent extends Person)
(deftemplate Sister extends Sibling)
(deftemplate Brother extends Sibling)
(deftemplate Child extends Person)
(deftemplate Uncle extends Relative)
(deftemplate Woman extends Person)
(deftemplate Man extends Person)
### SWRL Rules

<table>
<thead>
<tr>
<th>Name</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasAunt &amp; hasSon &amp; hasNephew &amp; hasParent &amp; hasParent &amp; hasParent &amp; hasParent</td>
<td>hasAunt(?x, ?y) ∧ hasSon(?y, ?z) ∧ hasNephew(?x, ?z) ∧ hasParent(?x, ?y) ∧ hasParent(?y, ?z) ∧ hasParent(?z, ?w)</td>
</tr>
<tr>
<td>hasBrother &amp; hasSister &amp; hasParent &amp; hasParent &amp; hasParent &amp; hasParent</td>
<td>hasBrother(?x, ?y) ∧ hasSister(?y, ?z) ∧ hasParent(?x, ?y) ∧ hasParent(?y, ?z) ∧ hasParent(?z, ?w)</td>
</tr>
</tbody>
</table>
Outstanding Issues

• SWRL Bridge does not know about all OWL restrictions:
  – Contradictions with rules possible!
  – Consistency must be assured by the user incrementally running a reasoner.
  – Hard problem to solve in general.

• Integrated reasoner and rule engine would be ideal.

• Possible solution with Pellet, KAON2.
SWRL Built-in Bridge

- SWRL provides mechanisms to add user-defined predicates, e.g.,
  \[ \text{Person(?p)} \land \text{hasAge(?p, ?age)} \land \\
  \text{swrlb:greaterThanOrEqual(?age, 18) -> Adult(?p)} \]

- These built-ins could be implemented by each rule engine.

- However, the SWRL Bridge provides a dynamic loading mechanism for Java-defined built-ins.

- Can be used by any rule engine implementation.
Defining a Built-in in Protégé-OWL

• Describe library of built-ins in OWL using definition of `swrl:BuiltIn` provided by SWRL ontology.
• Provide Java implementation of built-ins and wrap in JAR file.
• Load built-in definition ontology in Protégé-OWL. Put JAR in plugins directory.
• Built-in bridge will make run-time links.
Example: defining `stringEqualIgnoreCase` from Core SWRL Built-ins Library

- Core SWRL built-ins defined by:
  - [http://www.w3.org/2003/11/swrlb](http://www.w3.org/2003/11/swrlb)
- Provides commonly needed built-ins, e.g., add, subtract, string manipulation, etc.
- Normally aliased as ‘swrlb’.
- Contains definition for `stringEqualIgnoreCase`
Example Implementation Class for Core SWRL Built-in Methods

```java
package edu.stanford.smi.protegex.owl.swrl.bridge.builtins.swrlb;

import edu.stanford.smi.protegex.owl.swrl.bridge.builtins.*;
import edu.stanford.smi.protegex.owl.swrl.bridge.exceptions.);

public class SWRLBuiltInLibraryImpl extends SWRLBuiltInLibrary
{
    public SWRLBuiltInMethodsImpl() { ...}
    public void reset() {...}

    public boolean stringEqualIgnoreCase(List arguments) throws BuiltInException { ... }
    ....
} // SWRLBuiltInLibraryImpl
```
Example Implementation for Built-in `swrlb:stringEqualIgnoreCase`

```java
public boolean stringEqualIgnoreCase(List<Argument> arguments) throws BuiltInException {
    SWRLBuiltInUtil.checkNumberOfArgumentsEqualTo(2, arguments.size());

    String argument1 = SWRLBuiltInUtil.getArgumentAsAString(1, arguments);
    String argument2 = SWRLBuiltInUtil.getArgumentAsAString(2, arguments);

    return argument1.equalsIgnoreCase(argument2);
} // stringEqualIgnoreCase
```
Invocation from Rule Engine

• Use of `swrlb:stringEqaulIgnoreCase` in rule should cause automatic invocation.
• SWRL rule engine bridge has an invocation method.
• Takes built-in name and arguments and performs method resolution, loading, and invocation.
• Efficiency a consideration: some methods should probably be implemented natively by rule engine, e.g., add, subtract, etc.
The SWRL Query Built-In Library is one of the SWRLTabBuiltInLibraries. It defines a set of built-ins that can be used in SWRL rules to query OWL ontologies. The built-ins in this library can be used to effectively turn SWRL into a query language. They provide SQL-like operations retrieve knowledge from an OWL ontology. The resulting query language does not alter SWRL's semantics and uses the standard presentation syntax supported by the SWRLTab.

These built-ins are defined in the SWRL Query Ontology. It has the default namespace prefix query. A copy of this file can be found in the standard Protege-OWL repositories, and can be imported through the 'Import Ontology' option in the Metadata tab. Java implementations for these built-ins are also included in the Protege-OWL 3.3 beta, build XXX distribution.

Basic Queries

Assume we have a simple ontology with classes person, which has subclasses male and female with associated properties hasAge and hasName, and a class Car, that can be associated with individual of class person through the hasCar property.

Here, for example, is a simple SWRL query to extract all known persons in an ontology whose age is less than 25, together with their ages:

- $\text{Person(?p)} \land \text{hasAge(?p, ?a)} \land \text{swrlb:lessThan(?a, 25)} \rightarrow \text{query:select(?p, ?a)}$
Def-hasAunt  hasParent(?x, ?y) ∧ hasSister(?y, ?z) → hasAunt(?x, ?z)
Def-hasBrother hasParent(?x, ?y) ∧ Man(?y) → hasBrother(?x, ?y)
Def-hasDaughter hasChild(?x, ?y) ∧ Woman(?y) → hasDaughter(?x, ?y)
Def-hasFather hasParent(?x, ?y) ∧ Man(?y) → hasFather(?x, ?y)
Def-hasMother hasParent(?x, ?y) ∧ Woman(?y) → hasMother(?x, ?y)
Def-hasNephew hasSibling(?x, ?y) ∧ hasSon(?y, ?z) → hasNephew(?x, ?z)
Def-hasNiece hasSibling(?x, ?y) ∧ hasDaughter(?y, ?z) → hasNiece(?x, ?z)
Def-hasParent hasConsort(?y, ?x) ∧ hasParent(?x, ?y) → hasParent(?x, ?y)
Def-hasSibling hasChild(?y, ?x) ∧ hasChild(?y, ?z) ∧ differentFrom(?x, ?z) → hasSibling(?x, ?z)
Def-hasSister hasSibling(?x, ?y) ∧ Woman(?y) → hasSister(?x, ?y)
Def-hasSon hasChild(?x, ?y) ∧ Man(?y) → hasSon(?x, ?y)
Def-hasUncle hasParent(?x, ?y) ∧ hasBrother(?y, ?z) → hasUncle(?x, ?z)
Query-name Man(?x) ∧ name(?x, ?n) → query:select(?x, ?n)
Query-child Man(?man) ∧ name(?man, ?n1) ∧ hasChild(?man, ?child) ∧ name(?child, ?n2) → query:select(?n1, ?n2)
Query-age Person(?p) ∧ name(?p, ?n) → query:select(?p, ?n)
SWRLQueryTab

- Available as part of Protégé-OWL SWRLTab in current Protégé-3.3.1
- Low-level JDBC-like API for use in embedded applications
- Can use any existing rule engine back end
Other Built-in Libraries

A number of built-in libraries are provided by the SWRLETab. These include:

- **Core SWRL Built-Ins Library**: Contains implementations for the core built-ins defined by the SWRL Submission. It is documented here.

- **Query Built-In Library**: Defines a set of built-ins that can be used in SWRL rules to query OWL ontologies. It is documented here.

- **ABox Built-Ins Library**: Defines built-ins that can be used to query an ABox. It is documented here.

- **TBox Built-Ins Library**: Defines built-ins that can be used to query a TBox. It is documented here.

- **Extensions Built-ins Library**: Defines some experimental built-ins that can be used to increase the expressivity of SWRL. It is documented here.

New SWRL built-in libraries can be defined using the SWRLBuiltInBridge.
SWRLTab Java APIs

• The SWRLTab provides APIs for all components
• These APIs are accessible to all OWL Protégé-OWL developers.
• Third party software can use these APIs to work directly with SWRL rules and integrate rules into their applications
• Fully documented in SWRLTab Wiki
Future Plans

• Port to Protégé 4
• Integrated reasoner/inference support, most likely with Pellet
• Dynamic relational-OWL mapping for inferencing and querying (static already available with Datamaster)
• SQWRL (‘squirrel’): enhanced query support – negation, disjunction