Supporting Rule System Interoperability on the Semantic Web with SWRL

Martin O'Connor[1], Holger Knublauch[1], Samson Tu[1], Benjamin Grosof[2], Mike Dean[3], William Grosso[4], and Mark Musen[1]

[1] Stanford Medical Informatics,
Stanford University School of Medicine, Stanford, CA 94305
musen@smi.stanford.edu
[2] Sloan School of Management, MIT, Cambridge, MA 02142
bgrossof@mit.edu
[3] BBN Technologies, Ann Arbor, MI 48103
mdean@bbn.com
[4] Echopass Corp., San Francisco, CA 95105
wgrosso@echopass.com

Abstract. Rule languages and rule systems are widely used in business applications including computer-aided training, diagnostic fact finding, compliance monitoring, and process control. However, there is little interoperability between current rule-based systems. Interoperation is one of the main goals of the Semantic Web, and developing a language for sharing rules is often seen as a key step in reaching this goal. The Semantic Web Rule Language (SWRL) is an important first step in defining such a rule language. This paper describes the development of a configurable interoperation environment for SWRL built in Protégé-OWL, the most widely-used OWL development platform. This environment supports both a highly-interactive, full-featured editor for SWRL and a plugin mechanism for integrating third party rule engines. We have integrated the popular Jess rule engine into this environment, thus providing one of the first steps on the path to rule integration on the Web.

1 Introduction

Many business processes, such as workflow management, computer aided training, compliance monitoring, diagnostic fact finding, and process control, are often best modeled using a declarative approach, leading to a very active commercial interest in rule-based systems. However, interoperability among the multitude of current rule-based systems is limited. Given that interoperability is one of the primary goals of the Semantic Web and that rules are a key part of those goals, there has been significant recent interest in standardization.[1] The goal of sharing rule bases and processing them with different rule engines has resulted in RuleML, SWRL, Metalog, and ISO Prolog, and other standardization efforts.

One of the key steps to rule interoperation on the Web is SWRL[2] which was designed to be the rule language of the Semantic Web. SWRL is based on a combination of the

OWL DL and OWL Lite sublanguages of the OWL Web Ontology Language[3] the Unary/Binary Datalog[4] sublanguages of the Rule Markup Language. SWRL allows users to write Horn-like rules expressed in terms of OWL concepts to reason about OWL individuals. The rules can be used to infer new knowledge from existing OWL knowledge bases.

The SWRL Specification[5] does not impose restrictions on how reasoning should be performed with SWRL rules. Thus, investigators are free to use a variety of rule engines to reason with the SWRL rules stored in an OWL knowledge base. They are also free to implement their own editing facilities to create SWRL rules. In this way, SWRL provides a convenient starting point for integrating rule systems to work with the Semantic Web.

To this end, we have developed the Protégé SWRL Editor, a full-featured highly interactive open-source rule editor for SWRL. This editor operates within Protégé-OWL[6] and is tightly integrated with it. It adopts the look-and-feel of Protégé-OWL and allows users to seamlessly switch between SWRL rule editing and normal OWL editing of OWL entities. Users can also easily incorporate OWL entities into rules they are authoring.

One of the main goals of the SWRL Editor is to permit interoperability between SWRL and existing rule engines. An important component of this interoperability is the editor's mechanism for supporting tight integration with rule engines. This mechanism is supported by a subsystem called the Protégé SWRL Factory. The SWRL Factory supports a rule engine plugin mechanism that permits API-level interoperation with existing rule engines. It also allows developers to access real estate on the SWRL Editor tab, which allows the interface for the rule engine to coexist with the SWRL Editor. Developers integrating an existing rule engine with the SWRL Editor have full control of the area inside the panel.

Of course, a looser form of interoperation can also occur at the OWL knowledge base level. Investigators are free to use the SWRL rules created by the editor and stored in OWL files as input to their rule engines. Researchers in the SweetRules[7]project, for example, have already used rules created by the SWRL Editor to perform inference using a Jena 2-based[8]rule engine.

We used the SWRL Factory mechanism to integrate the Jess rule engine with the SWRL Editor. With Jess, users can run SWRL rules interactively to create new OWL concepts and then insert them into an OWL knowledge base. SWRL, coupled with Jess, can provide a rich rule-based reasoning facility for the Semantic Web and can serve as a starting point for further rule integration efforts.

2 Semantic Web Rule Language

In common with many other rule languages, SWRL rules are written as antecedent-consequent pairs. In SWRL terminology, the antecedent is referred to as the rule *body* and the consequent is referred to as the *head*. The head and body consist of a conjunction of one or more *atoms*. At present, SWRL does not support more complex logical combinations of atoms.

SWRL rules reason about OWL individuals, primarily in terms of OWL classes and properties. For example, a SWRL rule expressing that a person with a male sibling has a brother would require capturing the concepts of 'person', 'male', 'sibling' and 'brother' in OWL. Intuitively, the concept of person and male can be captured using an OWL class called `Person` with a subclass `Man`; the sibling and brother relationships can be expressed using OWL properties `hasSibling` and `hasBrother`, which are attached to `Person`. The rule in SWRL would then be:

```
Person (?x1) ^ hasSibling(?x1,?x2) ^ Man(?x2) →
hasBrother(?x1,?x2)
```

Executing this rule would have the effect of setting the `hasBrother` property to `x2` in the individual that satisfies the rule, named `x1`.

SWRL rules can also refer explicitly to OWL individuals. For example, the following example is a variant of the above rule, inferring that a particular individual Fred has a brother:

```
Person(Fred) ^ hasSibling(Fred,?x2) ^ Man(?x2) →
hasBrother(Fred,?x2)
```

In this case `Fred` is the name of an OWL individual.

SWRL also supports data literals. For example, assuming an individual has a `hasAge` property, it is possible to ask if Fred has a 40 year-old brother:

```
Person(Fred) ^ hasSibling(Fred,?x2) ^ Man(?x2) ^
hasAge(?x2,40) → has40YearOldBrother(Fred,?x2)
```

String literals — which are enclosed in single quotes — are also supported.

SWRL also supports the common same-as and different-from concepts. For example, the SWRL `sameAs` atom can determine if two OWL individuals Fred and Frederick are the same individual:

```
sameAs(Fred, Frederick)
```

Similarly, the `differentFrom` atom can be used to express that two OWL individuals are not the same.

SWRL also has an atom to determine if an individual, property, or variable is of a particular type. For example, the following example determines if variable `x` is of type unsigned integer:

```
xsd:unsignedInt(?x)
```

These atoms — which are called data range atoms in SWRL — must be preceded by the '`xsd:`' namespace qualifier. The type specified must be an XML Schema data type.

A second form of a data range atom can be used to express one-of relationships in SWRL. For example, the following SWRL atom indicates that variable `x` must be one of 3, 4 or 5:

```
[3, 4, 5](?x)
```

SWRL also supports a range of built-in predicates, which greatly expand its expressive power. SWRL built-ins are predicates that accept several arguments. They are described in detail in the SWRL Built-in Specification[9]. The simplest built-ins are comparison operations. For example, the `greaterThan` built-in determines if an individual has an older brother.

```
hasBrother(?x1,?x2) ^ hasAge(?x1,?age1) ^
hasAge(?x2,?age2) ^ swrlb:greaterThan(?age2,?age1) →
hasOlderBrother(?x1,?x2)
```

All built-ins in SWRL must be preceded by the namespace qualifier '`swrlb:`'.

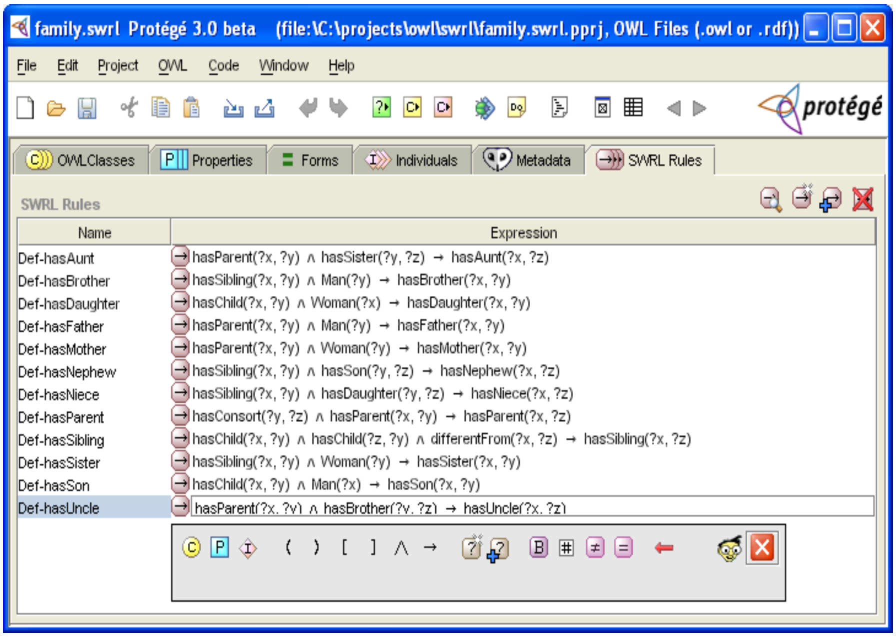
Finally, SWRL supports more complex mathematical built-ins. For example, the following rule determines if an individual has a brother who is exactly 10 years older:

```
hasBrother(?x1,?x2) ^ hasAge(?x1,?age1) ^
hasAge(?x2,?age2) ^ swrlb:subtract(10,?age2,?age1) →
hasDecadeOlderBrother(?x1,?x2)
```

The SWRL Built-in Ontology[10] describes the range of built-ins supported by SWRL. In addition to mathematical built-ins, there are built-ins for strings, dates, and lists. Additions may be made to this namespace in the future so the range of built-ins supported by SWRL can grow.

3 The Protégé SWRL Editor

The Protégé SWRL Editor is an extension to Protégé-OWL that permits interactive editing of SWRL rules. Users can create, edit, and read/write SWRL rules. With the exception of arbitrary OWL expressions (see Section 6), the SWRL Editor supports the full set of language features outlined in the current SWRL Specification. It is tightly integrated with Protégé-OWL and is primarily accessible through a tab within it. When editing rules, users can directly refer to OWL classes, properties, and individuals within an OWL knowledge base. They also have direct access to the full

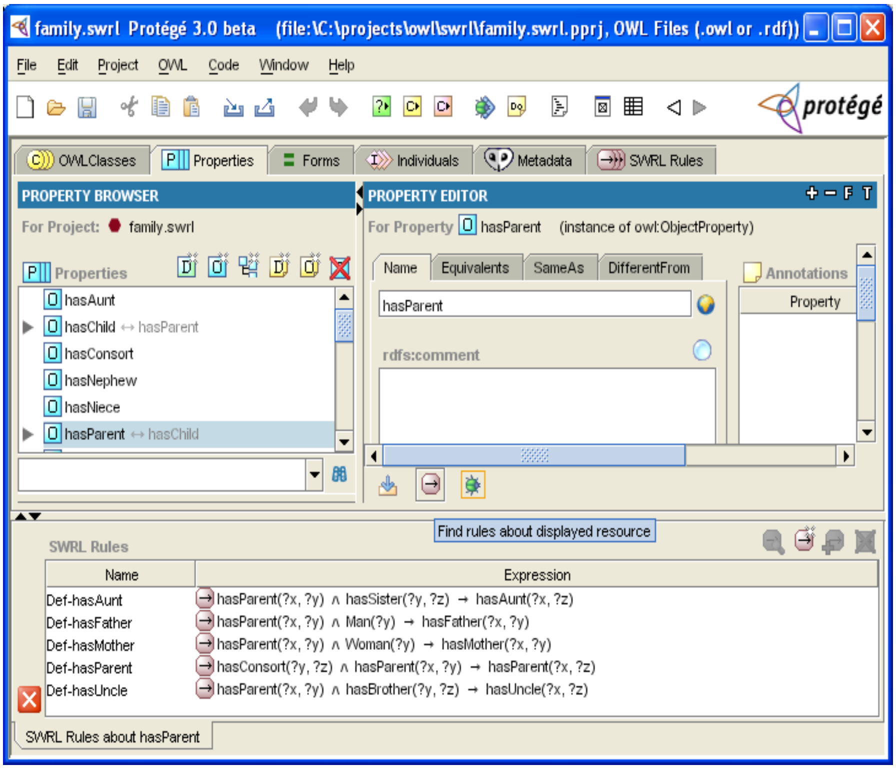


Name	Expression
Def-hasAunt	hasParent(?x, ?y) ∧ hasSister(?y, ?z) → hasAunt(?x, ?z)
Def-hasBrother	hasSibling(?x, ?y) ∧ Man(?y) → hasBrother(?x, ?y)
Def-hasDaughter	hasChild(?x, ?y) ∧ Woman(?x) → hasDaughter(?x, ?y)
Def-hasFather	hasParent(?x, ?y) ∧ Man(?y) → hasFather(?x, ?y)
Def-hasMother	hasParent(?x, ?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, ?z) ∧ hasParent(?x, ?y) → hasParent(?x, ?z)
Def-hasSibling	hasChild(?x, ?y) ∧ hasChild(?z, ?y) ∧ differentFrom(?x, ?z) → hasSibling(?x, ?z)
Def-hasSister	hasSibling(?x, ?y) ∧ Woman(?y) → hasSister(?x, ?y)
Def-hasSon	hasChild(?x, ?y) ∧ Man(?x) → hasSon(?x, ?y)
Def-hasUncle	hasParent(?x, ?y) ∧ hasBrother(?y, ?z) → hasUncle(?x, ?z)

Fig. 1. The Protégé SWRL Rules tab in Protégé-OWL. The SWRL Rules tab provides a tabular listing of all SWRL rules in an OWL knowledge base. These rules can be edited in place or with a multi-line interactive editor, which can be popped up.

set of built-ins described in the SWRL Built-in Ontology and to the full range of XML Schema data types. Figure 1 shows a screenshot of the Protégé SWRL Rules tab. The SWRL Editor also supports inference with SWRL rules using the Jess[11] rule engine (see Section 5). Documentation for the editor is available in the Protégé SWRL Editor FAQ[12].

The SWRL Editor is automatically enabled in Protégé-OWL when loading any OWL knowledge base that imports the SWRL Ontology[13]. It is disabled by default if a loaded knowledge base does not import this ontology. A user can use Protégé-OWL's configuration menu to enable this tab for a knowledge base that does not import the SWRL Ontology; he will then be given an option to import this ontology so that all future loads of the knowledge base will activate the SWRL Editor.


family.swrl Protégé 3.0 beta (file:\C:\projects\owl\swrl\family.swrl.pprj, OWL Files (.owl or .rdf))
File Edit Project OWL Code Window Help
protégé
OWLClasses Properties Forms Individuals Metadata SWRL Rules
PROPERTY BROWSER
PROPERTY EDITOR
Properties
Name Equivalents SameAs DifferentFrom
Annotations
Property
rdfs:comment
Find rules about displayed resource
SWRL Rules
SWRL Rules about hasParent

For Project: family.swrl

For Property hasParent (instance of owl:ObjectProperty)

Name: hasParent

- hasAunt
- hasChild ↔ hasParent
- hasConsort
- hasNephew
- hasNiece
- hasParent ↔ hasChild

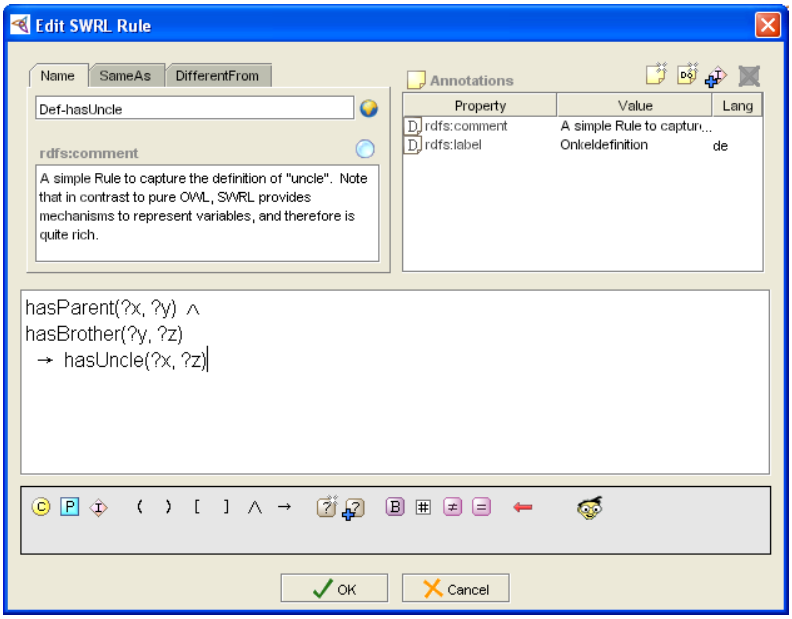
Name	Expression
Def-hasAunt	hasParent(?x, ?y) ∧ hasSister(?y, ?z) → hasAunt(?x, ?z)
Def-hasFather	hasParent(?x, ?y) ∧ Man(?y) → hasFather(?x, ?y)
Def-hasMother	hasParent(?x, ?y) ∧ Woman(?y) → hasMother(?x, ?y)
Def-hasParent	hasConsort(?y, ?z) ∧ hasParent(?x, ?y) → hasParent(?x, ?z)
Def-hasUncle	hasParent(?x, ?y) ∧ hasBrother(?y, ?z) → hasUncle(?x, ?z)

Fig. 2. Protégé-OWL Properties tab showing all SWRL rules that refer to the hasParent property. In common with the SWRL Editor tab, displayed rules can be edited in place or with a multi-line interactive editor.

There are two ways of interacting with the SWRL Editor in Protégé-OWL:

1. The primary mechanism is through the SWRL Rules tab (see Figure 1). This tab shows all the SWRL rules in a loaded OWL knowledge base in tabular form.

2. A second mechanism allows users to find rules relating to a selected OWL class, property, or individual in the respective Protégé-OWL tabs for those entities. For example, if a user is examining a class using the OWL Classes tab, he can display a list of SWRL rules referring to that class. The same mechanism applies in the properties and individuals tabs. Figure 2 shows a Protégé-OWL Properties tab displaying all SWRL rules that refer to a selected property, hasParent.



Def-hasUncle

rdfs:comment

A simple Rule to capture the definition of "uncle". Note that in contrast to pure OWL, SWRL provides mechanisms to represent variables, and therefore is quite rich.

Property	Value	Lang
rdfs:comment	A simple Rule to captur...	
rdfs:label	Onkeldefinition	de

```
hasParent(?x, ?y) ∧
hasBrother(?y, ?z)
 → hasUncle(?x, ?z)
```

Fig. 3. SWRL Multi-Line Editor Dialog. The icon panel provides selection dialog boxes to select things including OWL classes, properties, and individuals. It also includes shortcuts and selection dialog boxes for various other entities.

There are two editing modes for rules. Users can edit them in place in the table containing them, or they can pop up a multi-line editor (see Figure 3). The difference between the two modes is primarily visual. The same interaction mechanisms apply in both modes, though the multi-line editor has some additional options.

The SWRL Editor allows users to enter rules completely as text. However, it also allows users to select OWL entities from a currently loaded knowledge base and insert them into the rule being edited. This task is performed via an icon panel. The icon panel provides access to selection dialog boxes, select OWL classes, properties, and individuals. It also includes selection dialog boxes for SWRL built-ins, XML Schema data types, and shortcuts for various other entities.

The SWRL Editor performs syntactic and semantic checking as a rule is being entered. It ensures that each rule is syntactically correct and also ensures that any references to OWL entities are valid. It will also ensure that any variables referred to in a rule consequent are present in the head. If a user makes a mistake while entering a rule, the rule text entry box grays out and a textual explanation of the error is presented. A user can continue typing and immediately fix the error or fix it later. The editor does not allow users to save incomplete or erroneous rules.

The editor also has convenience features such as auto-completion. Pressing the tab key while editing an OWL entity, built-in, or XML Schema data type name auto-completes a name if it has a unique expansion. If the name is not unique, the software brings up a selection box containing a list of possible completions.

4 The SWRL Editor and Interoperability

SWRL rules are stored as OWL individuals with their associated knowledge base when an open OWL project is saved. The classes that describe these individuals are described by the SWRL Ontology. The highest level class in this Ontology is `swrl:Imp`, which is used to represent a single SWRL rule. It contains an antecedent part, which is referred to as the *body*, and a consequent part, which is referred to as the *head*. Both the body and head are instances of the `swrl:AtomList` class, which represents a list containing rule atoms. The abstract `swrl:Atom` class is used to represent a rule atom. The various types of atoms described in the SWRL Specification are described by subclasses of this class. The SWRL Ontology also includes a class called `swrl:Builtin` to describe built-ins and a class called `swrl:Variable` that can be used to represent variables.

The SWRL Editor comes packaged with a Java API called the Protégé SWRL Factory, which allows developers to directly manipulate SWRL rules in an OWL knowledge base. The SWRL Factory provides a mapping from the OWL individuals representing SWRL rules to analogous Java instances. It also provides Java classes representing SWRL classes in the SWRL Ontology and mechanisms to create run-time instances of classes that mirror individuals in an OWL knowledge base. It is used internally by the SWRL Editor. However, it is accessible to all Protégé-OWL developers. SWRL Plugin developers can base their work directly on the classes created by this factory and can, for example, use it to integrate existing rule engines with Protégé-OWL. Indeed, this API could also be used to create new SWRL rule editors.

Each class described in the SWRL Ontology has a direct Java equivalent SWRL Factory class to represent it. The factory has utility functions to create Java instances of all these classes. When one Java instance is created, an equivalent OWL individual is also created in the knowledge base. Java instances mirroring existing OWL individuals can also be created. For example, the factory provides methods to create `SWRLImp` and `SWRLAtomList` Java classes that can be used to represent instances of the equivalent `swrl:imp` and `swrl:AtomList` OWL classes. Documentation of the SWRL Factory API is outlined in the Protégé SWRL Factory FAQ[14].

The SWRL Editor itself has no inference capabilities. It simply allows users to edit SWRL rules and save and load them to and from OWL knowledge bases. However, the SWRL Factory supports a rule engine plugin mechanism that permits API-level interoperation with existing rule engines. It also allows developers to access real estate on the SWRL Editor tab, which allows the interface for the rule engine to coexist with the SWRL Editor. Developers integrating an existing rule engine with the SWRL Editor have full control of the area inside the panel.

The SWRL Factory provides this functionality in a class called `SWRLRuleEngineAdapter`. The primary call provided by this class is `getSWRLTabRealEstate`, which returns a Java `JPanel` Swing object representing an area of the screen in the SWRL tab. This class also provides a method called `setRuleEngineIcon` that allows users to access the rule engine interactively. This icon is displayed on the top right of the rule table in the SWRL tab and can be used to toggle the screen real estate of the associated rule engine. If several rule engines are available, multiple icons are displayed. However, only one rule engine can be active at a time.

The SWRL Factory also provides a listening mechanism that allows users to register for rule creation, modification, and deletion events. Thus, when a SWRL rule is modified, a loaded rule engine can maintain an up-to-date image of the SWRL rule base automatically. Rule engine developers also have access to the Protégé event mechanism so that they can be immediately informed of modifications to a loaded OWL knowledge base. Thus, users can configure a rule engine so that it is synchronized with the SWRL rule base and the OWL knowledge base, and so that it performs immediate inference when changes are made to them. This approach has been used successfully in the Protégé environment for both the Jess[15] and Algernon[16] rule engines. These characteristics allow the SWRL factory to provide a bridge for third-party rule engines to interact with the SWRL Editor at run time, allowing users of these engines to experience seamless interaction between the SWRL Editor and the rule engine.

Of course, developers of rule engines may prefer a less dynamic relationship between the knowledge base and the rule engine. For example, instead of continually updating rule engine state in response to modifications to the associated SWRL rules or the loaded OWL knowledge base, a more user-controlled interaction may be desired.

In this regard, users can choose step-by-step control of the inference process. Thus, a user can incrementally control loading of SWRL rules and OWL knowledge into a rule engine, execution of those rules on the knowledge, the review of the results, and storing concluded results back into the OWL knowledge base. This approach may be preferable during early development and testing of a set of rules when the consequences of rules firing may not be obvious. Erroneous rules could easily create hundreds or more incorrect relationships between OWL entities.

5 Integrating the SWRL Editor and the Jess Rule Engine

A large number of rule engines work well with Java[17], and many are available as open source software. Some of the most popular engines include Jess, Algernon[18] and SweetRules. We chose Jess as the first integration candidate for the SWRL Editor because it works seamlessly with Java, has an extensive user base, is well documented, and is very easy to use and configure. Several research teams have also demonstrated that mappings between SWRL and Jess[19,20,21] and between RuleML and Jess[22] are possible. Jess provides both an interactive command line interface and a Java-based API to its rule engine. This engine can be embedded in Java applications and provides a flexible two-way run-time communication between Jess rules and Java. It is not open source but can be downloaded free for a 30-day evaluation period and is available free to academic users.

The Jess system consists of a rule base, a fact base, and an execution engine. The execution engine matches facts in the fact base with rules in the rule base. These rules can assert new facts and put them in the fact base or execute Java functions.

SWRL rules reason about OWL individuals, primarily in terms of OWL classes and properties. When a SWRL rule is fired, it can create new classifications for existing individuals. For example, if a rule consequent asserts that an individual is to be classified as a member of a particular class, that individual must be made a member of that class within OWL when the rule fires. Similarly, if a SWRL rule asserts that two individuals are related via a particular property, then that property must be associated with each individual that satisfies the rule.

Thus, four main tasks must be performed to allow Jess to interoperate with the SWRL Editor: (1) represent relevant knowledge about OWL individuals as Jess facts; (2) represent SWRL rules as Jess rules; (3) perform inference using those rules and reflect the results of that inference in an OWL knowledge base; and (4) control this interaction from a graphical interface.

5.1 Representing OWL Concepts as Jess Knowledge

Relevant knowledge about OWL individuals must be represented as Jess knowledge. The two primary properties that must be represented are 1) the classes to which an individual belongs and 2) the properties the individual possesses. Same-as and different-from information about these individuals must also be captured.

The Jess template facility provides a mechanism for representing an OWL class hierarchy. A Jess template hierarchy can be used to model an OWL class hierarchy using a Jess slot to hold the name of the individual belonging to the hierarchy. Thus, for example, a user must define a Jess template to represent the `owl:Thing` class:

```
(deftemplate OWLThing (slot name))
```

A hierarchy representing a class `Man` that subclasses a direct subclass of `owl:Thing` called `Person` could then be represented as follows in Jess:

```
(deftemplate Person extends OWLThing)[1]

(deftemplate Man extends Person)
```

Using this template definition, the OWL individual can be asserted as a member of the class `Man`:

```
(assert (Man (name Fred)))
```

OWL property information can be directly represented as Jess facts. For example, the information that an individual Fred is related to individual Joe through the `hasUncle` property can be directly asserted using:

```
(assert (hasUncle Fred Joe))
```

Similarly, OWL's same-as and different-from relationships between individuals can be directly represented in Jess. For example, the information that Fred and Frederick are the same OWL individual can be expressed:

```
(assert (sameAs Fred Frederick))
```

[1] In practice, a fully qualified namespace would precede each entity name, but we have omitted it here for clarity.

XML Schema data types can be represented using the same approach. For example, the information that individual x is an unsigned integer can be written:

```
(assert (xsd:unsignedInt ?x))
```

Finally, built-ins can be represented using the Jess 'test' mechanism. For example, the SWRL built-in `greaterThan` applied to two integers can be written:

```
(test (> 10 34))
```

5.2 Representing SWRL Rules as Jess Rules

The representation of SWRL rules in Jess using these facts is relatively straightforward. For example, take the following SWRL rule:

```
Person(?x) ^ Man(?y) ^ hasSibling(?x,?y) ^
hasAge(?x,?age1) ^ hasAge(?y,?age2) ^
swrlb:greaterThan(?age2,?age1) ->
hasOlderBrother(?x,?y)
```

This rule can be represented in Jess — using the representation of individuals outlined above — as:

```
(defrule aRule (Person (name ?x))(Man (name ?y))
               (hasSibling ?x ?y)(hasAge ?x ?age1)
               (hasAge ?y ?age2)(test (> ?age2 ?age1))
 => (assert (hasOlderBrother ?x ?y))
```

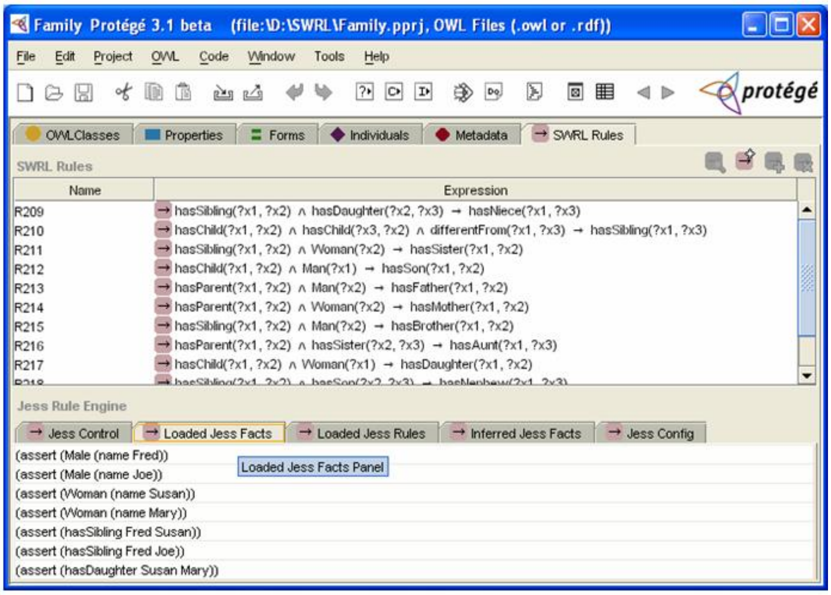
5.3 Executing Jess Rules and Updating an OWL Knowledge Base

Once the relevant OWL concepts and SWRL rules have been represented in Jess, the Jess execution engine can perform inference. As rules fire, new Jess facts are inserted into the fact base. Those facts are then used in further inference. When the inference process completes, these facts can then be transformed into OWL knowledge, a process that is the inverse of the mapping mechanism outlined in Section 5.1.

5.4 Visual Interface to the Jess Rule Engine

Interaction between the SWRL Editor and the Jess rule engine is user-driven. The user controls when OWL knowledge and SWRL rules are transferred to Jess, when inference is performed using those knowledge and rules, and when the resulting Jess facts are transferred back to Protégé-OWL as OWL knowledge.

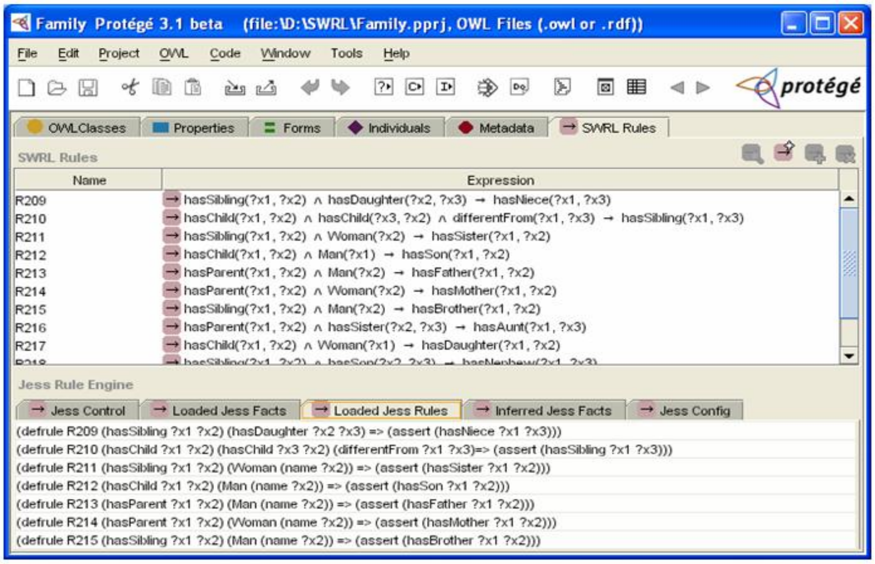
Five tabs in the Jess rule panel control this interaction: (1) *a Jess Control tab*, which is used to initiate fact and rule transfer and perform inference; (2) *a Source Facts tab*, which presents Jess facts that have been transferred from an OWL knowledge base (see Figure 4); (3) *a Rules tab*, which shows the Jess representation of SWRL rules (see Figure 5), (4) *an Asserted Facts tab* showing facts resulting from Jess inference, and (5) *a Configuration tab* that can be used to set Jess operating parameters.


Family Protégé 3.1 beta (file:\D:\SWRL\Family.pprj, OWL Files (.owl or .rdf))
File Edit Project OWL Code Window Tools Help
protégé
OWLClasses Properties Forms Individuals Metadata SWRL Rules
SWRL Rules
Jess Rule Engine
Jess Control Loaded Jess Facts Loaded Jess Rules Inferred Jess Facts Jess Config
Loaded Jess Facts Panel

Name	Expression
R209	hasSibling(?x1, ?x2) ∧ hasDaughter(?x2, ?x3) → hasNiece(?x1, ?x3)
R210	hasChild(?x1, ?x2) ∧ hasChild(?x3, ?x2) ∧ differentFrom(?x1, ?x3) → hasSibling(?x1, ?x3)
R211	hasSibling(?x1, ?x2) ∧ Woman(?x2) → hasSister(?x1, ?x2)
R212	hasChild(?x1, ?x2) ∧ Man(?x1) → hasSon(?x1, ?x2)
R213	hasParent(?x1, ?x2) ∧ Man(?x2) → hasFather(?x1, ?x2)
R214	hasParent(?x1, ?x2) ∧ Woman(?x2) → hasMother(?x1, ?x2)
R215	hasSibling(?x1, ?x2) ∧ Man(?x2) → hasBrother(?x1, ?x2)
R216	hasParent(?x1, ?x2) ∧ hasSister(?x2, ?x3) → hasAunt(?x1, ?x3)
R217	hasChild(?x1, ?x2) ∧ Woman(?x1) → hasDaughter(?x1, ?x2)

```
(assert (Male (name Fred))
(assert (Male (name Joe))
(assert (Woman (name Susan))
(assert (Woman (name Mary))
(assert (hasSibling Fred Susan))
(assert (hasSibling Fred Joe))
(assert (hasDaughter Susan Mary))
```

Fig. 4. Jess Loaded Facts Tab in the Protégé SWRL Editor. This tab shows the Jess representation of relevant OWL individuals that will be used by Jess to perform inference.


Family Protégé 3.1 beta (file:\D:\SWRL\Family.pprj, OWL Files (.owl or .rdf))
File Edit Project OWL Code Window Tools Help
protégé
OWLClasses Properties Forms Individuals Metadata SWRL Rules
SWRL Rules
Jess Rule Engine
Jess Control Loaded Jess Facts Loaded Jess Rules Inferred Jess Facts Jess Config

Name	Expression
R209	hasSibling(?x1, ?x2) ∧ hasDaughter(?x2, ?x3) → hasNiece(?x1, ?x3)
R210	hasChild(?x1, ?x2) ∧ hasChild(?x3, ?x2) ∧ differentFrom(?x1, ?x3) → hasSibling(?x1, ?x3)
R211	hasSibling(?x1, ?x2) ∧ Woman(?x2) → hasSister(?x1, ?x2)
R212	hasChild(?x1, ?x2) ∧ Man(?x1) → hasSon(?x1, ?x2)
R213	hasParent(?x1, ?x2) ∧ Man(?x2) → hasFather(?x1, ?x2)
R214	hasParent(?x1, ?x2) ∧ Woman(?x2) → hasMother(?x1, ?x2)
R215	hasSibling(?x1, ?x2) ∧ Man(?x2) → hasBrother(?x1, ?x2)
R216	hasParent(?x1, ?x2) ∧ hasSister(?x2, ?x3) → hasAunt(?x1, ?x3)
R217	hasChild(?x1, ?x2) ∧ Woman(?x1) → hasDaughter(?x1, ?x2)

```
(defrule R209 (hasSibling ?x1 ?x2) (hasDaughter ?x2 ?x3) => (assert (hasNiece ?x1 ?x3)))
(defrule R210 (hasChild ?x1 ?x2) (hasChild ?x3 ?x2) (differentFrom ?x1 ?x3)=> (assert (hasSibling ?x1 ?x3)))
(defrule R211 (hasSibling ?x1 ?x2) (Woman (name ?x2)) => (assert (hasSister ?x1 ?x2)))
(defrule R212 (hasChild ?x1 ?x2) (Man (name ?x2)) => (assert (hasSon ?x1 ?x2)))
(defrule R213 (hasParent ?x1 ?x2) (Man (name ?x2)) => (assert (hasFather ?x1 ?x2)))
(defrule R214 (hasParent ?x1 ?x2) (Woman (name ?x2)) => (assert (hasMother ?x1 ?x2)))
(defrule R215 (hasSibling ?x1 ?x2) (Man (name ?x2)) => (assert (hasBrother ?x1 ?x2)))
```

Fig. 5. Jess Rules Tab in the Protégé SWRL Editor. This tab shows the Jess representation of SWRL rules.

6 Discussion

With the exception of allowing arbitrary OWL expressions in SWRL rules, the Protégé SWRL Editor supports all language features in the current SWRL Specification. The inability to use arbitrary OWL expressions is easy to work around by creating a named OWL class that models an expression and using it in the rule.

One limitation of our inference support is that it is not integrated with an OWL classifier. Conflicts can arise in an OWL knowledge base between new information asserted by Jess and inserted into an OWL knowledge base and existing OWL restrictions. For example, a `hasUncle` property may be asserted for an individual as a result of firing a SWRL rule, but a class level restriction in OWL may forbid this property from belonging to that individual. As present, these conflicts are not detected automatically, and resolving the conflict—which is essentially between a SWRL rule and an OWL restriction and has nothing to do with Jess—is left to the user. Conflicts can be identified by running an OWL classifier on the knowledge base that has been populated by additional Jess-inferred knowledge.

To resolve these conflicts automatically, all OWL restrictions relating to classes and properties operated on by Jess would have to be captured as Jess knowledge. Jess rules would be needed to replicate the functionality of both a classifier and rule integrity checker. An additional issue is that a conflict-free execution of the classifier may also infer new knowledge that may in turn produce information that may benefit from further SWRL inference, a process that may require several iterations before no new knowledge is generated. Clearly, having this classification and inference functionality in a single module would be desirable.

The SWRL Editor has been available as part of Protégé-OWL since late 2004. User response has been very positive. Our hope now is that other investigators will use the Protégé SWRL Factory mechanism to integrate other rule engines with the SWRL Editor, eventually providing a range of rule engine choices. Jess, for example, has a very good implementation of a forward chaining rule engine but has weak backward chaining support. Consequently, a rule engine like Algernon may be more appropriate for users needing this support. Developers working within the Jena 2 environment would probably prefer the inference capabilities available in that environment. As more rule engines become available for SWRL, it should rapidly become a conduit for rule integration on the Semantic Web.

Acknowledgements

Funding for the Protégé SWRL Editor was provided by the DARPA Agent Markup Language Program. We thank Valerie Natale for her valuable editorial assistance.

References

1. W3C Workshop for Rule Languages for Interoperability: http://www.w3.org/2004/12/rules-ws/cfp
2. SWRL: http://www.daml.org/rules/proposal/
3. OWL Web Ontology Language: http://www.w3.org/TR/owl-features/

4. RuleML: http://www.ruleml.org/
5. SWRL Specification: http://www.w3.org/Submission/SWRL/
6. H. Knublauch, R. W. Fergerson, N. F. Noy, M. A. Musen. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. Third International Semantic Web Conference (2004)
7. SweetRules: http://sweetrules.projects.semwebcentral.org/
8. Jena-2: http://www.hpl.hp.com/semweb/jena.htm
9. SWRL Built-in Specification: http://www.daml.org/rules/proposal/builtins.html
10. SWRL Built-in Ontology: http://www.w3.org/2003/11/swrlb.owl
11. Jess Rule Engine: http://herzberg.ca.sandia.gov/jess/
12. Protégé SWRL Editor FAQ: http://protege.stanford.edu/plugins/owl/swrl/
13. SWRL Ontology: http://www.daml.org/rules/proposal/swrl.owl
14. Protégé SWRL Factory FAQ: http://protege.stanford.edu/plugins/owl/swrl/SWRLFactory.html
15. Jess Protégé Tab: http://www.ida.liu.se/~her/JessTab/JessTab.ppt
16. Algernon Protégé Tab: http://algernon-j.sourceforge.net/doc/algernon-protege.html
17. Java-based Rule Engines: http://www.manageability.org/blog/stuff/rule_engines/view
18. Algernon: http://www.cs.utexas.edu/users/qr/algy/
19. Kuan M. Using SWRL and OWL DL to Develop an Inference System for Course Scheduling. Masters Thesis, Chung Yuan Christian University, Taiwan, R.O.C. (2004)
20. Mei J., Bontas EP. Reasoning Paradigms for SWRL-Enabled Ontologies Protégé With Rules Workshop held at the 8th International Protégé Conference, Madrid Spain (2005)
21. Golbreich, C., Imai, A. Combining SWRL rules and OWL ontologies with Protégé OWL Plugin, Jess, and Racer. 7th International Protégé Conference, Bethesda, MD (2004)
22. Grosof B., Gandhe, M., Finin, T. SweetJess: Translating DamlRuleML to Jess. International Workshop on Rule Markup Languages for Business Rules on the Semantic Web. First International Semantic Web Conference (2002)