SWRL2COOL: Object-Oriented Transformation of SWRL in the CLIPS Production Rule Engine

Emmanouil Rigas[1], Georgios Meditskos[2], and Nick Bassiliades[2]

[1] School of Electronics and Computer Science, University of Southampton, UK
er2g11@soton.ac.uk
[2] Department of Informatics, Aristotle University of Thessaloniki, Greece
{gmeditsk,nbassili}@csd.auth.gr

Abstract. The Semantic Web Rule Language (SWRL) is a W3C member submission rule language for ontologies. It is based on a combination of the OWL DL and OWL Lite sublanguages of the OWL Web Ontology Language with the Unary/Binary Datalog RuleML sublanguages of the Rule Markup Language. In this paper we propose a transformation of SWRL rules into the object-oriented rule language of CLIPS (COOL). The purpose of this transformation is to enhance an already existing CLIPS-based OWL ontology reasoner, namely O-DEVICE, with the ability to import and execute SWRL rules during the process of building custom ontology-based production rule programs.

Keywords: SWRL, Production Rules, CLIPS, OWL.

1 Introduction

SWRL [1] is a rule language based on a combination of OWL with the Unary/Binary Datalog sublanguages of RuleML. SWRL enables Horn-like rules to be combined with an OWL knowledge base. Negation is not explicitly supported but only indirectly through OWL DL (e.g. complements). Its main purpose is to provide a formal meaning of OWL ontologies and extend OWL DL with rules.

In existing ontology reasoning implementations, although it is possible to manipulate ontologies using the SWRL rule notation, for example in KAON2 [2] and Pellet [3], it is not possible (or it is not efficient at least) to define a complete rule program over the ontology since they are not native rule engines. To this end, the use of a rule system that is able to reason over ontologies gives the opportunity to utilize directly the ontology information by building knowledge-based systems. Ontologies can be inserted into the system, and after the materialization of the semantics through the reasoning procedure, that is, the application of inference rules in order to deduce new information (entailment), user-defined rules can operate over the materialized knowledge. Based on this idea, we have developed O-DEVICE [4] on top of the CLIPS production rule engine. Its reasoning process is characterized by the transformation of ontological information into the object-oriented model of the COOL language of CLIPS and the application of inference production rules over the generated object-oriented

I. Maglogiannis, V. Plagianakos, and I. Vlahavas (Eds.): SETN 2012, LNAI 7297, pp. 49–56, 2012.

schema. In that way, custom object-oriented rule programs can be developed on top of the transformed ontological knowledge using the efficient RETE engine of CLIPS. An example of such an application is Software Antipatterns [5].

In this paper, we propose an object-oriented transformation of SWRL rules into the COOL language of CLIPS in order to be able to operate on top of the transformed object-oriented ontological model that is created by O-DEVICE. Our goal is to enable O-DEVICE to import and execute SWRL rules and therefore, easing the development of knowledge-based systems on top of ontologies in CLIPS using a well-defined and widely adopted ontology rule language.

The rest of the paper is structured as follows: Section 2 overviews the syntax of the COOL production rules in CLIPS. Section 3 describes the XSLT transformations that are applied by SWRL2COOL and present a complete example. Finally, in sections 4 and 5 we present related work and we conclude, respectively.

2 The CLIPS Production Rule Engine

CLIPS[1] is a RETE-based production rule engine written in C that was developed in 1985 by NASA's Johnson Space Center and it has undergone continual refinement and improvement ever since. One of the most interesting capabilities of CLIPS is that it integrates the production rule paradigm with the OO model, which can be defined using the COOL (CLIPS Object-Oriented Language) language. In that way, classes, attributes and objects can be matched on the production rule conditions, as well as to be altered on rules actions.

2.1 COOL Production Rules in CLIPS

A production rule in CLIPS is defined using the `defrule` construct that consists of conditions and actions separated with the symbol `=>`. The conditions can match both facts and objects, whereas the actions define the actions that should be taken upon the satisfaction of all the conditions.

Objects of user-defined classes in COOL can be pattern-matched on the left-hand side of rules using object patterns of the form

```
<object-pattern> ::= (object <attribute-constraint>*)
<attribute-constraint> ::= (is-a <constraint>) |
(name <constraint>) | (<slot-name> <constraint>*)
```

The `is-a` constraint is used for specifying class constraints and it also encompasses subclasses of the matching classes. The `name` constraint is used for specifying a specific object on which to pattern-match. Constraints are also used in slots/multislots in order to restrict certain type of values. Both in fact and object patterns, it is possible to use variables in order to be matched with certain values. A single-value variable is denoted as `?x`, whereas a multivalue variable is denoted as `$?x`. An example rule that prints all the objects of the class `Person` is presented below.

[1] `http://clipsrules.sourceforge.net/`

```
(defrule test-rule2
    (object (is-a Person) (name ?x))
=>  (printout t ?x crlf))
```

3 Transformation Procedure

SWRL2COOL is based on the XML syntax of SWRL and its main functionality is to transform SWRL rules into the COOL rule language of CLIPS. Currently, the transformed rules are used in the O-DEVICE reasoner [4] in order to allow the definition of custom production rule programs on top of ontologies.

The transformation procedure takes place in two phases (Fig. 1). In the first phase the rules which are written in SWRL are being processed by an XSLT file named `swrl2cool.xsl`. The procedure produces a file which represents the rules in an intermediate format, where SWRL constructs are transformed into CLIPS format, but properties are not yet encapsulated into object patterns. In the second phase this file is given as input to the main SWRL2COOL application (a java program), which produces the final file with the COOL rules. In parallel, the ontology classes and instances are transformed with O-DEVICE into COOL classes and objects. Finally, the two transformed pieces of knowledge (rules and ontology) are joined together to produce the object-oriented rule-based application.

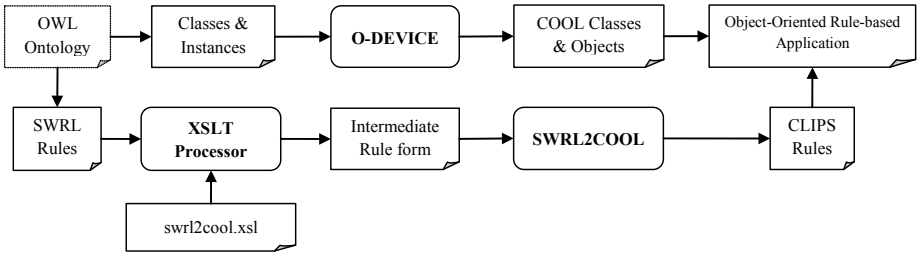

Fig. 1. The transformation procedure

3.1 XSLT Transformations

The SWRL rules have two parts: the body and the head. In most of the cases, the constructs of SWRL correspond to different constructs in COOL depending on whether they are in the head or the body of the rule. In order to handle these two cases, the XSLT transformation procedure is based on the use of modes in the `xsl:template` and `xsl:apply-templates` expressions. The modes allow an element to be processed many times, each time producing a different output. An `xsl:apply-templates` element with a mode argument, is applied to a template rule with a matching mode. The above are being used in this stylesheet as follows:

```
<xsl:template match="ruleml:imp">
<xsl:apply-templates select="ruleml:_body" mode="body" />
=>
<xsl:apply-templates select="ruleml:_head" mode="head" />
  end-of-rule
</xsl:template>
```

The `"=>"` symbol gets before the head of the rule in order to line-up with the CLIPS syntax and the phrase `"end-of-rule"` gets at the end of each rule in order to be a point of reference considering the line where one rule ends and another begins. This is used during the second phase of the transformation.

swrl:classAtom. The `classAtom` consists of a description, which is usually the name of a class and the name of a variable. When a `classAtom` is in the body of a rule then the description is matched in COOL as `object is-a "name of class"` and the variable as `name "name of variable"`. Essentially, we refer to an instance of a class that has already been set earlier. On the other hand, when a `classAtom` is in the head of a rule then the variable is matched in COOL as `make-instance <name of variable>` and the description as `of <name of class>`. The head of the rules contains actions that have to be performed in relation to the elements that exist in the body of the rule. For this reason when there is a `classAtom` in the head of a rule, then we create a new instance of a class that was set before.

```
<xsl:template match="swrlx:classAtom" mode="body" >
  object (is-a <xsl:value-of select="owlx:Class/@owlx:name" /> )
  (name <xsl:apply-templates />)
</xsl:template>
<xsl:template match="swrlx:classAtom" mode="head" >
   (make-instance <xsl:apply-templates /> of <xsl:value-of
   select="owlx:Class/@owlx:name" />)
</xsl:template>
```

swrl:individualPropertyAtom. The `individualPropertyAtom` consists of the name of a property and the names of two variables. When it is in the body of the rule, the property must be matched to an object that has been defined earlier inside the rule. Thus, the first variable is the name of the object and the second variable is the value of the property. For example:

```
<swrlx:individualPropertyAtom swrlx:property="property_name">
     <ruleml:var>var1</ruleml:var>
     <ruleml:var>var2</ruleml:var>
</swrlx:individualPropertyAtom>
```

The output of the processing begins with the qualifier `"property"`, followed by the `"object name"` of the object that the second property refers to. The qualifiers are used in the second phase of the transformation process, where COOL rules take their final format.

```
<xsl:template match="swrlx:individualPropertyAtom" mode="body">
  property
    (object (name<xsl:apply-templates select="ruleml:var[1]"/>)
    (<xsl:value-of select="@swrlx:property"/>
    <xsl:apply-templates  select="ruleml:var[2]"/> )
</xsl:template>
```

When an `individualPropertyAtom` is in the head of the rule, then the property must be matched to an object that has been defined earlier and change its value.

```
<xsl:template match="swrlx:individualPropertyAtom" mode="head">
  (slot-insert$ <xsl:apply-templates  select="ruleml:var[1]"/>
  <xsl:value-of select="@swrlx:property" />
  1 <xsl:apply-templates  select="ruleml:var[2]"/>)
</xsl:template>
```

The above stylesheet makes use of the `(slot-insert$ <ins> <p> <i> <v>)` function of CLIPS that inserts the value `v` in the slot `p` of the object `ins` at position `i`.

swrlx:datavaluedPropertyAtom. The `datavaluedPropertyAtom` consists of the name of the property, a name of a variable and a name of a constant. The `datavaluedPropertyAtoms` are being processed like the `individualPropertyAtoms` with one difference. The property takes as a constant value and not a variable.

```
<xsl:template match="swrlx:datavaluedPropertyAtom" mode="body">
  property
  (object (name<xsl:apply-templates select="ruleml:var[1]"/>)
    (<xsl:value-of select="@swrlx:property" />
     <xsl:value-of select="owlx:DataValue" /> )
</xsl:template>
<xsl:template match="swrlx:datavaluedPropertyAtom" mode="head">
  (slot-insert$ <xsl:apply-templates  select="ruleml:var[1]"/>
   <xsl:value-of select="@swrlx:property" />
   1 <xsl:value-of select="owlx:DataValue" />)
</xsl:template>
```

swrlx:builtInAtom. SWRL supports a large amount of built in functions, for comparisons, mathematical expressions, Booleans, strings, date, time and lists. Currently, SWRL2COOL handles only built in functions for comparisons and mathematical expressions and it is assumed that the body of the rule contains functions for comparisons and the head functions for mathematical expressions. The functions for comparisons are treated with the `test` construct of CLIPS and the functions for mathematical expressions are treated with the `bind` CLIPS function, to compute a new value that must be added or updated.

```
<xsl:template match="swrlx:builtinAtom" mode="body">
test ( <xsl:value-of select="@swrlx:builtin" />
  <xsl:apply-templates  select="ruleml:var[1] "/>
  <xsl:apply-templates select="ruleml:var[2]|owlx:DataValue"/>)
</xsl:template>
<xsl:template match="swrlx:builtinAtom" mode="head">
  bind <xsl:apply-templates select="ruleml:var[1]"/>
( <xsl:value-of select="@swrlx:builtin" />
  <xsl:apply-templates  select="ruleml:var[2]"/>
<xsl:apply-templates select="ruleml:var[3]|owlx:DataValue[1]"/>)
</xsl:template>
```

3.2 Transformation Example

We now present a transformation example of a complete SWRL rule (below in presentation syntax) and the results of all transformation phases.

```
Car(?c), tank-capacity(?c,?tc), fuel-consumption(?c,?fc)
  -> swrlb:divide(?a,?tc,?fc), autonomy(?c,?a)
```

SWRL Rule - XML syntax:

```
<ruleml:imp>
<ruleml:_body>
  <swrlx:classAtom>
    <owlx:Class owlx:name="car"/>   <ruleml:var>c</ruleml:var>
  </swrlx:classAtom>
  <swrlx:individualPropertyAtom swrlx:property="tank-capacity">
    <ruleml:var>c</ruleml:var>      <ruleml:var>tc</ruleml:var>
  </swrlx:individualPropertyAtom>
  <swrlx:individualPropertyAtom swrlx:property="fuel-consumption">
    <ruleml:var>c</ruleml:var>      <ruleml:var>fc</ruleml:var>
  </swrlx:individualPropertyAtom>
</ruleml:_body>
<ruleml:_head>
  <swrlx:builtinAtom swrlx:builtin="divide">
    <ruleml:var>a</ruleml:var>
      <ruleml:var>tc</ruleml:var> <ruleml:var>fc</ruleml:var>
    </swrlx:builtinAtom>
    <swrlx:individualPropertyAtom swrlx:property="autonomy">
      <ruleml:var>c</ruleml:var> <ruleml:var>a</ruleml:var>
    </swrlx:individualPropertyAtom>
</ruleml:_head>
</ruleml:imp>
```

Intermediate rule format:

```
object (is-a car) (name ?c)
property (object (name ?c) (tank-capacity ?tc)
property (object (name ?c) (fuel-consumption ?fc)
=>
bind ?a (divide ?tc ?fc)
(slot-insert$ ?c autonomy 1 ?a)
end-of-rule
```

Final CLIPS rule:

```
(defrule r2
  (object (is-a car ) (name ?c)
    (tank-capacity $? ?tc $?) (fuel-consumption $? ?fc $?))
=>
  (bind ?a (/ ?tc ?fc))
  (slot-insert$ ?c autonomy 1 ?a))
```

4 Related Work

A lot of approaches have been proposed towards the development of frameworks able to execute SWRL rules on top of OWL ontologies in native rule engines. Many of them use the Jess[2] rule engine and follow a fact-based approach, where both rules and OWL ontologies are mapped on fact-based Jess constructs, in contrast to our approach that follows an object-oriented implementation using the COOL language of CLIPS. To the best of our knowledge, this is the first effort that enables the CLIPS rule engine to import and execute SWRL rules in an object-oriented manner. In the following, we briefly present existing SWRL transformation approaches.

SWRLJessTab [6] is a plugin in Protege that allows the execution of SWRL rules using the Jess rule engine. The interaction between OWL 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 Protege as OWL knowledge. SWRLJessTab actually maps SWRL rules on fact-based Jess rules that match facts, in contrast to our approach that SWRL rules are transformed into object-oriented rules and match objects in CLIPS.

In [7] the authors reuse the SWRLJessTab along with SweetRules[3] and other standard tools [8] in order to transform an extended version of SWRL suitable in teaching scenarios into Jess fact-based rules. This is also a fact-based approach where the knowledge and the rules are represented in terms of Jess facts.

A similar approach to SWRLJessTab is the OWL2Jess [9] tool that enables the transformation of OWL ontologies to Jess and thus enables OWL models to

[2] http://herzberg.ca.sandia.gov/jess

[3] http://sweetrules.projects.semwebcentral.org/

be extended by means of rules. Facts are derived from an initial OWL file by one XSLT stylesheet, while the RDF(S) and OWL Semantics are pre-defined as Jess rules.

5 Conclusions

In this paper, we presented an object-oriented transformation of SWRL rules into the COOL language of CLIPS. The approach is based on XSLT transformations over the XML syntax of SWRL. In that way, we enable the CLIPS production rule engine to import and execute SWRL rules over object-oriented COOL models.

Currently, we use our tool in the O-DEVICE reasoner that processes and maps OWL ontologies on the COOL model of CLIPS. In that way, SWRL rules can be combined with classes and instances from an OWL ontology that has been transformed into classes and instances in the COOL model using the O-DEVICE. In the future, we plan to add support for more built-in SWRL constructs and to implement SWRL2COOL as a Protege plugin.

References

1. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A semantic web rule language combining OWL and RuleML. W3C member submission, World Wide Web Consortium
2. Motik, B.: KAON2 - scalable reasoning over ontologies with large data sets. ERCIM News 2008(72) (2008)
3. Sirina, E., Parsia, B., Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. Journal of Web Semantics 5(2), 51–53 (2007)
4. Meditskos, G., Bassiliades, N.: A rule-based object-oriented OWL reasoner. IEEE Transactions on Knowledge and Data Engineering 20(3), 397–410 (2008)
5. Settas, D., Meditskos, G., Stamelos, I., Bassiliades, N.: SPARSE: A symptom-based antipattern retrieval knowledge-based system using semantic web technologies. Expert Syst. Appl. 38(6), 7633–7646 (2011)
6. Golbreich, C., Imai, A.: Combining SWRL rules and OWL ontologies with Protege OWL plugin, Jess, and Racer. In: 7th International Protege Conference (2004)
7. Wang, E., Kim, Y.S.: Using SWRL for ITS through keyword extensions and rewrite meta-rules. In: 5th Int'l Workshop on Ontologies and Semantic Web for E-Learning, SWEL@AIED 2007 (2007)
8. Wang, E., Kim, Y.S.: A teaching strategies engine using translation from SWRL to Jess. In: Intelligent Tutoring Systems 2006, pp. 51–60 (2006)
9. Mei, J., Bontas, E.P., Lin, Z.: OWL2Jess: A Transformational Implementation of the OWL Semantics. In: Chen, G., Pan, Y., Guo, M., Lu, J. (eds.) ISPA-WS 2005. LNCS, vol. 3759, pp. 599–608. Springer, Heidelberg (2005)