

# SWRLp: An XML-based SWRL Presentation Syntax

Christopher J. Matheus

Versatile Information Systems, Inc.  
Framingham, Massachusetts U.S.A.  
cmatheus@vistology.com  
<http://www.vistology.com>

**Abstract.** RuleML and SWRL are closely related rule markup languages that are growing in popularity particularly within the Semantic Web community. Both are based on XML and as such are more amenable to computer processing than human generation and interpretation. Even so it is possible to read and write RuleML and SWRL rules manually using a text editor. Due to certain constraints imposed within SWRL, however, reading and writing SWRL rules is significantly more difficult. This paper introduces an alternative XML-based presentation syntax, SWRLp, that lends itself more readily to the reading and writing of SWRL rules. SWRLp affords a simplification of the structural representation of rules by relying on a few basic conventions. A simple example of the improvement in readability of SWRLp rules is presented along with details about its design. XSLT transformation scripts that have been developed for translating to and from SWRLp and SWRL, RuleML, pseudo-Prolog and Jess are described.

## 1 Introduction

RuleML has been around for a number of years and has enjoyed a growing popularity as a standardized rule representation language [1]. It was explicitly designed to make it relatively easy to translate rules to and from the various rule representations used by popular reasoning engines (e.g., Jess, XSB, Clips, cwm), which is one reason why the language is based in XML. While XML is ideal for machine processing/translation it can be challenging for humans to use depending upon the design of the language (i.e., the schema) and the experience of the user. While one can argue that RuleML is more difficult to manually process than some other rule representation languages, the designers of the language have achieved a syntax that is manageable enough for a knowledgeable developer with experience with rule languages and XML to readily edit rules with a text editor.

SWRL (Semantic Web Rule Language) [2] was more recently introduced as a way to integrate rules with OWL DL ontologies [3]; this integration has been an issue of some debate as earlier efforts attempted to meld DAML (DARPA Agent Markup Language, a

precursor to OWL) [4] or OWL and RuleML [5]. Many of the constructs of RuleML were incorporated into SWRL and in fact elements from the RuleML namespace are used extensively throughout the language. Whereas RuleML is designed to be all encompassing and quite flexible in the types of rules it can be used to implement, SWRL was purposely constrained in ways that permit it to be used in the context OWL DL and thereby inherit important semantic characteristics that make automated reasoning more tractable. One of the characteristics of OWL inherited by SWRL is the restriction to unary- and binary-termed predicates (excluding the built-ins). While this constraint does not limit the logical statements that can be represented by SWRL it does add to the complexity and verbosity of the rules, as is demonstrated below. Additional aspects of the SWRL syntax, such as the manner in which properties are specified, contribute to the difficulty of reading and editing rules represented using the SWRL concrete XML syntax, referred to as SWRLx from here on.

It can be fairly argued that use of XML adds to the problem of human processing of SWRL rules. Some have suggested that what is needed is a non-XML based syntax for SWRL specifically designed for human consumption. If the only use of a set of rules is for presenting information to humans, such an approach would be sufficient. Most of the time, however, when we write rules it is with the intent of ultimately processing them with an automated inference engine (or engines). For this reason it is prudent to create rules using an XML-based syntax that can be easily translated into other XML as well as non-XML representations. Until SWRL is mature enough to have truly useful graphical editors that abstract out the underlying complexities in the syntax, we are in a situation where most of the editing of rules will be achieved with text-based or XML-based editors; even then there will always be some who will at least periodically wish to edit parts of rules by hand. This being the case, this paper introduces a simplified presentation syntax for SWRL that is based in XML. A presentation syntax alone, however, is not very useful unless there is a way to translate rules represented in the syntax into forms usable by inference engines or other reasoning tools. Consequently, several XSLT scripts have been written to translate to and from the presentation syntax and SWRLX, RuleML, Jess and a simplified Prolog-like syntax intended for communicating the content of the rules between humans, herein referred to as pseudo-Prolog.

The paper begins by presenting a simple example of a rule from the domain of supply logistics and then demonstrates the increasing complexity encountered as one moves from pseudo-Prolog syntax to RuleML syntax to SWRLx syntax. The next section introduces the proposed presentation syntax, SWRLp, contrasts it to the SWRLx syntax and discusses the set of conventions that are needed to permit its simplified structure. A set of XSLT scripts written to translate between the SWRLp syntax and various other representations are then described.

## 2 Representations of a Simple Rule

To illustrate the added complexity introduced by moving from RuleML to SWRLx a simple rule from the supply logistics domain is presented. To clearly communicate the content of the rule it is first shown using a simplified, easily read representation referred to in this paper as pseudo-Prolog syntax.

### 2.1 Pseudo-Prolog Syntax

Assume we have a set of rules defining knowledge pertinent to the domain of supply logistics. Included among these will be rules that capture the basic notion of what it means for a consumer to be “in supply of” a given resource at a given point in time. Let us define the predicate representing this notion to be `inSupplyOfAt (_, _, _)` with the three ordered arguments representing the `consumer`, the `resource` and the `time`. Assume we wish to state that a consumer is in supply of a resource at a particular time if it has sufficient reserves of that resource to meet or exceed its expected consumption of said resource. This bit of knowledge is captured by the following pseudo-Prolog code:

```
inSupplyOfAt (Consumer, Resource, Time)
:- reserve (Consumer, Resource, Time, Reserve)
   consumption (Consumer, Resource, Consumption)
   greaterThanOrEqual (Reserve, Consumption) .
```

The first line of the pseudo-code represents the head of the rule (i.e., the consequence) and the subsequent lines capture the rule’s body (i.e., the antecedent). The `reserve` predicate binds the quantity of `reserve` of a particular `resource` held by the `consumer` at the specified `time`. The `consumption` predicate binds the quantity of consumption of the `resource` by the `consumer` for a unit interval of time. The `greaterThanOrEqual` predicate returns true if the amount in `reserve` is greater than the amount of `consumption`. Granted this rule may be a bit simplistic for an actual supply logistics system; however, it will be sufficient for the illustrative purposes of this paper.

### 2.2 RuleML Syntax

Using a straightforward mechanical process it is quite simple to translate the pseudo-Prolog representation of our sample rule into the following RuleML representation<sup>1</sup>:

---

<sup>1</sup> The predicates and classes used in the RuleML and SWRL rules actually come from specific OWL ontologies and therefore should be qualified with namespace prefixes.

```

<imp>
  <_head>
    <atom>
      <_opr><rel>inSupplyOfAt</rel></_opr>
      <var>?consumer</var>
      <var>?resource</var>
      <var>?time</var>
    </atom>
  </_head>
  <_body>
    <and>
      <atom>
        <_opr><rel>reserve</rel></_opr>
        <var>?consumer</var>
        <var>?resource</var>
        <var>?time</var>
      </atom>
      <atom>
        <_opr><rel>consumption</rel></_opr>
        <var>?consumer</_var>
        <var>?resource</_var>
        <var>?consumption</_var>
      </atom>
      <atom>
        <_opr><rel>greaterThanOrEqual</rel></_opr>
        <var>?reserve</var>
        <_var>?consumption</var>
      </_atom>
    </_and>
  </_body>
</_imp>

```

The `imp` tag contains the specification of a single rule comprised of a `head` and a `body` that are in turn made up of **atoms** consisting (in this case) of relational operands (`<_opr><rel>`) with variables (`<var>`) as arguments. Clearly, this representation is more verbose and not as easy to take in at a glance, as is the pseudo-Prolog code. Though more cumbersome, the basic logical structure of the rule remains intact and apart from the

---

These prefixes were removed from the rules to simplify the presentation of the rules. Since this simplification was applied to both RuleML and SWRL rules it does not affect the comparisons of their relative complexities.

verbose syntax the rule is readily comprehensible. While it would be somewhat tedious to write and maintain this rule with a text processor, it is not a terribly onerous task for someone versed in XML and RuleML.

### 2.3 SWRL Syntax

To represent this same rule in SWRLx requires some additional transformations, both syntactic and structural. First, the three-termed predicates in the rule must be converted into binary predicates as SWRL's conformance to OWL prohibits the use of user defined higher-order predicates. It is always possible to convert rules with higher-order predicates into rules with only binary predicates through the introduction of reified variables that carry along the additional terms as property values (described below). Unfortunately this process changes the basic structure of the rule and adds considerably to its length and complexity, as evidenced in the rule's SWRLx representation shown below.

Second, the SWRLx syntax distinguishes between predicates representing OWL class individuals, those representing OWL object properties and those representing OWL datatype properties. This requirement means that different elements must be used depending upon the nature of the particular predicate being used (i.e., `<swrlx:classAtom>` for class individuals, `<swrlx:individualPropertyAtom>` for object properties, and `<swrlx:datavaluePropertyValue>` for datatype properties). Having this additional semantic information embedded in the element's name might prove useful to the reader unfamiliar with the ontologies used in the rules but they also force the reader to spend additional cognitive effort in parsing the rules.

Third, SWRLx intermixes elements from the `ruleml`, `swrlx`, `swrlb` and `owlx` namespace. One consequence of this that namespace prefixes must be used on all element names. In RuleML there is only a single namespace, which means namespace prefixes can be omitted from element names rendering them a bit more readable and concise.

Collectively, these three transformation requirements render the relatively simple original rule into the following complex and much more difficult to read representation:

```
<ruleml:imp>
  <ruleml:_head>
    <swrlx:classAtom>
      <owlx:Class owlx:name="InSupplyOfAt" />
      <ruleml:var>?reifiedRelation</ruleml:var>
    </swrlx:classAtom>
    <swrlx:individualPropertyAtom swrlx:property="consumer">
      <ruleml:var>?reifiedRelation</ruleml:var>
      <ruleml:var>?consumer</ruleml:var>
    </swrlx:individualPropertyAtom>
    <swrlx:individualPropertyAtom swrlx:property="resource">
      <ruleml:var>?reifiedRelation</ruleml:var>
```

```

        <ruleml:var>?resource</ruleml:var>
    </swrlx:individualPropertyAtom>
    <swrlx:datavaluedPropertyAtom swrlx:property="dateTime">
        <ruleml:var>?reifiedRelation</ruleml:var>
        <ruleml:var>?time</ruleml:var>
    </swrlx:datavaluedPropertyAtom>
</ruleml:_head>
<ruleml:_body>
    <swrlx:individualPropertyAtom swrlx:property="reserve">
        <ruleml:var>?consumer</ruleml:var>
        <ruleml:var>?reserve</ruleml:var>
    </swrlx:individualPropertyAtom>
    <swrlx:individualPropertyAtom swrlx:property="resource">
        <ruleml:var>?reserve</ruleml:var>
        <ruleml:var>?resource</ruleml:var>
    </swrlx:individualPropertyAtom>
    <swrlx:datavaluedPropertyAtom swrlx:property="dateTime">
        <ruleml:var>?reserve</ruleml:var>
        <ruleml:var>?time</ruleml:var>
    </swrlx:datavaluedPropertyAtom>
    <swrlx:individualPropertyAtom swrlx:property="quantity">
        <ruleml:var>?reserve</ruleml:var>
        <ruleml:var>?quantity</ruleml:var>
    </swrlx:individualPropertyAtom>
    <swrlx:datavaluedPropertyAtom swrlx:property="amount">
        <ruleml:var>?quantity</ruleml:var>
        <ruleml:var>?amount</ruleml:var>
    </swrlx:datavaluedPropertyAtom>
    <swrlx:individualPropertyAtom swrlx:property="consumption">
        <ruleml:var>?consumer</ruleml:var>
        <ruleml:var>?consumption</ruleml:var>
    </swrlx:individualPropertyAtom>
    <swrlx:individualPropertyAtom swrlx:property="consumptionType">
        <ruleml:var>?consumption</ruleml:var>
        <ruleml:var>?resource</ruleml:var>
    </swrlx:individualPropertyAtom>
    <swrlx:datavaluedPropertyAtom swrlx:property="consumptionRate">
        <ruleml:var>?consumption</ruleml:var>
        <ruleml:var>?rate</ruleml:var>
    </swrlx:datavaluedPropertyAtom>
    <swrlx:builtinAtom swrlx:builtin="&swrlb;greaterThanOrEqual">
        <ruleml:var>?amount</ruleml:var>
        <ruleml:var>?rate</ruleml:var>
        <ruleml:var/>
    </swrlx:builtinAtom>
</ruleml:_body>
</ruleml:imp>

```

Whereas we have a single predicate in the head of the pseudo-Prolog and RuleML rules, the SWRLx rule has four predicates in the head. Similarly, the pseudo-Prolog and RuleML rule bodies contained just three predicates each while the SWRLx rule body consists of nine. This expansion is the result of having to convert the three-term predicates in the original rule into sets of binary predicates. To see how this took place consider the predicate in the head of the rule. To capture the three terms a new class was created in the ontology (not shown) called `InSupplyOfAt` and three properties were associated with it; each corresponding to one of the terms `consumer`, `resource` and `time`. An instance of this class is created in the head in the first predicate and the three properties are then associated with variables that become bound when the predicates in the body are satisfied.<sup>2</sup> Note also that this same reified-object technique is employed in the body although in this case the objects are not create but are expected to be found in the working memory of the inference engine.

If we simply consider the number of lines required for the SWRLx representation we see an expansion factor of around 15 times relative to the pseudo-Prolog code and approximately two times over the RuleML code. Add to this the additional cognitive processing required to parse the various predicate types and namespace elements and it is clear that SWRLx is not a language anyone would choose to use for a rule base that is to be manually maintained.

### 3 SWRLp

We do not yet have graphical editors for SWRL to hide the complexities inherent in the SWRLx syntax and since the language is still in the early proposal stage it could be some time before we have anything approximating a stable GUI-based editing tool. Even when we do there will be situations where hand editing raw SWRL rules will be desirable. There thus seems to be a useful role for an alternative representation that makes text-editing SWRL rules more manageable. In fact it was through the painful suffering over writing some very simple SWRL rules that this author was driven to find an easier way. The result is the SWRL presentation syntax that is the focus of this section. For convenience we will refer to this syntax as SWRLp.

The best way to introduce SWRLp is with consideration of the following code that captures our sample supply logistics rule:

```
<rule>
  <head>
```

---

<sup>2</sup> Technically this rule violates the current SWRL proposal because it specifies an existential quantifier in the head. This “feature” of SWRL is a matter of contention that the author hopes will be resolved soon. If it is not it is possible to move the creation of the reified object into the body.

```

    <InSupplyOfAt      indv="?reifiedRelation"          />
    <consumer          subj="?reifiedRelation"  objt="?consumer"/>
    <resource          subj="?reifiedRelation"  objt="?resource"/>
    <dateTime          subj="?reifiedRelation"  data="?time"/>
</head>
<body>
  <reserve           subj="?consumer"          objt="?reserve"/>
  <resource          subj="?reserve"          objt="?resource"/>
  <dateTime          subj="?reserve"          data="?time"/>
  <quantity          subj="?reserve"          objt="?quantity"/>
  <amount            subj="?quantity"         data="?amount"/>
  <consumption       subj="?consumer"         objt="?consumption"/>
  <consumptionType  subj="?consumption"       objt="?resource"/>
  <consumptionRate  subj="?consumption"       data="?rate"/>
  <greaterThanOrEqual arg1="?amount"          arg2="?rate"/>
</body>
</rule>

```

SWRLp is XML-based but all of the pertinent information about the predicates and terms has been moved into element names and attribute values.; except for the rule, head and body tags, all elements are empty. The advantage this affords is that it flattens the structure of the code and permits each predicate to fit on a single line making it significant easier to parse and permitting a grasp of the overall structure in a single glance. This format very closely approximates the code that would result if pseudo-Prolog syntax were used to represent the binary-predicate-restricted version of the rules (i.e., what the pseudo-Prolog rules would look like if higher-order predicates were transformed into binary predicates). This in fact was the intent: to make the rules as concise and readable as the pseudo-Prolog syntax to the extent permissible under the language constraints of XML.

The selection of the names for the attributes was also driven by readability concerns. Because the values of attributes are easier to read if they can be easily aligned it was decided to use a constant number of letters in the attribute names. Since there can be multiple arguments to built-ins it made sense to use arg# (where # is an integer) as the builtin attribute names. This set the character count for names to four and the rest of the attribute names were selected based on that constraint.

Aside from the desire to make the rules highly readable it was also critical that they capture all information required to be able to generate the corresponding SWRLx code. The major technique used to achieve this was the use of different attribute names to distinguish the various types of SWRLx predicates. If an element has subj (subject) and objt (object) attributes it represents a `iswrlx:ndividualPropertyAtom`; if it has subj and data attributes it is a `swrlx:datavaluePropertyAtom`; if it has an indv attribute it is a `swrlx:ClassAtom`.; if it has one or more attribute beginning with “arg” it is a `swrlx:builtinAtom`. Another simplifying technique assumes the convention that all variables begin with a question mark. With this assumption it is possible to encode all terms as non-structured values thereby permitting them to be used as attribute values.

To illustrate how these techniques are employed let us walk through their application to the first predicate. The element's name is "InSupplyOfAt" and it has a single `indv` attribute. Because of this `indv` attribute we know it is a `swrlx:ClassAtom` definition and since the value of the `indv` attribute begins with a question mark we know it is a variable. These pieces of information are sufficient to be able to generate the following SWRLx coded needed to represent the predicate:

```
<swrlx:classAtom>
  <owlx:Class owlx:name="InSupplyOfAt"/>
  <ruleml:var>?reifiedRelation</ruleml:var>
</swrlx:classAtom>
```

Each of the other predicates are handled in a similar fashion, as can be seen if you compare the SWRLp code with the SWRLx code from above.

### 3.1 Caveats

There are a few caveats about what SWRLp can and cannot accomplish. It was already mentioned that SWRLp assumes the convention of prefixing all variables with a question mark. If the inference engine or reasoning tool that will receive the SWRLx rules prohibits the use of question marks at the beginning of variables, then an alternative approach will be needed. This is not a significant shortcoming and in fact some applications actually require the use of a question to demark variables (e.g., Jess).

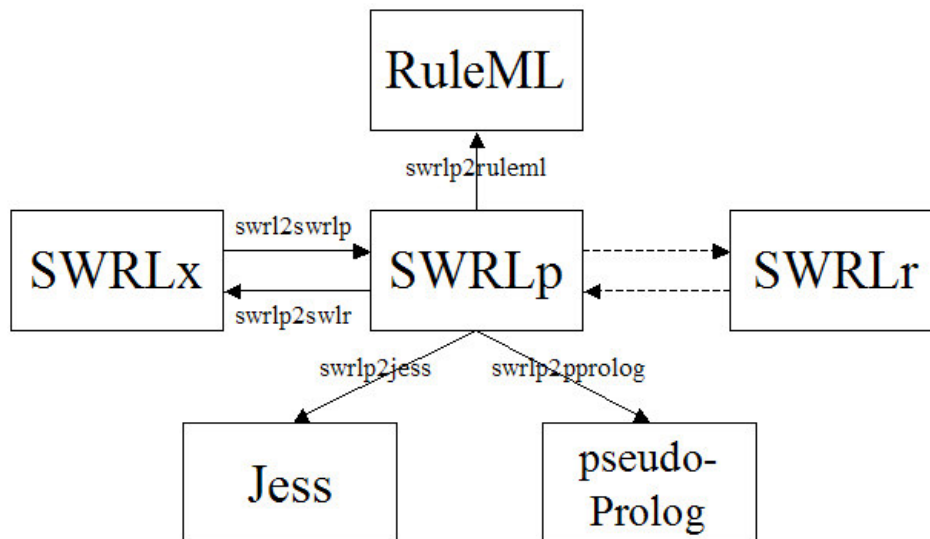
A more serious limitation for some applications may be the current prohibition on the use of arbitrary OWL code within rules. SWRL itself permits the inclusion of OWL DL constructs within the definition of rules. This feature affords the convenience of being able to construct, for example, classes that are specific to particular rules but otherwise have no significant meaning outside of the rule set (in other words, they do not belong in the domain ontology). The current implementation of SWRLp assumes that any OWL-specific constructs are defined in an ontology/annotation files that are separate from the SWRLp file. This assumption in no way constrains the expressiveness of the rules that can be constructed using SWRLp but it may represent a slight inconvenience over the use of SWRLx in certain situation. It is believed, however, that this minor limitation is far outweighed by the benefits to rule editing provided by SWRLp.

The biggest challenge in moving from RuleML to SWRL is the need to work with unary and binary predicates. As shown above we can convert higher-order predicates into binary predicates by reifying an object that then is used to carry along the values of the higher order terms as property values, however, this process changes the structure of the rules and significantly increases their length. While the SWRLp syntax helps by making these complex rules easier to read it really does not address the cause of the added complexity. What we would really like to have is a way to be able to use higher order predicates in our rules and have them be automatically converted into binary predicates.

It is probably not possible to accomplish this in a fully automated fashion under the current assumptions of SWRLp (since the reified classes need to be part of the underlying ontology) but there are likely some things that could be automated.

## 4 XSLT Translation

SWRLp by itself would not be very useful if it were not possible to translate it into forms that can be directly used by other tools, reasoning systems or humans. This section discusses several XSLT translation scripts that have been implemented to make such translations possible. Figure 1 depicts the current set of available translations along with the names of their corresponding scripts. The actual XSLT scripts will be made available online at <http://www.vistology.com/swrlp>.



**Figure 1** Translations between various rule representation syntaxes. Solid labelled arrows represent translations that have been implemented as XSLT scripts; dotted lines represent possible future scripts.

Since RuleML, SWRLx, and SWRLr (the RDF syntax for SWRL) are all based in XML it is in theory possible to write XSLT scripts to convert to and from these languages and SWRLp. As was implied above, RuleML does not encode enough detail about the predicates used in its rules to differentiate between the different predicates employed by

SWRL. For this reason the figure does not show a transformation arrow from RuleML to SWRLp. It would be possible to write an XSLT script that pulled in the ontologies used to define the predicates in a set of RuleML rules and thereby have the information necessary to construct appropriate SWRLp rules from the RuleML rules. It is not clear at this stage, however, that there is sufficient need for such a translation to warrant the time (though not overly significant) required to implement such a script. For similar reasons no effort has been made at this stage to implement the scripts to convert to and from SWRLr.

#### 4.1 Sample Rules in pseudo-Prolog

The `swrlp2preso.xsl` script converts from SWRLp to pseudo-Prolog syntax. The following is the output it produced from the SWRLp representation of the sample rule. Note that the convention in Prolog is for variables to begin with an upper case letters rather than a question mark. It should also be noted that in conventional Prolog the head of a rule is limited to a single predicate whereas this rule contains four predicates, which is one reason why the term “pseudo-Prolog” is used in this paper.

```
sl:InSupplyOfAt (ReifiedRelation)
sl:consumer (ReifiedRelation, Consumer)
sl:supplyType (ReifiedRelation, Resource)
sl:dateTime (ReifiedRelation, Time)
:-
sl:reserve (Consumer, Reserve)
sl:supplyType (Reserve, Resource)
sl:dateTime (Reserve, Rime)
sl:quantity (Reserve, Quantity)
sl:amount (Quantity, Amount)
sl:consumption (Consumer, Consumption)
sl:consumptionType (Consumption, Resource)
sl:consumptionRate (Consumption, Rate)
swrlb:greaterThanOrEqual (Amount, Rate, ) .
```

#### 4.2 Sample Rule in Jess

In our research with rules at Versatile Information Systems we use Jess as our primary inferencing engine. Consequently we need to convert SWRLp rules into Jess syntax. The XSLT script to perform this operation is quite similar to the `swrlp2prolog` script. The two major differences involve the implementation of SWRL built-ins and the need to generate unique symbols for the reified objects that are created in rule heads. For built-ins we implement a call to a Java object that implements the subset of `swrlx:builtin` methods

required for our needs. To create the reified objects we simply perform a gensym (a builtin Jess function) to generate a unique reference. The following is the code for the sample rule as generated by swlp2jess.xsl (note that in Jess the body of the rule appears before the head):

```
(defrule inSupplyDuring
  (pv (p "timeIncrement") (s "") (o ?timeIncrement))
  (call ?builtinObject "addDayTimeDurationToDateTime"
    ?timeStart2 ?timeStart ?timeIncrement)
  (pv (p "rdf:type") (s ?reifiedRelation2) (o "InSupplyOfAt"))
  (pv (p "consumer") (s ?reifiedRelation2) (o ?consumer))
  (pv (p "resource") (s ?reifiedRelation2) (o ?resource))
  (pv (p "dateTime") (s ?reifiedRelation2) (o ?timeStart))
  (pv (p "rdf:type") (s ?reifiedRelation3) (o "InSupplyOfDuring"))
  (pv (p "consumer") (s ?reifiedRelation3) (o ?consumer))
  (pv (p "resource") (s ?reifiedRelation3) (o ?resource))
  (pv (p "startTime") (s ?reifiedRelation3) (o ?timeStart2))
  (pv (p "endTime") (s ?reifiedRelation3) (o ?timeEnd))
=>
  (bind ?reifiedRelation (gensym*))
  (assert (pv (p "rdf:type") (s ?reifiedRelation) (o "InSupplyOfDuring")))
  (assert (pv (p "consumer") (s ?reifiedRelation) (o ?consumer)))
  (assert (pv (p "resource") (s ?reifiedRelation) (o ?resource)))
  (assert (pv (p "startTime") (s ?reifiedRelation) (o ?timeStart)))
  (assert (pv (p "endTime") (s ?reifiedRelation) (o ?timeEnd))))
```

## 5 Conclusion

Having suffered sufficiently from an initial experience writing SWRL rules with a text editor, the author was driven to develop an alternative representation for SWRL that would be much easier to read and edit. The result is SWRLp, simplified presentation syntax for SWRL that retains the language's basis in XML. The actual work involved in the creation of SWRLp was the development of XSLT scripts that make it possible to convert to and from other representations including SWRLx, Jess and a pseudo-Prolog representation intended for human communication. This paper was written to explain the motivation for this effort, provide an example of its use, offer some specifics concerning the syntax and describe the XSLT transformation scripts that have been developed thus far.

---

## References

- 1 RuleML: <http://www.ruleml.org>.
- 2 DARPA DAML Program, SWRL: A Semantic Web Rule Language Combining OWL and RuleML, April 2004 <http://www.daml.org/rules/proposal/>.
- 3 Web Ontology Language, OWL. <http://www.w3c.org/2001/sw/WebOnt/>.
- 4 DARPA Agent Markup Language. <http://www.daml.org>.
- 5 C. Matheus, K. Baclawski, M. Kokar, and J. Letkowski, Constructing RuleML-Based Domain Theories on top of OWL Ontologies. Proc. of the RuleML Workshop at the International Semantic Web Conference, Sanibel Island, Florida, October 2003.