BaseVISor: A Triples-Based Inference Engine Outfitted to Process RuleML and R-Entailment Rules

Christopher J. Matheus
Versatile Information Systems
Framingham, MA USA
cmatheus@vistology.com

Ken Baclawski
Northeastern University
Boston, MA USA
kenb@ccs.neu.edu

Mieczyslaw M. Kokar
Northeastern University
Boston, MA USA
mkokar@ece.neu.edu

Abstract

BaseVISor is a forward-chaining inference engine based on a Rete network optimized for the processing of RDF triples. A clause within the body and head of a rule either represents an RDF triple or invokes a procedural attachment (either built-in or user defined). This paper describes how BaseVISor has been outfitted to process RuleML and R-Entailment rules. In the case of RuleML, n-ary predicates are automatically translated into binary predicates and reified statements that encapsulate the n-ary predicates' arguments. For R-Entailment rules, the appropriate R-Entailment axioms, axiomatic triples and consistency rules are automatically imported into the engine and then used to derive all triples entailed by any set of triples asserted into the fact base. Operation of the system is illustrated using sample rule sets for both RuleML and R-Entailment and instructions are provided on how to obtain the BaseVISor beta release and process the examples.

1. Introduction

The Web Ontology Language OWL [1] has become popular for formally capturing the classes and simple properties relevant to a particular domain of interest, usually with the intent of automatically reasoning about instances from the domain. An appealing characteristic of OWL DL is its formal semantics grounded in description logic for which positive decidability and complexity results have been proven [2]. To make use of OWL DL's semantics one needs to employ a reasoning system capable of using the axioms of the language; well-known examples of such systems include the tableaux reasoners FaCT [3], Pellet [4], RACER[5] and Cerebra [6].

A common criticism of OWL is its expressive limitations with regards to constructing composite properties (i.e., properties composed of other properties, related to the notion of joins in relational databases) with the prototypical example being that of "uncle" which can be composed from the properties "parent" and "brother" [7]. This limitation has forced us and many others to find ways to extend OWL through the use of rule languages [8][9][10]. The approach taken with SWRL is an augmentation of OWL DL with Horn-style rules [11]. Unfortunately SWRL is known to be undecidable and there are few reasoning engines that support it, with Hoolet being perhaps the most complete effort, though it is not "in any way an effective reasoner" [12].

To support our development of intelligent information fusion systems we have the need for a rule language and engine that will permit the representation of complex logical conditions and support a reasonable set of DL constructs while remaining sound, complete and tractable. Furthermore we would like the language to permit a high enough level of abstraction so that we do not have to think and code at the low level of raw triples (a modern day equivalent of programming in assembly language). Towards this end we have developed a forward-chaining inference engine called BaseVISor that 1) is based on a Rete network optimized for the processing of triples, 2) is able to process RuleML [13] rules containing n-ary predicates and 3) incorporates the axioms and consistency rules for R-Entailment [14] which supports all of RDF/RDFS and a part of OWL semantics along with simple Horn rules. This paper describes the key features of BaseVISor. It explains the process used for translating RuleML rules and facts with n-ary predicates to-and-from BaseVISor rules and facts with binary predicates; and it discusses the implementation of the R-Entailment

axioms and illustrates the system's application to some common examples. The paper concludes with information on how to obtain the BaseVISor distribution package.

2. BaseVISor

At the heart of BaseVISor is a Rete-based [15], forward-chaining inference engine optimized for the processing of RDF triples. This engine is similar to other Rete-based engines such as Jess [16] and CLIPS [17]. The primary difference from these engines is that BaseVISor uses a simple data structure for its facts (i.e., triples) rather than arbitrary list structures; the advantage this affords is greatly enhanced efficiency in pattern matching which is at the core of a Rete network (cf. [8]). BaseVISor is written in Java and includes an API for readily adding user-defined procedural attachments. A large subset of the built-ins defined for SWRL is included in the BaseVISor distribution as built-in procedural attachments. The BaseVISor API also facilitates the embedding of the system within another Java application.

BaseVISor's native rule language uses a simple XML syntax to define facts, create rules and issue queries. A fact is a **triple** defined by **subject**, **predicate** and **object** elements, as shown in the following two examples:

```
<triple>
  <subject resource="#Bill"/>
  <predicate resource="#spouseOf"/>
  <object resource="#Hillary"/>
</triple>

<triple>
  <predicate resource="#age"/>
  <subject resource="#Bill"/>
  <object datatype="&xsd;integer">50
  </object>
</triple>
```

The subject and predicate elements of a fact always refer to a resource which is specified using the ***resource*** attribute. The object element of a fact can be either a resource or a literal. In the latter case the value is defined in the content of the element and the XSD datatype of the literal is specified using the ***datatype*** attribute; if no datatype is specified, a plainLiteral is assumed. The subject, predicate and object elements can appear in any order within a triple element.

Rules are defined within a **rulebase** with each **rule** consisting of a **body** element and a **head** element (which can occur in either order). The ***name*** attribute can be used to assign a name to a rulebase or rule. An example of the typical structure of a **rule** within a **rulebase** is shown here:

```
<rulebase name="Rule Set A">

  <rule name="Rule 1">
    <body>
      <triple>…</triple>
    </body>

    <head>
      <assert>
        <triple>…</triple>
      </assert>
    </head>
  </rule>
    …
</rulebase>
```

The body of a rule usually contains one or more **triples** which share the syntax used by facts described above except that triples within rule bodies can contain variables. A variable is used by providing its name as the value of the ***variable*** attribute on the subject, object or predicate element, as in this example:

```
<triple>
  <subject variable="X"/>
  <predicate resource="#spouseOf"/>
  <object variable="Y"/>
</triple>
```

In addition to triples, bodies may also contain procedural attachments, either built-ins or user-defined. Built-in procedural attachments include **print/println** to output text to the console, **bind** for explicitly binding a value to a variable, **assert** for asserting a triple into the fact base, **retract** for retracting a triple from the fact base, **gensym** for generating a symbol to represent a resource, **not** for matching on the absence of one or more triples within the fact base, equality/inequality functions (i.e., **>,<,>=,<=,=,neq**) and common mathematical functions (e.g., **+,-,*,/,mod,****). Procedural attachments may also occur within the head of a rule except for **not** which is restricted to use within rule bodies. The head is typically where assertions and retractions are made.

It is possible to query the fact base outside of a rule using the **query** element and placing within it one or more triples containing variables. Here is a sample query:

```
<query name="Vertebrate and Mammal">
  <triple>
    <subject variable="X"/>
    <predicate resource="rdf:type"/>
    <object resource="#Vertebrate"/>
  </triple>
  <triple>
    <subject variable="X"/>
    <predicate resource="rdf:type"/>
    <object resource="#Mammal"/>
  </triple>
</query>
```

The result of a query is a list of variable bindings that satisfy the constraints of the query.

The usual way of using BaseVISor involves writing an XML file containing facts (i.e., raw triples), a rulebase and possibly one or more queries and then submitting the file to the standard BaseVISor Batch processor. It is also possible to specify the inclusion other files during batch processing. In particular you can include multiple rulebases using the **include** element by simply specifying the location of the rulebase using either the ***path*** or ***url*** attribute. This **include** element can also be used to import an RDF/OWL document by specifying the ***lang*** attribute to be "RDF". Note that importing an RDF/OWL document has the effect of asserting all of the explicit RDF triples resulting from the parsing of the document but none of the semantically derivable triples are asserted. Deriving these inferable triples requires the use of an **axiomSet** as describing below in Section 4 on R-Entailment.

3. RuleML to BaseVISor Rules

While the native BaseVISor language is simple and concise it is not expected that many people will choose to use it directly as a language for manually writing rules. Most real-world problems deal with concepts at a higher abstraction level than raw triples. In these cases, being forced to think and compose rules in terms of low level triples is tedious at best (see [18]). Instead, we expect users to either develop a high-level language suited for their specific needs, which can then be converted to BaseVISor rules, or use RuleML and take advantage of the built-in ability of BaseVISor to convert RuleML rulebases into native BaseVISor code. This conversion is performed automatically when including a rulebase that is identified as being written in RuleML, e.g.,

```
<include lang="RuleML" path="gen.rml"/>
```

BaseVISor carries out the conversion through the application of an XSLT script which has been written to work with most versions of RuleML although not all of the features of later versions are supported. In general, BaseVISor is relevant to the modules that support Hornlog rules; specifically, the RuleML elements handled by the translation script include: **Implies/imp**, **body/_body**, **head/_head**, **And/and**, **Atom/atom**, **Rel/rel**, **Ind/ind**, **Data**, **Equal**, **Naf** and **Query**.

For RuleML rules that only contain unary or binary predicates, such as M. Dean's GEDCOM rulebase [19], the translation is straight forward and amounts to little more than changing element names and converting atoms into triples. Atom conversion involves mapping the first element of the atom to a predicate element, the second element to a subject element and the third element to an object element and then determining for each whether it represents a variable (**<Var>**), a resource (**<Ind>**) or a datatype literal (**<Data>**).

When n-ary predicates are used in the RuleML rules things become more complicated by the need to convert everything down into binary predicates. This process needs to be done for all facts defined by n-ary predicates and all rules involving n-ary predicates. To convert an n-ary predicate fact, a new resource is created to which the predicate's name and its n arguments can be associated; the n-ary fact is then replaced with a set of n+1 binary predicates (i.e., triples) in which the new resource serves as the subject of each triple. This approach is modelled after use case three in [20]. As an example, consider the 3-ary predicate fact

```
parentsOf('Bill', 'Hillary', 'George')
```

which would be converted into binary predicates represented by the following four triples:

```
<triple>
  <subject resource="#__R1"/>
  <predicate resource="#__property"/>
  <object resource="#parentsOf"/>
</triple>

<triple>
  <subject resource="#__R1"/>
  <predicate resource="#__arg1"/>
  <object resource="#Bill"/>
</triple>
```

```
<triple>
  <subject resource="#__R1"/>
  <predicate resource="#__arg2"/>
  <object resource="#Hillary"/>
</triple>

<triple>
  <subject resource="#__R1"/>
  <predicate resource="#__arg3"/>
  <object resource="#George"/>
</triple>
```

The resource #__R1 is (by its use as a subject) inferred by the system to be an instance of some (anonymous) rdfs:Class which need not be explicitly defined. Likewise, the resource #__property is inferred to be an rdfs:Property even though no explicit statement to this effect is (or need be) made.

When an n-ary predicate occurs within a rule body or head it is similarly converted into a set of binary predicates, but in this case the subject of the generated binary predicates will be a variable with a randomly generated name that is the same for all n+1 triples corresponding to the n-ary predicate. When the n-ary predicate appears in the head of a rule the generated variable name is first bound to a new resource created by an explicit call to **gensym** and then each of the binary predicate triples is individually asserted. For example, consider the following RuleML head taken from a rule in H. Boley's discount rules [21]:

```
<_head>
    <atom>
      <_opr><rel>discount</rel></_opr>
      <var>customer</var>
      <var>product</var>
      <ind>5.0 percent</ind>
    </atom>
</_head>
```

This would be translated into a BaseVISor rule head of the following sort:

```
<head>
  <bind variable="?Var-d0e10">
    <gensym/>
  </bind>
  <assert>
    <triple>
      <predicate
        resource="#__predicate"/>
      <subject
        variable="?Var-d0e10"/>
      <object
        resource="#discount"/>
    </triple>
  </assert>
  <assert>
    <triple>
      <predicate
        resource="#__arg1"/>
      <subject
        variable="?Var-d0e10"/>
      <object
        variable="?customer"/>
    </triple>
  </assert>
  <assert>
    <triple>
      <predicate
        resource="#__arg2"/>
      <subject
        variable="?Var-d0e10"/>
      <object
        variable="?product"/>
    </triple>
  </assert>
  <assert>
    <triple>
      <predicate
        resource="#__arg3"/>
      <subject
        variable="?Var-d0e10"/>
      <object
        resource="#5.0 percent"/>
    </triple>
  </assert>
</head>
```

When a rule with a head like this fires, the four triples are asserted into the fact base and can be matched up with the bodies of rules that contain a similarly structured set of triples containing one or more variables. When rules stop firing the fact base can be dumped to the console or queried. If dumped to the console there is a second XSLT script that can be applied to the inferred facts to reverse the translation of any binary-encoded n-ary predicates back into their standard RuleML (version 0.9) form. For example the discount business rules and sample facts [21] were converted to BaseVISor rules and processed by the inference engine resulting in the following inferred facts (translated back into their n-ary form):

```
<Atom>
  <Rel>premium</Rel>
  <Ind>Peter Miller</Ind>
</Atom>

<Atom>
  <Rel>discount</Rel>
  <Ind>Peter Miller</Ind>
  <Ind>Honda</Ind>
  <Ind>5.0 percent</Ind>
</Atom>
```

```
<Atom>
   <Rel>discount</Rel>
   <Ind>Peter Miller</Ind>
   <Ind>Porsche</Ind>
   <Ind>7.5 percent</Ind>
</Atom>
```

4. R-Entailment Rules

In [14], H. ter Horst proposed a language consisting of RDF, RDFS, part of OWL DL and simple rules for which he defined the general notion of R-Entailment. The desirable characteristics of this language are 1) that the set of entailed triples is (usually) finite and (usually) in PSPACE making it well suited for a forward-chaining inference engine, 2) it is decidable (for rules that do not introduce blank nodes) and 3) its complexity is in NP (for rules that do not introduce blank nodes and that satisfy a bound on the size of rule bodies) but reduces to being in P if the target RDF graph is grounded. The price of obtaining these qualities is that not all of OWL is supported. The language does include all RDF and RDFS elements, plus rules with variables and the following OWL elements:

```
owl:FunctionalProperty
owl:Restriction
owl:InverseFunctionalProperty
owl:onProperty
owl:SymmetricProperty
owl:hasValue
owl:TransitiveProperty
owl:someValuesFrom
owl:sameAs
owl:allValuesFrom
owl:inverseOf
owl:differentFrom
owl:equivalentClass
owl:disjointWith
owl:equivalentProperty
```

Notable elements missing from this list include:

```
owl:ObjectProperty
owl:DatatypeProperty
owl:cardinality
owl:minCardinality
owl:maxCardinality
owl:oneOf
owl:unionOf
owl:complementOf
owl:intersectionOf
```

The absence of support for **owl:ObjectProperty** and **owl:DatatypeProperty** is not an issue in practice because as soon as a resource is used as a predicate it is identified by the R-Entailment axioms as being an **rdf:Property**. And as explained in [14], part of the semantics of **owl:unionOf**, **owl:intersectionOf** and **owl:complementOf** can be obtained by alternative means. For the case of **owl:intersectionOf** we have gone one step further and implemented an extension to R-Entailment as described in Section 5.

The R-Entailment semantics are defined by a set of forty-four proper rules, one axiom, several dozen axiomatic triples plus two consistency rules. These have all been translated into BaseVISor triples and rules. In order to implement the conditions of a number of the rules (particularly those dealing with literals and blank nodes) a set of procedural attachments were developed to handle the identification of literals and resources (i.e., **isLiteral**, **isPlainLiteral**, **isResource**, **isLiteralBlank**, and **isTypedLiteral**) and to generate blank nodes and obtain literal values (i.e., **getLiteralBlankNode**, **getTypedLiteralType** and **getLiteralBlankNodeLiteral**).

An example of part of the R-Entailment implementation in BaseVISor is shown here for illustration purposes. The full set of axioms and rules is included with the BaseVISor distribution (see Section 6). The following is a small subset of the P axiomatic triples:

```
<triple>
  <subject
    resource="owl:FunctionalProperty"/>
  <predicate
    resource="rdfs:subClassOf"/>
  <object
    resource="rdfs:Property"/>
</triple>

<triple>
  <subject
    resource="owl:SymmetricProperty"/>
  <predicate
    resource="rdfs:subClassOf"/>
  <object
    resource="rdfs:Property"/>
</triple>

<triple>
  <subject
    resource="owl:TransitiveProperty"/>
  <predicate
    resource="rdfs:subClassOf"/>
  <object
    resource="rdfs:Property"/>
</triple>
```

The following two rules provide a partial representation of the BaseVISor implementation of R-Entailment rules that make use of the procedural attachments created specifically for the purpose of supporting R-Entailment. The ***name*** attribute values on these rules relate them to the R-Entailment rules as labeled in [14].

```
<rule name="rdf2-D">
  <body>
    <triple>
      <subject variable="v"/>
      <predicate variable="p"/>
      <object variable="l"/>
    </triple>
    <isTypedLiteral>
      <param variable="l"/>
    </isTypedLiteral>
  </body>
  <head>
    <bind variable="bl">
      <getLiteralBnode>
        <param varaible="l"/>
      </getLiteralBnode>
    </bind>
    <bind variable="a">
      <getTypedLiteralType>
        <param variable="bl"/>
      </getTypedLiteralType>
    </bind>
    <assert>
      <triple>
        <subject variable="bl"/>
        <predicate resource="rdf:type"/>
        <object variable="a"/>
      </triple>
    </assert>
  </head>
</rule>


<rule name="rdfs1">
  <body>
    <triple>
      <subject variable="v"/>
      <predicate variable="p"/>
      <object variable="l"/>
    </triple>
    <isPlainLiteral>
      <param variable="l"/>
    </isPlainLiteral>
  </body>
  <head>
    <bind variable="bl">
      <getLiteralBlankNode>
        <param varaible="l"/>
      </getLiteralBlankNode>
    </bind>
    <assert>
      <triple>
        <subject variable="bl"/>
        <predicate resource="rdf:type"/>
        <object
          resource="rdfs:Literal"/>
      </triple>
    </assert>
  </head>
</rule>
```

In BaseVISor the R-entailment axioms can be used to derive inferable facts from RDF/OWL triples (either those asserted or those derived by the firing of user defined rules) by including the **axiomSet** element as follows:

```
<axiomSet name="R-Entailment"/>
```

Inclusion of this element has two effects: the R-Entailment rules and axioms are loaded into the Rete network and are applied to all triples added to the fact base and 2) any BaseVISor rules that are loaded into BaseVISor are first processed by the three R-Entailment rules dealing with literals in rules (i.e., rules lg-R, rdf2-DR and rdfs1-R from [14]) .

5. Discussion

Some may question the value of combining n-ary predicate translation with R-Entailment since the reified binary predicates cannot be reasoned about with RDF/RDFS/OWL axioms. One example of where it does make sense is when an R-Entailment-based ontology is used to define classes and properties but rules are used to determine membership in some of the classes or to assign values to some of the properties (cf., [10]). In such a case it might be simpler to compose some complex membership rules using n-ary predicates (e.g., for chaining between rules) even though the final facts that would be of interest would be binary predicates (i.e., RDF triples) that could lead to additional derived triples via the firing of some of the R-Entailment rules.

According to the conditions of R-Entailment, variables are not permitted in the heads of rules unless they also appear in the body. It would seem that the approach used for translating n-ary predicates into binary predicates violates this condition. This situation could be remedied by moving the **bind** statement from the head to the body, without loss of generality or any effect on the performance of the system. It has been left this way for readability.

We have run BaseVISor with R-Entailment on a number of different rule sets and have been using it to support our development efforts in the area of information fusion and situation awareness. In one of these efforts we had the need to define the intersection of two classes. Although **owl:intersectionOf** is not part of R-Entailment it was relatively easy to extend R-Entailment to accommodate this through the addition of the following rule:

```
<rule name="simple-intersectionOf">
  <body>
    <triple>
      <subject
        variable="u"/>
      <predicate
        resource="owl:intersectionOf"/>
      <object
        variable="classlist"/>
    </triple>
    <triple>
      <subject
        variable="classlist"/>
      <predicate
        resource="rdf:first"/>
      <object
        variable="v"/>
    </triple>
    <triple>
      <subject
        variable="classlist"/>
      <predicate
        resource="rdf:rest"/>
      <object
        variable="rest-list"/>
    </triple>
    <triple>
     <subject
       variable="rest-list"/>
     <predicate
       resource="rdf:first"/>
     <object
        variable="w"/>
    </triple>
    <triple>
      <subject
        variable="x"/>
      <predicate
        resource="rdf:type"/>
      <object
        variable="v"/>
    </triple>
    <triple>
      <subject
        variable="x"/>
      <predicate
        resource="rdf:type"/>
      <object
        variable="w"/>
    </triple>
  </body>
  <head>
    <assert>
      <triple>
        <subject
          variable="x"/>
        <predicate
          resource="rdf:type"/>
        <object
          variable="u"/>
      </triple>
    </assert>
  </head>
</rule>
```

While this approach does not support the more general notion of **owl:intersectionOf** that permits the intersection of an arbitrary number of classes, it is possible to achieve the same effect by the nested use of **owl:intersectionOf** as in this example:

```
<owl:intersectionOf>
   <owl:Class
     rdf:about="SomeClass"/>
   <owl:intersectionOf>
     <owl:Class
       rdf:about="SomeOtherClass"/>
     <owl:Class
       rdf:about="YetAnotherClass"/>
   </owl:intersectionOf>
</owl:intersectionOf>
```

This approach was successfully employed for a Capability Maturity Model Integration (CMMI) ontology recently developed by a Northeastern graduate student. This ontology consists of over 300 classes and a handful of properties used to classify the maturity level of an organization's software development process [22]. Using BaseVISor and R-Entailment the maturity level of an organization exhibiting specific characteristics can be parsed and classified within a few seconds; attempts to process the ontology using a couple of well known tableaux reasoners failed to produce usable results.

It remains to be determined exactly what effect the addition of this intersectionOf rule to R-Entailment might have on the complexity of the language, but the single, simple assertion made by the rule and the lack of direct recursion make it appear relatively innocuous.

At the time of this writing we do not have sufficient experimental results to include in the paper. We have yet to conduct thorough performance evaluations (although see [18] for a simple performance comparisons to Jess) that are needed to clearly demonstrate the benefits of BaseVISor.

6. Obtaining the BaseVISor Distribution

BaseVISor is being made available free of charge for research and educational purposes. The binary distribution along with documentation and several sample rule sets can be downloaded from http://www.vistology.com/BaseVISor.

7. Conclusion

This paper described the core triples-based inferencing capabilities of BaseVISor and introduced two extensions to the system. In the first extension, BaseVISor has been given the ability to process RuleML rules, including those with n-ary predicates which are automatically translated to and from BaseVISor's native binary predicates encoded within triples. In the second extension, R-Entailment rules and axioms have been translated into BaseVISor rules and facts and specialized procedural attachments were written to enable the semantics for RDF, RDFS, part of OWL and rules to be realized within BaseVISor. With this latter capability BaseVISor moves beyond the realm of rule based inference engines into the space of description logic reasoners. BaseVISor is freely available for research and education purposes.

Acknowledgements

This work was partially supported by U.S. ONR STTR Contract Number N00014-05-C-0367 and U.S. Army SBIR Contract Number W15P7T-05-C-T204.

References

[1] W3C Web Ontology Language (OWL) homepage. http://www.w3.org/2004/OWL/

[2] The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, 2002. ISBN 0521781760. Edited by F. Baader, D. Calvanese, et al.

[3] FaCT++ homepage. http://owl.man.ac.uk/factplusplus/

[4] Pellet homepage. http://www.mindswap.org/2003/pellet/index.shtml

[5] RACER homepage. http://www.racer-systems.com/

[6] Cerebra homepage. http://www.cerebra.com/index.html

[7] I. Horrocks and P. F. Patel-Schneider. A Proposal for an OWL Rules Language. In Proc. of the Thirteenth Int'l World Wide Web Conf.(WWW 2004). ACM, 2004.

[8] C. Matheus, K. Baclawski, M. Kokar, and J. Letkowski, Constructing RuleML-Based Domain Theories on top of OWL Ontologies. In Proceedings of Rules and Rule Markup Languages for Sematic Web: Second International Workshop, RuleML 2003, Sanibel Island, Florida, October 2003.

[9] C. Matheus, Using Ontology-based Rules for Situation Awareness and Information Fusion. Position Paper presented at the W3C Workshop on Rule Languages for Interoperability, April 2005.

[10] M. J. O'Connor, H. Knublauch, S. W. Tu, B. Grossof, M. Dean, W. E. Grosso, M. A. Musen. Supporting Rule System Interoperability on the Semantic Web with SWRL. Fourth International Semantic Web Conference, Galway, Ireland, 2005.

[11] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A semantic web rule language combining OWL and RuleML. W3C Member Submission, 21 May 2004. http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/

[12] Hoolet homepage. http://owl.man.ac.uk/hoolet/

[13] RuleML homepage: http://www.ruleml.org/

[14] H. ter Horst. Combining RDF and Part of OWL with Rules: Semantics, Decidability, Complexity. In Proc. of the Fourth Int'l Semantic Web Conference. Y. Gil et al. (Eds.): ISWC 2005, LNCS 3729, pp. 668–684, 2005.

[15] C. L. Forgy, "Rete: a fast algorithm for the many pattern/many object pattern match problem," Artificial Intelligence, 1982, pp.17-37.

[16] Jess homepage. http://herzberg.ca.sandia.gov/jess/

[17] CLIPS homepage. http://www.ghg.net/clips/CLIPS.html

[18] C. Matheus, M. Kokar, K. Baclawski and J. Letkowski, Using SWRL and OWL to Capture Domain Knowledge for a Situation Awareness Application Applied to a Supply Logistics Scenario. In Proc. of International Conference on Rules and Rule Markup Languages for the Semantic Web, RuleML-2005, Galway, Ireland, November, 2005.

[19] M. Dean's GEDCOM homepage. http://www.daml.org/2001/02/gedcom-ruleml/

[20] N. Noy and A. Rector (Eds.). Defining N-ary Relations on the Semantic Web. W3C Working Group Note 12 April 2006. http://www.w3.org/TR/swbp-n-aryRelations/

[21] H. Boley's discount business rules. http://www.ruleml.org/exa/0.8/discount.ruleml

[22] Capability Maturity Model Integration (CMMI), Version 1.1 CMMI for Software Engineering (CMMI-SW, v1.1) Staged Representation. Technical Report CMU/SEI-2002-TR-029, ESC/TR-2002-029, Carnegie Mellon, Software Engineering Institute, Pittsburgh, 2002.