BaseVISor: A Forward-Chaining Inference Engine Optimized for RDF/OWL Triples

Christopher J. Matheus, Robert Dionne, Douglas F. Parent
Versatile Information Systems, Inc., Framingham, Massachusetts, U.S.A

Kenneth Baclawski and Mieczyslaw M. Kokar
Northeastern University, Boston, Massachusetts, U.S.A

Abstract

BaseVISor is a forward-chaining inference engine based on a Rete network optimized for the processing of RDF triples. 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, the R-Entailment axioms, axiomatic triples and consistency rules are imported into the engine and then used to derive all triples entailed by a base set of triples. Operation of the system will be demonstrated using sample rule sets employing RuleML and R-Entailment.

BaseVISor [1] is a Rete-based [2], forward-chaining inference engine optimized for the processing of RDF triples. It is similar to other Rete-based engines such as JESS [3] and CLIPS [4]. 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, which permits greatly enhanced efficiency in pattern matching which is at the core of a Rete network. In BaseVISor, a clause within the body or head of a rule either represents an RDF triple or invokes a procedural attachment (either built-in or user defined). BaseVISor is written in Java and includes an API for easily adding user-defined procedural attachments. A large subset of the built-ins defined for SWRL [5] are included in the BaseVISor distribution as built-in procedural attachments. The beta release of BaseVISor will be available for free download at Versatile Information Systems' homepage: http://www.vistology.com.

BaseVISor's native 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, e.g.:

```
<triple>
  <subject resource="#Bill"/>
  <predicate resource="#age"/>
  <object datatype="xsd:integer">45</object>
</triple>
```

The subject and predicate elements of a fact always refer to a resource specified using the *resource* attribute. The object element of a fact can be either a resource or a literal, in which case the value is defined in the content of the element and the XSD datatype of the literal is specified using the *datatype* attribute. 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 (occurring in either order). The *name* attribute can be used to assign names 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 that share the syntax used by facts described above except that triples within rule bodies can contain variables. Variables are indicated by providing the variable's name as the value of the *variable* attribute on the subject, object or predicate element, e.g.:

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

An abbreviated syntax is also provided that greatly simplifies the writing of triples for multiple properties that share the same subject. In this alternative syntax an `Individual` element is used to identify the subject and an unlimited number of predicates pertaining to the individual can be included in the element's body with their associated resource/variable values. Furthermore, the objects of the predicates can be additionally qualified with predicates of their own, and so on to any arbitrary depth. For example, the following statement says that Bob has a brother named Bill and a sister named Jill and that Bill is 45 years old and is married to Sally who has a sister named Kate:

```
<Individual resource="#Bob">
  <brother resource="#Bill">
    <age datatype="xsd:integer">45</age>
    <spouse resource="#Sally">
      <sister resource="#Kate"/>
    </spouse>
  </brother>
  <sister resource="#Jane"/>
</Individual>
```

This abbreviated syntax has the advantage of being more succinct (and thus less prone to common typographical errors) while more clearly depicting the structural organization of the data (making it easier to catch logical errors).

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., `>,<,>=,<=,=,!=`), common mathematical functions (e.g., `+,-,*,/,mod,**`) and date and string manipulation functions. Any procedural attachment may occur within the head of a rule except for `not` which is restricted to use within rule bodies.

BaseVISor may be embedded within a Java application or it can be used as a stand alone inference engine. Use of the stand alone version involves writing an XML file containing facts (i.e., raw triples), a rulebase and possibly one or more queries which is then submitted to the standard BaseVISor Batch processor. It is also possible to write statements to include other files in the batch processing. In particular you can include external rulebases with the `include` element and import an RDF document with the `importRDF` element.

Embedding BaseVISor within a Java application permits finer control of the loading and operation of the inference engine and allows the use of user-defined procedural attachments. Such attachments can be used in the body of rules to perform tests that are more appropriately handled by procedural code; in the context of rule bodies or heads, procedural attachments may be used to interface to other applications (e.g., a database or web service).

BaseVISor is also able to process RuleML [6] rulesets by transforming them into BaseVISor rulesets. In cases where predicates of arity greater than two are used within a RuleML rule, BaseVISor will automatically convert them into an equivalent binary predicate representation along the lines suggested in [7].

R-Entailment [8] is a language proposed by H. ter Horst that combines RDF, RDFS and a part of OWL DL with simple Horn-style rules. Desirable characteristics of this language are that it is finite (unlike OWL which inherently entails an infinite model), it is decidable (under certain conditions) and its complexity is P (under certain constraints). The R-Entailment axioms have been implemented as BaseVISor rules that can be preloaded into BaseVISor's Rete network. In this way it becomes possible to use BaseVISor to deduce the set of triples entailed by an RDF/OWL document that conforms to the R-Entailment specifications.

BaseVISor is available for research purposes from http://www.vistology.com/basevisor.

References

1 C. Matheus, K. Baclawski and M. Kokar. BaseVISor: A Triples-Based Inference Engine Outfitted to Process RuleML and R-Entailment Rules. In *Proc. of the 2nd Internation Conference on Rules and Rule Languages for the Semantic Web*, Athens, GA, Nov. 2006.

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

3 Jess homepage. http://herzberg.ca.sandia.gov/jess/

4 CLIPS homepage. http://www.ghg.net/clips/CLIPS.html

5 Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML, 2004. http://www.daml.org/rules/proposal/

6 RuleML homepage: http://www.ruleml.org/

7 N. Noy, A. Rector, P. Hayes and C. Welty. Defining N-ary Relations on the Semantic Web. W3C Working Group Note 12 April 2006. W3C Working Group Note 12 April 2006. http://www.w3.org/TR/swbp-n-aryRelations/

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