Understandability and Semantic Interoperability of Diverse Rules Systems

Adrian Walker, Reengineering [1]

Position Paper for the W3C Workshop on Rule Languages for Interoperability
27-28 April 2005, Washington, D.C.

1 Introduction

There are two issues that, if resolved, could speed up the adoption of the Semantic Web.

Issue 1: It is difficult for application writers, and for users, to relate the meanings of the machine oriented notations of the SW, such as in RDF and in technical rule notations, to the meanings of real world business and scientific concepts.

Issue 2: It is difficult for a user to entirely trust a real world result that has been obtained by a complex reasoning process over distributed RDF or other data. The data, the rules, or the reasoning engine could be in error, or could be used in a wrong context.

This paper suggests a way of resolving the issues, and of promoting interoperability, using rule technologies.

Rules are often characterized as being "forward chaining" or "backward chaining". However, a rule engine controls the way in which chaining is done, not the rules. Thus, a given set of rules could be executed forward, backward, or in other ways, and the results will in general depend on which variation of which method is chosen.

This has important consequences for understandability and interoperability of rules systems on the Semantic Web:

- It will not be sufficient only to recommend a common rule syntax.

- There are many different rule engines in commercial use. So, rather than trying to select a single kind of engine to recommend, we should recommend a way in which diverse rule systems can interoperate. One possible way is outlined in this Section 2.

- In the medium term, the actions of interoperating rules systems can be made understandable to nontechnical users by including some lightweight natural language capability, together with end user level English explanations built on proof trees.

- In the long term, the logical semantics of diverse, interoperating rule systems can be clarified by model theory and proof theory studies, together with proofs of soundness, completeness, and (where possible) termination.

The next Section outlines an approach in which a vendor-neutral collection of messages could allow diverse rule systems to interoperate. Section 3 describes a practical example covering some details of the approach. The example is an online rule system that extends the *data semantics* of the SW with *application semantics* at the real world, end user level. The system is based on some published research into inferencing based on a logical model theory. We illustrate the use of lightweight natural language and end-user level explanations by means of some simple SW examples that one can view, run and change, using a browser.

2 Messages for Interoperation of Diverse Rules Systems

There will be many SW "rulespaces", many of them corresponding to vendor-specific rule systems. How can such rulespaces interact, and what should be in each rulespace?

2.1 Interaction For interaction, we suggest a loose coupling of rulespaces, using the following kinds of messages.

An *input capability message* consists of one or more table headings that describe what data a rulespace can accept as input.

An *output capability message* consists of one or more table headings that describe what results a rulespace can produce.

An *explanation capability message* consists of one or more table headings that describe classes of results for which a rulespace can produce English proof trees as output.

A *data message* consists of one or more tables of data with headings. Some or all of the tables in a message can consist of RDF triples. In a data message, an empty table, consisting only of a heading, is a request for rule processing that will fill in the table. A specially marked line in a table is a request for an English explanation of that line.

An *explanation message* is a proof tree having English (or other natural language) sentences at its nodes.

For transport across the SW, when needed, a *rulespace message* consists of a set of rules, together with a means for executing them, plus capabilities as above and URIs as described below. Different rulespace messages may contain rules in any syntax, provided they are supported by two kinds of semantics, as described next. (In the simple case that no rulespaces are moved across the web, there are no rulespace messages -- each rulespace is sufficiently described by its input-output behavior as above.)

2.2 What Should be in Each Rulespace A rulespace should – where possible -- not only contain rules in a particular syntax. It should also contain a full set of semantics for the rules, in the sense of real world end-user meanings in lightweight English, and also in the sense of computational logic semantics. We suggest including as many as possible of the following items:

1. The input, output, and explanation capability messages.

2. An executable grammar of the rules.

3. A logical model- or fixpoint-theory defining exactly what should be inferred from any given set of rules and facts. (Although if-then rules seem to have simple intuitive meanings at first glance, the devil is in the details, and a logical model theory provides a clear statement of what consequences should follow from a collection of rules and facts.)

4. A reference implementation of a proof theory corresponding to the model theory, aka a rule interpreter. This need only be efficient enough to run regression test cases offline.

5. A paper setting out a formal proof that the rule interpreter is sound and complete with respect to the model theory, and (where possible) terminating.

6. A gradually expanding set of regression test cases.

7. One or more efficient rule engines intended for real world SW applications.

8. A means of extracting English explanations, at the business analyst level, of results computed by a rule engine. (Such explanations can also be viewed as *plans*, e.g. for linking web services.)

9. A means whereby Google, or similar search engines, can index and retrieve sets of rules.

10. For use in *rulespace messages*, each rulespace defining an application should be packaged on the web together with URIs of the relevant versions of 1-6 and 8-9. The package should also include a physical copy of 7 in object form, and where possible also in source form.

The above could lead to a large payoff in reliable, interoperable, and understandable distributed rule systems.

3 A Practical Instance of the Approach

To see one way that the above general approach can work out, we describe a system that is online, and we describe some simple examples of its use.

3.1 The Rulespace and Engine A system that exemplifies many of the above general features is online and available for experimental use. It is called Internet Business Logic (IBL), and it can be used to write and run rules, by pointing a browser to [1]. Non-commercial use of the system is free.

The system supports the writing and running of rules in open vocabulary English. The approach to natural language processing is lightweight, in the sense that there is no need to maintain a dictionary or grammar of English. However, the real world semantics are strict, in that the system only treats two sentences as the same if there are rules that so specify.

The lightweight approach to English is supported by an inference engine that assigns a highly declarative meaning to the rules. It is based on the model theory and computational semantics in [2,3]. [2] introduces the model theory, and describes a simple, but inefficient, rule interpreter. [3] describes a more efficient rule engine, and gives proofs that the engine is sound, complete and terminating with respect to the model theory.

As mentioned in Section 1, while collections of rules are often described as either "forward chaining" or "backward chaining", the kind of chaining is a property of the rule engine, not of the rules. In the IBL system, the engine performs both forward and backward chaining, and switches automatically between the two. However, people who write rules need not be concerned with such procedural details, since the end result is that rules simply have a highly declarative meaning. The engine in the IBL system is based on the one described in [3].

Since the rules are written in executable English, the engine can produce English explanations of its reasoning on demand. An explanation starts with the headlines, and one can drill down into detail on a web page as needed. Also, since the rules are in English, they are indexed by Google and other search engines. The *Complete Examples* section [4] of the IBL is indexed in this way.

Here are some examples of using the system for tasks related to the SW. Each example can be viewed, run, and changed by pointing a browser to [1]

3.2 An RDF Query Example In the paper *A Comparison of RDF Query Languages* [5], Haase et al describe a use case in which 14 test questions are put to some RDF data about published papers and their authors. The paper describes several SW query languages, and shows that none of those languages can answer all 14 questions.

It is straightforward to write rules that allow the IBL system to answer all 14 questions [6]. However, reasoning over RDF, even in this relatively simple example, is quite hard for a person to follow. A nontechnical user, who was unsure whether to act on an answer, would have a hard time convincing himself of its validity simply by looking at the rules and the RDF data, and would find it impossible to do so with a SQL-like query language. On the plus side, the IBL system can supply a step-by-step English explanation of any answer that it produces. It can also explain in English why it failed to give an expected answer.

To run the example, one can point a browser to [1] and select *RDFQueryLangComparison1*.

3.3 A Semantic Resolution Example In the paper *Semantic Resolution for E-Commerce*, [7], Yun Peng et al describe an example of name resolution for e-commerce, using three namespaces: retailer, manufacturer, and shared.

In the example, a retailer orders computers from a manufacturer. However, in the retailer's terminology, a computer is called a PC for Gamers, while in the manufacturer's terminology, it is called a Prof Desktop.

Fortunately, the retailer and the manufacturer can agree that both belong to the class of Workstations/Desktops. We also find out to what extent a Prof Desktop has the required memory, CPU and so forth for a PC for Gamers.

The data tables and English rules describing the task are at [8]. To run the example, one can point a browser to [1] and select *SemanticResolution1*.

3.4 A Simple Example of Abstraction Reasoning for SW Capability Search Suppose that a user or agent on the future SW is searching for an output capability (in the sense of Section 2.1 above) for "sibling". Suppose further that nothing is found. In that case, it may be sensible to search for an abstraction of the relation "sibling", such as "family". If that fails, it may be useful to abstract the search even further.

If a more general output capability is found, it may be associated with other more specific capabilities such as "brother" or "sister", so that the search can succeed via abstraction although it cannot succeed directly.

The data tables and English rules describing a simple version of the task are at [9]. To run the example, one can point a browser to [1] and select *SemanticWebOntology1*.

3.5 One Relation is Transitive Over Another Mikhail Khlopotov gave an example of a task in which one should be able to deduce, from transitivity of the links in an organization tree, that a person who works in a group in a company also works for that company.

Some data tables and English rules describing the task are at [10]. To run the example, one can point a browser to [1] and select *TransitiveOver1*.

3.6 Interoperation of Ontologies Using Different Units of Measure In the paper *Enhancing Data Interoperability with Ontologies, Canonical Forms, and Include Files*, [11], Roger L. Costello described a task in which agents that cooperate use different measures, such as kph and mph respectively for speed.

Some data tables and English rules describing the task are at [12]. An "output capability message" (in the sense of 2.1 above) is

> an-agent sending one unit of a-quantity in some-units should first convert it to a-number some-standard-units

To run the example, one can point a browser to [1] and select *OntologyInterop2*.

4 Summary

We have outlined a general approach to interoperability of rulespaces, in which capabilities, questions, and results are represented in a manner that can support loose coupling of diverse rule systems. We have also suggested that lightweight natural language should play a role where possible, particularly in explaining the complex deductions that arise, even from simple examples.

In the simple case in which rulespaces do not move around the web, input-output behavior based on capability and tabular messages is suggested.

In the case when rulespaces do move around the web, we suggest an approach that can be partly summarized by the slogan "no representation without denotation". In the context of the rules on the SW, the denotation, (or semantics, or meaning) should be addressed at two levels.

The first level of semantics concerns what should in principle be deducible from any collection of rules and facts or RDF triples. We have suggested that a move away from procedural notions of rule execution is warranted by the conceptual complexity of the situation. Thus, rules no longer have "forward chaining" or "backward chaining" meanings. Rather the meaning of a set of rules depends on a logical model theory that specifies what *should* be deducible, together with a formal proof of correctness of a rule interpreter wrt to the model theory.

The second level of semantics concerns computing directly with English sentences that carry real world meaning.

The general approach is illustrated in a practical online system, with a number of tutorial and other examples that one can view, run and change, using a browser.

5 References

[1] http://www.reengineeringllc.com

[2] *Towards a Theory of Declarative Knowledge*, K. Apt, H. Blair and A. Walker. In: Foundations of Deductive Databases and Logic Programming, J. Minker (Ed.), Morgan Kaufman.

[3] *Backchain Iteration: Towards a Practical Inference Method that is Simple Enough to be Proved Terminating, Sound and Complete,* A. Walker. Journal of Automated Reasoning, 11:1-22.

[4] http://www.reengineeringllc.com/demo_agents/

[5] *A Comparison of RDF Query Languages* Peter Haase, Jeen Broekstra, Andreas Eberhart, and Raphael Volz, http://www.aifb.uni-karlsruhe.de/WBS/pha/rdf-query.

[6] http://www.reengineeringllc.com/demo_agents/RDFQueryLangComparison1.agent

[7] *Semantic Resolution for E-Commerce* Yun Peng, Youyong Zou, and Xiaocheng Luan (UMBC) and Nenad Ivezic, Michael Gruninger and Albert Jones. http://www.mel.nist.gov/msidlibrary/doc/semantic.pdf

[8] http://www.reengineeringllc.com/demo_agents/SemanticResolution1.agent

[9] http://www.reengineeringllc.com/demo_agents/SemanticWebOntology1.agent

[10] http://www.reengineeringllc.com/demo_agents/TransitiveOver1.agent

[11] *Enhancing Data Interoperability with Ontologies, Canonical Forms, and Include Files*, Roger L. Costello, http://www.xfront.com/interoperability/CanonicalForms.html.

[12] http://www.reengineeringllc.com/demo_agents/OntologyInterop2.agent