NIST "Position Paper" for the W3C Workshop on Rule Languages for Interoperability

General principles

It is necessary to have a rigorous foundation for the "meta-concepts" of an ontology language. This is done (correctly) in OWL, in RDF, in SWSL/PSL, and in ISO CL. It seems to be done in SweetRules, and presumably therefore for RuleML (v1.0?).

It is useful for ontology development to permit multiple representation languages. But it is necessary for communication to have the reference meta-concept set and force all ontology languages' concepts to map exactly to reference meta-concepts. The map can be one-to-many where needed, but it cannot be partial.

It is useful for many applications to ensure that ontologies can be successfully processed by reasoners. There are two ways to do this:

- disablement -- allow in the language only those constructs that can be processed by a class of reasoner
- marking -- allow many useful constructs in the language, with some kind of syntactic provision that enables a tool to recognize which constructs require which class of reasoner.

Both, but especially the latter, require some "discipline" in the development of ontologies. Most domain users will construct erroneous ontologies or excessively complex ontologies, if we cannot clearly present to them a primer on "how to think" in developing ontologies (in a given language, perhaps). Most domain users will never see (or understand) reasoner traces. We need somehow to ensure that when they get a result, it makes sense. We also need to ensure that when they get a result that they believe, some tests of the ontology have been made to ensure that that result is likely to be valid. (We refer to this as a "logic safety discipline" by analogy to the lessons learned about discipline for parallel programming and "thread safety".)

Whatever the standard form of an ontology is, reasoners will transform the standard form to their internal forms. The above user "discipline" also requires that the transform not lose or add information, so that any result that is true in the transformed ontology is true in the original ontology under the original foundation model. It also requires that any result in the transformed ontology must be mapped back into the terminology and perhaps phrasing of the original, so that it will be meaningful to the user.

Certain constructs that are commonly used in information models cannot per se be processed by DL engines, but there are "circumlocutions" that permit them to be phrased in a language (OWL) that can be processed by DL engines. This is one of the reasons for using the "markup" approach. In many cases, the corresponding transformations can be automated. However, the last observation above requires that the DL results be interpretable by the user, and in some cases these transformations are not readily invertible -- in a one-to-many transform, some of the results may be in terms of artifacts of the transformation.

Requirements for a SW rules language

Particularly for manufacturing and business data, it is necessary for a rules language to support some standard mathematical reasoning systems beyond sets. These requirements arise from two areas of work:

- capturing the rules needed to support translation among measurement values in different units (within and across unit systems)
- capturing structural differences between representations of the same information in different schemas, when the values themselves are semantically equivalent. For example, organization of the elements of person-names, representations of specific time intervals: (begin, end) vs. (begin, duration)
- XML schema "restriction" reasoning, e.g. only some instances of this class are permitted in a given usage; only some elements of a structure are permitted in a given usage; a given attribute never has a value in this usage.
- reasoning about region containment, both geometric and geographic

The following additional concerns have been identified:

- variables to insure co-reference (e.g. instances of a class with the same hasLocation value)
- partOf properties – derived property values, rollup properties of assemblies, co-reference of whole as means of identifying siblings
- property constraints where the results of mathematical functions operating on two or more values (or derived values) are bounded. e.g. the weight of two components cannot exceed 1000 pounds.
- property constraints that specify implication between two properties for a given class, e.g. if the car has the towing package, it must have the heavy duty transmission.
- property constraints that specify incompatibility between two properties for a given class, e.g.: a new MINI cannot both have an automatic hasTransmission value and a supercharged hasEngine value.

These last two cases can be modeled by creating artificial subclasses in OWL and selectively attaching properties to them. Fortunately, these subclasses can be "defined", which allows them to be invisible to the domain expert. Unfortunately, the reasoner results may in some cases be stated in terms of them, which relates to the "meaningful" results issue above.

Dynamic/Production rules language issues

"Production rules" are rules whose primary intent is to direct behavior. Many of these are derivatives of logic-programming languages. We are particularly concerned about the relationship of such rule systems to Webservices and business workflows.

The main characteristics of these systems are:

- the concept of time
- the complex information space
- external agents

The concept of time is critical to them. Some knowledge in these systems is transient. What is true now may not be true earlier or later. The same rule applied at different times may produce different results.

Rules are fired at explicit points in time, not "instantaneously". This concept of time is not the same as that of LP metarules. Rather it refers to points in real time when the relevant parts of the information space are stable (consistent), so that rule firing is meaningful. It is not meaningful to interrogate the antecedents of action rules when the information space is in a transition state. And some agency outside the rule engine determines the state of the information space and invokes the rule engine.

The information space for these rules is a model of a system of active components and the environment of that system. These tend to be complex information models with both "static" and "dynamic" rules that are intended to ensure the characterization of the state of the system and the consistency of that characterization. Some of these rules are visible in the rule engine rulebase, but others are simply implemented by certain components. The antecedents of the corresponding action rules can be complex interrogations of the state of that space.

We cannot force the user to explicitly flatten state structures and nested interrogations and actions into conjunctions of symbols. The source rules languages must allow these things to be written in a more natural way. And software can be written to perform the transform to conjunctions of symbols. When these structures are flattened into rule sets that can be supported by the rule engine, those transforms have to be validated against the original intent, as observed above.

In general, some actions dictated by the rules are executed by the rules engine, but many involve the invocation of external agents, such as webservices, workflow systems and the physical resources they direct. It is necessary for the rules engine to interoperate properly in this environment.

Rules languages for Webservices have to have an LP (or SCLP) "core" capability, extended by invocation of actions of external agents.

Events in the environment, including those produced by the actions of external agents, must be permitted to have representation in the language. (And the rules engine must provide a means by which they can be communicated into its knowledge base.)

The rules language must allow for synchronization of rule-firing with external actions and external events. In particular, meta-rules that control rule-firing may involve events.

External agents are finite resources. It may be impossible to invoke an agent at a given time because it is busy with another invocation. It may be that the agent supports queuing of actions. How this interacts with invocations from the rule system is not clear, but it is clear that it needs to. The rules language may need meta-rules to handle "conflicting resource activations".

Simultaneous actions by external agents can have unintended interferences. It may be necessary to have meta-rules that prevent invocations of one agent while the actions of another agent are in progress.

All of this indicates that the standard rules language must have a standard "meta-rules" language as well.

Realize that there are essentially three classes of rule engine here:
- core SCLP engines
- webservices rules engines, which use most of the above features but with only one kind of resource with standard formal invocation rules and standard behavioral representations.
- workflow rules engines, which use all of the above features, support a variety of resources with a variety of invocation forms and patterns and no standard behavioral representation.

In the business/workflow rules engines in particular, the vendors have invested in the development of a complex system that will only change slowly. In each case, they have built specialized extensions to core SCLP. It is neither necessary nor feasible to get a standard that supports all of their current capabilities. It is necessary to have a standard that doesn't prevent or invalidate the use of those capabilities.

Kinds of rules

While rules language experts think in terms of the nature of rules, modelers often think in terms of the purpose of rules. There are many distinct purposes:
- rules that describe the objects and are useful for inferencing
- rules that are meant to be tests for the validity of the information base, or are meant to be used to determine the times at which the information base is in a "consistent" state
- rules for the representation and organization of the information (typically in some repository)

- rules for the behaviors of agents, in terms of their effects on the system state.

Some rules are intended primarily to guarantee interoperation of software (if not correctness of results). Some are intended to guarantee reliability of the system and its information base. Some are intended to capture important business aspects of the system and its information. In transforming rules into rules engines, it is important to know which is which.

When combinations of these rules are presented together, some assume the others, and it is not unusual for a business requirement to be phrased as a structural requirement, or a business requirement to be phrased in a way that assumes consequences of a structural requirement.

Many rules assume underlying "meta-rules" for the behavior of components. When we transform into an exchange form, it is necessary to ensure that these assumptions are conveyed as well.

Authors:

Edward J. Barkmeyer
edbark@nist.gov, +1 301-975-3528
Evan K. Wallace
ewallace@nist.gov, +1 301-975-3520
Ravi Raman
ravi.raman@nist.gov, +1 301-975-4463

National Institute of Standards & Technology
Manufacturing Systems Integration Division
100 Bureau Drive, Stop 8264
Gaithersburg, MD 20899-8264