

Business Rule Exchange - the Next XML Wave

Margaret Thorpe <mthorpe@ilog.fr>

Abstract

This presentation will cover several dynamic standardization initiatives and technological innovations that are currently converging to create the ability to exchange knowledge via XML, not just data.

1. Part I: Introduction

1.1. The First XMLWave

In the past, the exchange of information between trading partners and applications has been costly and resource-intensive because of the proprietary nature of our data models. This has limited the sharing of information within key industries where information flow is an important component of a multi-step, multi-partner business process. There have been many attempts to standardize the format and contents of this data, some of them vendor-driven and some of them driven by standards bodies. EDI is probably the most familiar of these initiatives, providing a mechanism for partners within an industry to collaboratively define data exchange formats for standard transaction sets. However, the standardization process was time-consuming and resource-intensive, making it slow and only feasible for large corporations to participate in. Additionally, the EDI format was difficult to understand and required the implementation of a layer of specialized EDI translation software in order to interpret the EDI format.

XML is more powerful and more flexible than EDI. It is also simpler to use and less expensive to implement. The XML "standardization process" is easier, but its lack of formality and centralized clearinghouse make for a chaotic environment where many different overlapping "standards" for the same domain can proliferate (ebXML and BPMI, for example). This is because XML doesn't imply any type of standards body or industry validation processes, it simply provides tools to describe shared

vocabularies. There are some repositories evolving on the Web to house XML vocabularies, but the process for contributing, searching and indexing these XML descriptions are still fairly loose.

Nonetheless, XML is being rapidly adopted and used in B2B marketplaces, and it is bringing about some fundamental changes in the ways that companies communicate with each other. XML support is becoming ubiquitous, as demonstrated by its availability within Netscape Navigator 5, Internet Explorer 5 and Windows 2000. XML standards for eBusiness infrastructure are maturing as well, with standards like SOAP, eXML, UDDI and LDAP gaining widespread support. The adoption of XML vocabularies within vertical industries is increasing, with XML standards like Health Level 7 being adopted for the exchange of health records by 100% of the vendors and 80% of the users within the health care industry. XML-based marketplaces are maturing rapidly, with RosettaNet and BizTalk being notable examples.

1.2. The Next XMLWave

The next wave of XML standardization, adoption and technological evolution will make it possible to exchange not just data, but knowledge. This will bring about a whole new set of possibilities for technology vendors, application developers and users of XML technology. The different actors in a given transaction (as distributed, interactive and collaborative as the business process may be) will be able to share not just the "facts" (ie. customer data, order data, invoice data) but also the "business rules" - the business policies and procedures associated with this data within the context of a particular business process. So, for example, a web storefront will be able to use XML to send the business rules representing contractual terms and conditions related to pricing, promotions, discounts, refunds, cancellations and lead times to its trading partners. Partners will be able to understand and execute these business rules on their systems to generate plans and make buying/selling decisions, even if they are using different application software than the storefront uses to execute these same business rules.

In this environment knowledge will become portable and business logic, the most dynamic component of most business applications, will be shipped around the Internet and executed wherever it is determined most efficient. With business logic no longer hard-coded into business applications, these applications will be able to

adapt and change instantly leading to significant reductions in the average time-to-market of software applications, the cost of application development and the maintenance workload of application programmers.

Sound like Utopia? While there are many obstacles to the realization of this dream even in a limited form, there are several initiatives converging right now that are creating the foundation for this environment. This paper describes a number of these initiatives, focusing on the representation of rule languages using XML and the standardization of a Java API for rule engines. It is the author's belief that technological advances occurring in these two important areas will be pivotal in bringing about the next XML wave.

2. Part II: Rules

Rules represent a fundamental conceptual construct in the reasoning process of humans. First formalized by Aristotle over 2,000 years ago, they have been used extensively ever since in the development of our various systems of logic. The computer programming languages that we have invented rely heavily on them, and rules have been used extensively within the Artificial Intelligence community over the last 30 years, with the creation of the PROLOG (PROgramming in LOGic - a declarative, logic-based programming language), and the development of rule-based systems.

Rule-based systems have evolved significantly over the last 20 years and the commercial rule engines available today are quite flexible compared to their predecessors. They are designed to integrate into a company's existing technical infrastructure and rely on industry-standard object-oriented languages like C++ and JAVA. This, combined with their suitability for business rule applications has caused widespread interest in rule-based technology from outside of the AI community.

2.1. Rules and the Evolution of Logic

Aristotle was the first to formalize rules, with his syllogisms using the deductive form *modus ponens*. As a result, even though he lived in the pre- Ice Cream era of civilization, if he had known that:

All ice cream is delicious.

and that:

Breyer's Peanut Butter Fudge Swirl is ice cream.

then he could have concluded that:

Breyer's Peanut Butter Fudge Swirl is delicious.

.....without ever having tasted it!

Over the last two thousand years scientists, philosophers, mathematicians and linguists have fully axiomatized the reasoning process. Copernicus, Kepler & Galileo brought us the Scientific Revolution, where the systematic application of the scientific method became the preferred method for understanding the world. Rene Descartes ingrained in us his methodology of sequential and ordered thinking. The work of Leibnez, Boole and Frege led to the development of the language of symbolic logic - first-order predicate calculus. Russell and Whitehead devised a completely formal system of mathematics based on the application of well-defined rules to axioms and theorems, and, along with Carnap and others argued that natural language was unsuited for scientific and philosophical debate. With this, the philosophy of logical positivism was born, whose goal was to systematize all knowledge through the symbols and rules of a logic-based language.

2.2. Rule-based Languages in AI

AI is the study of the mechanisms underlying intelligent behavior and is characterized by the use of computers to test out and enact those mechanisms. The most important influences on AI theories of reasoning come from 20th century analytical philosophy, and many of the basic analytic tools, like formal logics, were developed by philosophers and mathematicians. The dominant approach to AI today is still based on the physical symbol hypothesis, proposed by Newell and Simon in 1976, which claims that intelligence is achieved through the physical implementation of operations on symbol structures. As a result we have many different representation

languages available to us to describe and reason about the world.

Predicate calculus is the most commonly used representation language in AI because of its well-defined formal semantics and sound and complete inference rules (in particular, modus ponens and resolution). The major advantage of using general methods such as unification and modus ponens is that the resulting algorithms may search any space of logical inferences. The specifics of the problem can be described declaratively using predicate calculus assertions. Thus we can separate the actual problem-solving knowledge from its control and implementation on a computer. It is precisely this characteristic that makes rule-based technologies interesting within the context of this paper, as it: 1) enables business rules to be abstracted and externalized from the rest of the application code, and 2) facilitates the sharing of knowledge as rules can be specified, exchanged and executed independently of the logic that is needed to execute those rules.

Prolog is an important AI language based on the first-order predicate calculus. Prolog programs are collections of descriptive facts about a domain and rules for deriving conclusions from those facts. Prolog, as well as many goal-directed (backward chaining) rule engines, uses a particular type of pattern-directed search algorithm for searching a space of predicate calculus rules and facts. Given a goal, the algorithm uses unification to select the implications whose conclusions match the goal. After unifying the goal with the conclusion of the implication (or rule) and applying the resulting substitutions throughout the rule, the rule premise becomes a new goal - or subgoal. The algorithm then recurs on the subgoal. If a subgoal matches a fact in the knowledge base, search terminates. The series of inferences that led from the initial goal to the given facts prove the truth of the original goal.

Forward-chaining rule engines use a different type of algorithm to accomplish the same thing. It is based on the production system model of computation, which provides pattern-directed control of a problem-solving process and consists of a set of production rules, a working memory and a recognize-act control cycle (the inference cycle). A production rule is a condition-action pair that defines a single chunk of problem-solving knowledge. The condition part of the rule is a pattern that determines when the rule may be applied to a problem instance. The action part defines the associated problem-solving step. Working memory contains the objects that make up the current state of the world in a reasoning process. During the

inference cycle, the patterns in working memory are matched against the conditions of the production rules, producing a subset of production rules called the conflict set, whose conditions match the patterns in working memory. These production rules are said to be "eligible to fire" and one of the productions in the conflict set is selected to fire (that is, the action of the rule is performed, changing the contents of working memory). This selection is based on a conflict resolution strategy which may be simply the first rule that matches, or it may be more complex - based on recency, specificity, or developer-controlled priority. This cycle is repeated until there are no more production rules in the conflict set.

Of the three stages of the inference cycle - matching, conflict resolution & rule firing, the matching process takes approximately 80% of the computational resources. This is because the inference engine must re-evaluate all of the rules each time that there is a change to the data in working memory in order to determine which new rules have become eligible to fire. The common measure of speed of an inference engine is the number of Logical Inferences per Second (LIPS). Unlike the MIPS rating used for sequential processors, the inference engine does not score a point for each instruction that it executes; it scores a point only when it actually fires a rule. As a result, much work has been done on optimizing the pattern-matching process in production systems and the algorithm most widely used for this is called the Rete, developed by C.Forgy at Carnegie-Mellon. He found that most production systems exhibit temporal redundancy - that is, that there are actually very few changes to working memory from one cycle to the next. Accordingly, the Rete algorithm takes as input all of the changes to working memory - the facts that have been deleted, added or modified. Instead of regenerating the entire conflict set, it instead generates a list of changes to the conflict set, thus saving the matching done on one cycle so that it won't be duplicated unnecessarily on the next.

2.3. Business Rules and the Popularization of Rule Engines

Business rules are precise statements that describe, constrain and control the structure, operations and strategies of businesses. They differ from rules coded in a programming language in that they can be understood by non-technical users. Following is an example of a business rule:

```
If Customer traveled to Europe within the last year
```

Then send notification of European travel promotions.

Traditionally, business rules have been buried in application program code, embedded in database structures, or coded as DBMS triggers and stored procedures by programmers, thus making it impossible for non-technical staff to access them. Over the last few years, there has been increasing pressure to provide business users with the ability to create, modify and manage business rules. It is widely recognized that this will enable companies to compete more effectively as new products, pricing schemes and marketing campaigns can be quickly added to business applications by the business experts themselves, without the need for programmer intervention. Additionally, business rules are viewed as an essential means for customizing and personalizing the customer's experience within an automated business transaction, a key capability in today's competitive, impersonal, technology-driven business environment.

A business rule application is an application that contains business rules of a dynamic nature. Business rule applications require the ability to quickly change rules without modifying application code. They require a mechanism for executing rules that integrates easily with common application architectures. They require a high-level business rule language that can be understood by business administrators and/or users, and they require tools like rule editors that support the definition and maintenance of rules in the business rule language.

Rule engines are software components that have been designed to process rules quickly and efficiently. They implement sophisticated algorithms that optimize the use of computing resources for evaluating and executing rules. They also have tools to support the definition, debugging and management of rules. With a rule engine, rules are expressed declaratively in standalone, atomic statements. Thus they are separate from both the control code that determines when and how to execute the rules, as well as from the application code that controls the user interface, the data and transaction management.

This separation provides the ideal application architecture for business rule applications - one that is very easy to maintain. Since the business logic is by far the most dynamic component of the application, the ability to modify it without modifying the rest of the application speeds up development, decreases maintenance costs

and produces very flexible applications. This separation also makes it easy to externalize the business logic in the form of business rules and to provide access to them through tools like rule editors. Using rule editors, business analysts and users can easily modify the business rules as business policies change. And these changes can be made effective immediately within the executing systems, enabling companies to respond quickly to their markets and customers. For these reasons, rule engines are becoming popular as a tool for implementing business rule applications.

3. Part III: Rules & XML

3.1. The Suitability of XML for Language Representation

XML provides an excellent mechanism for describing rule-based languages as its document tree structure is very similar to the parse trees used to represent the grammatical structure of languages. A grammar is a system of rules by which a language can be produced. Grammars are usually expressed as a series of rewrite rules, each one defining the form for a symbol that is important in the language. The grammar for a language is defined by the complete collection of rules.

So for example, the following rewrite rules define a grammar for simple transitive sentences like "The girl likes the ice cream".

```
sentence => noun_phrase verb_phrase noun_phrase =>
noun noun_phrase => article noun verb_phrase =>
verb verb_phrase => verb noun_phrase article =>
a article => the noun => girl noun => ice cream verb => likes
```

The terms that describe the higher-level linguistic concepts (sentence, noun_phrase, etc.) are called *nonterminals*. The terms that describe actual English words form a dictionary of words that may appear in sentences of the grammar. These words are the *terminals* of the grammar. A legal sentence is any string of terminals that can be derived using these rules. In this example, a derivation would begin with the nonterminal "sentence" and would proceed to substitute symbols that match the left-hand side of a rule with the symbols on the right-hand side of the rule. A derivation is often represented as a tree structure, the parse tree, in which each node

is a symbol of the grammar. This diagram depicts the parse tree for the sentence "The girl likes the ice cream".

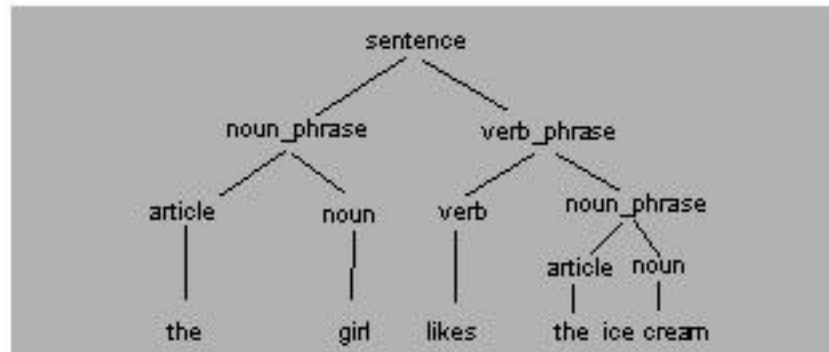


Figure 1. Parse Tree

Elements within an XML document are also structured this way, in a document tree, where the root node is the document element. Markup makes up the intermediate nodes (nonterminals), while character data corresponds to the leaf nodes (terminals).

If we imagine a simple XML DTD for this simple grammar, it might look something like this:

```
<!-- ELEMENT sentence (noun_phrase, verb_phrase)>
<!-- ELEMENT noun_phrase (article, noun)>
<!-- ELEMENT verb_phrase (verb, noun_phrase)>
<!-- ELEMENT article (#PCDATA)>
<!-- ELEMENT noun (#PCDATA)>
<!-- ELEMENT verb(#PCDATA)>
```

And we can imagine the simple sentence represented using the following markup:

```
<sentence> <noun_phrase> <article> the </article>
<noun> girl </noun> </noun_phrase>
<verb_phrase> <verb> likes </verb>
<noun_phrase> <article> the </article>
<noun> ice cream </noun> </noun_phrase> </verb_phrase>
</sentence>
```

As you can see, the XML representation of this simple grammar is quite straightforward and compact, and the expression of sentences using this markup is quite natural. As a result of these advantages, a number of different XML-based rule

language initiatives have arisen.

3.2. XML and Rule Language Initiatives

There have been many projects in both industry and academia, which have attempted to create languages for sharing knowledge. KIF is an early example of this type of language, and although it is not XML-based, it is quite comprehensive and established a solid foundation for describing knowledge in general terms with the goal of sharing. More recent work in this area takes a more practical approach, 1) using XML, because of its ability to make it easy to share vocabularies and knowledge, and either 2) tackling smaller subsets of language primitives, or 3) assuming a specific interpreter on the target platform. This has resulted in a large number of independent, and in some cases, overlapping XML rule language representations.

3.3. Seminal Work - the Knowledge Sharing Effort

The Knowledge Sharing Effort (<http://ksl.stanford.edu/knowledge-sharing>) was an ARPA-sponsored international consortium established to facilitate the sharing and reuse of knowledge. This initiative was very active during the early and mid-90's, and furthered the field significantly through their efforts. KQML, Ontolingua and KIF are among the most noteworthy contributions of this initiative, although their work continues to influence ongoing initiatives in the areas of Agent Communications, Rule Interchange and Knowledge on the Web. KQML (Knowledge Query and Manipulation Language) is a language and a protocol for exchanging information and knowledge between programs. It is both a message format and a message-handling protocol to support run-time knowledge sharing among agents. Ontolingua is a set of tools for developing, maintaining and delivering portable ontologies (shared vocabularies). It includes the KSL Interactive Ontology Server, a web-based service that provides an easy way to create, translate and publish ontologies.

Of specific interest, within the context of this paper, is the work done on KIF - the Knowledge Interchange Format. KIF is an ANSI-proposed language designed for use in the interchange of knowledge among heterogeneous software applications. It attempts to make it possible for applications that have been developed at different times, by different programmers, in different languages to communicate with each

other. It was designed as an interlingua, a mediator in the translation to and from other languages. Based on Noam Chomsky's theory of "Universal Grammar", the notion that the core linguistic structures of our brains are present at birth and are similar across all humans, regardless of native tongue and upbringing. Approaches to language translation that are based on this theory develop a generic intermediate representation for all languages and focus on creating mechanisms for translating between this intermediate representation (the interlingua) and other languages, in order to accomplish language translation.

KIF is a highly expressive logic language with completely declarative semantics, and as such can be understood without the need of an interpreter to manipulate its expressions. This can be contrasted with Prolog, also a logic language, which implements a backward pattern-directed search and requires the application of the unification algorithm in order to search a space of predicate calculus rules and facts. KIF is logically comprehensive, as it provides for the expression of arbitrary sentences in the first-order predicate calculus (unlike Prolog and SQL). KIF provides for the definition of objects, functions and relations. KIF allows for the representation of meta-knowledge, knowledge about knowledge. It also provides for the representation of nonmonotonic reasoning rules, which permits the expression of "reasonable assumptions" in light of uncertain information. The following examples intend to provide a "flavor" of this language.

With KIF, one can express simple data, such as two rows in a personnel database:

```
(salary 017-23-5678 Accounting 52000) (salary 062-41-4556  
Information Systems 74000)
```

More complex information can also be expressed, such as the following sentence which states that one chip is larger than another:

```
(> (* (width chip1) (length chip1)) (* width chip2) (length  
chip2))
```

Since KIF includes logical operators to assist with the specification of negation, disjunction, rules, quantified formulas and other logical information it can be used to construct complex sentences. The following asserts that the number obtained by

raising any real number $?x$ to an even power $?n$ is positive:

```
(=> (and (real-number ?x) (even-number ?n)) (> (expt ?x ?n) 0))
```

Because KIF is such a rich language, it can be difficult to use and can result in the development of systems that are larger and less efficient than they would be if they were developed using a more restricted language. Additionally, it was developed before XML came into widespread use, so there is no XML description of KIF. However, KIF was an important development in this area and has been used as a basis for some of the more recent rule language interchange formats (eg. BRML).

3.4. The Rule Markup Initiative

The Rule Markup Initiative (<http://www.dfki.uni-kl.de/ruleml>) is an international initiative consisting of an open community of researchers, vendors, and end user organizations whose goal is to standardize inference rules on the basis of XML. They hope to create a "RuleML kernel language" to serve as a specification for rule interchange, and to collaborate with participants to establish translations between this language and existing tag sets.

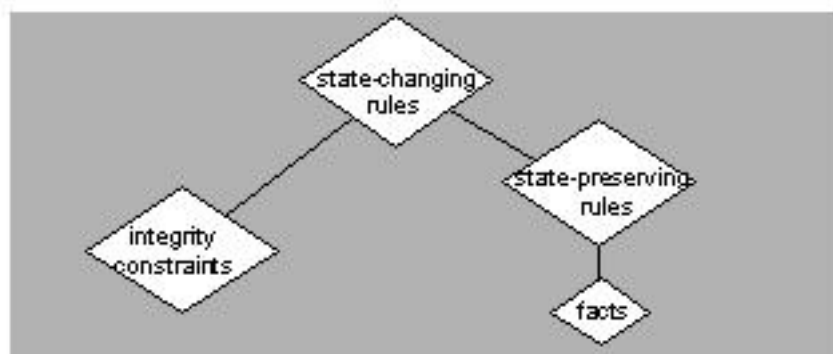


Figure 2. RuleML Rule Hierarchy

The RuleML initiative proposes a rule taxonomy that is hierarchical, with state-changing rules at the root. Integrity constraints and state-preserving rules specialize state-changing rules, while facts are a special type of state-preserving rule (see diagram).

- State-changing rules are considered to be rules that produce side-effects, and include active and trigger rules.
- Integrity Constraints are considered to be special state-changing rules that signal inconsistency when the rule conditions are met.
- State-preserving rules are considered special state-changing rules that assert a conclusion when the rule conditions are met.
- Facts are considered to be special state-preserving rules that have no conditions.

While all of these rules could be implemented as state-changing rules, the RuleML initiative has decided to introduce special-purpose syntaxes tailored to each of these categories. The preliminary markup syntax for the four categories looks like this:

- State-changing Rules:

```
<rule> <and> prem1...premN </and> action </rule>
```

- Integrity Constraints

```
<ic> <and> prem1....premN </and> </ic>
```

Implemented by:

```
<rule> <and> prem1....premN </and> <signal> inconsistency
</signal> </rule>
```

- State-preserving Rules

```
<if> conc <and> prem1 ....premN </and> </if>
```

Implemented by:

```
<rule> <and> prem1....premN </and> <assert> conc </assert>
</rule>
```

- Facts

```
<fact> conc </fact>
```

Implemented by:

```
<if> conc <and> </and> </if>
```

The RuleML Initiative has primarily been working on state-preserving rules and facts and has published a preliminary system of DTDs (current version is 0.7 as of the time of writing). The DTD for the Datalog (Horn clause) subset of RuleML has the following entities and elements:

```
<!-- ENTITY % conc "atom">
<!-- ENTITY % prem "(atom | and)">
<!-- ELEMENT rulebase (if*)>
<!-- ATTLIST rulebase label CDATA #IMPLIED>
<!-- ATTLIST rulebase direction (forward | backward |
bidirectional ) >
<!-- ELEMENT if (%conc;, %prem;)>
<!-- ATTLIST if label CDATA #IMPLIED>
<!-- ELEMENT and (atom*)>
<!-- ELEMENT atom (rel, (ind | var)*)>
<!-- ELEMENT ind (#PCDATA)>
<!-- ELEMENT var (#PCDATA)>
<!-- ELEMENT rel (#PCDATA)>
```

The state-preserving rule "A person owns an object if that person buys the object from a merchant and the person keeps the object" could be expressed using this tagset as:

```
<if> <atom> <rel>own</rel> <var>person</var>
<var>object</var> </atom>
<and> <atom> <rel>buy</rel> <var>person</var>
<var>merchant</var> <var>object</var>
</atom> <atom> <rel>keep</rel> <var>person</var>
<var>object</var> </atom> </and>
</if>
```

The RuleML Initiative plans to continue evolving this hierarchy of languages, and will soon begin to investigate integrity constraints and triggers. The initiative is currently investigating how RuleML can work with RDF. Some of the DTDs on the RuleML site handle URs and can inference with RDF-like facts.

3.5. BRML

IBM CommonRules is a Java library of components for developing & sharing business rules on the Web. It is available free with a trial licence through AlphaWorks (<http://alphaworks.ibm.com>), IBM's channel for making "alpha-code" available to early adopters before it is made available as a fully licensed product, or integrated into other IBM products. CommonRules enables the sharing of business rules across heterogeneous applications by providing a declarative knowledge representation that can be easily translated for execution on different kinds of rule systems. This knowledge representation is based on Logic Programs and captures a core common to different types of rule systems including SQL/RDBMS, Prolog & logic programming systems and production rule systems. CommonRules provides an XML DTD describing this interlingua called BRML (Business Rule Markup Language), and it includes some sample translators that translate between BRML and various types of rule systems.

Of particular interest for the sharing of rules across the Web is the prioritized conflict handling of Courteous Logic Programs, an extension to CommonRule's core knowledge representation. Integrating rules from different sources on the Web presents the possibility of conflicts and inconsistencies between rules when merging and updating rulesets, and can cause incorrect results to be produced during execution. Courteous Logic Programs allow the specification of the scope of potential conflict, via pairwise mutual exclusions called "mutex's". (E.g., one might specify that discounting price by X percent is mutually exclusive with discounting price by Y percent whenever X and Y are not equal). These mutual exclusions are then enforced in the sense that the conclusion set is guaranteed to be consistent with all of the specified mutual exclusions. Courteous logic programs further allow the specification of partially-ordered prioritization information that is naturally available based on relative specificity, recency, and authority. Conflict between rules is resolved using these priorities. Although CommonRules also includes a CLP inference engine, a Courteous Compiler is provided to transform a Courteous Logic Program into a semantically equivalent but expressively simpler ordinary Logic Program of the kind widely implemented in today's commercial rule systems. This way, rules can be translated for execution on these other rule systems.

As the BRML DTD is not freely available, it is not presented in this paper (however, it can easily be reviewed by downloading CommonRules from <http://alphaworks.ibm.com>). Instead, we will look at a couple of sample rules specified using BRML.

Rule:

```
LeadTimeRule1
```

Description:

```
If Buyer is a qualified customer Then Order must be placed
at least 14 days ahead of Requested Delivery Date
```

CLP Syntax:

```
<leadTimeRule1>
orderModificationNotice(?Order,days14) <-
preferredCustomerOf(?Buyer,?Seller)
AND purchaseOrder(?Order,?Buyer,?Seller)
```

Rule:

```
LeadTimeRule2
```

Description:

```
If Ordered item is a minor part Then Order must be placed at
least 30 days ahead of Requested Delivery Date
```

CLP Syntax:

```
<leadTimeRule2> orderModificationNotice(?Order,days30) <-
minorPart(?Order) AND purchaseOrder(?Order,?Buyer,?Seller)
```

Stmt Description:

```
Mutex statements specify the scope of conflict handling via
pairwise mutual exclusions between conditions.
```

Statement:

```
MUTEX_HEAD <- orderModificationNotice(?Order,?X) AND
orderModificationNotice(?Order,?Y) MUTEX_GIVEN
notEquals(?X,?Y)
```

Stmt Description: To resolve conflict between rules, precedence can be specified between rules via an override.

Statement:

```
overrides(leadTimeRule2, leadTimeRule1)
```

Following is some of the BRML that might be used to express these rules and statements:

```
<?xml version="1.0"?>
<clp>
  <clpname>
    <function name="none" />
  </clpname>
  <erule>
    <rulelabel>
      <function name="leadTimeRule1" />
    </rulelabel>
    <head>
      <cliteral>
        <predicate name="orderModificationNotice"
arity="2" />
        <larglist>
          <variable name="Order" />
          <function name="days14" />
        </larglist>
      </cliteral>
    </head>
    <body>
      <and>
        <fcliteral>
          <predicate name="preferredCustomerOf" arity="2" />
          <larglist>
            <variable name="Buyer" />
            <variable name="Seller" />
          </larglist>
        </fcliteral>
      </and>
    </body>
  </erule>
</clp>
```

```

        <predicate name="purchaseOrder" arity="3"/>
        <larglist>
            <variable name="Order"/>
        <variable name="Buyer"/>
            <variable name="Seller"/>
        </larglist>
        </fcliteral>
    </and>
</body>
</erule>
<erule>
    <rulelabel>
        <function name="leadTimeRule2"/>
    </rulelabel>
    <head>
        <cliteral>
            <predicate name="orderModificationNotice"
arity="2"/>
            <larglist>
                <variable name="Order"/>
                <function name="days30"/>
            </larglist>
        </cliteral>
    </head>
    <body>
        <and>
            <fcliteral>
                <predicate name="minorPart" arity="1"/>
            <larglist>
                <variable name="Order"/>
            </larglist>
        </fcliteral>
        <fcliteral>
            <predicate name="purchaseOrder" arity="3"/>
            <larglist>
                <variable name="Order"/>
                <variable name="Buyer"/>
                <variable name="Seller"/>
            </larglist>
        </fcliteral>
    </and>
    </body>
</erule>

```

etc.....

```

<mutex>
    <opposers>
        <ando>
            <cliteral>
                <predicate name="orderModificationNotice"
arity="2"/>

```

```

    <larglist>
      <variable name="Order"/>
      <variable name="X"/>
    </larglist>
  </cliteral>
<cliteral>
  <predicate name="orderModificationNotice" arity="2"/>
    <larglist>
      <variable name="Order"/>
      <variable name="Y"/>
    </larglist>
  </cliteral>
</ando>
</opposers>
<given>
  <fcliteral>
    <predicate name="notEquals" arity="2"/>
    <larglist>
      <variable name="X"/>
      <variable name="Y"/>
    </larglist>
  </fcliteral>
</given>
</mutex>

```

etc.....

```

<erule>
  <rulelabel>
    <function name="emptyLabel"/>
  </rulelabel>
  <head>
    <cliteral>
      <predicate name="overrides" arity="2"/>
      <larglist>
        <function name="leadTimeRule2"/>
        <function name="leadTimeRule1"/>
      </larglist>
    </cliteral>
  </head>
</erule>

```

4. Part IV: J2EE and Java Rule Engines

There are several commercial Java rule engines on the market right now (ILOG JRules, JESS, OPS/J, Blaze Advisor) with different feature sets. They each have their own set of Java APIs, and they each have their own proprietary rule language.

As a result, rules cannot be shared across applications that use different rule engines, and applications developed using these rule engines remain tied to a specific vendor implementation. This is somewhat similar to the situation that existed with RDBMs before JDBC was adopted with its standard API that isolates developers from different vendor RDBMS implementations. JSR-094 has been introduced to accomplish a similar objective with respect to Java rule engines. In order to understand JSR-094, we need to understand a little about how a Java rule engine works. This section presents an overview of rule execution in a Java rule engine.

There is one notable difference between JSR-094 and the JDBC standardization effort: SQL, the content language, had already been adopted as an ANSI standard and implemented within all of the vendor RDBMs. And while there were variations between the different vendor implementations of SQL, these were accommodated through an extension mechanism provided for by JDBC. But the core constructs of SQL were available on all of the vendor RDBMs. Similarly, JSR-094 aims to specify an API but not a content language. The API will provide a mechanism for making the rules available to the engine via XML, but the JSR will not specify the rule language that those rules will be expressed in. This paper proposes a preliminary DTD describing a vendor-independent rule language that can be used with the JSR-094 API. This language was developed based on a common subset of rule language constructs available in the popular Java rule engines. This is possible because these rule engines are forward-chaining rule engines based on the Rete algorithm with extensions specific to the Java language. In most cases, their rule languages evolved from the early production system pattern-matching language implemented in the OPS5 family of expert system shells, and therefore are similar.

4.1. Rule Execution in a Java Rule Engine

To implement an application using a Java rule engine, rules are written in a proprietary vendor rule language and the rule engine is imbedded into the Java application. The rule engine integrates with the application through an API that controls the loading of rules into the rule engine, the monitoring of application objects referenced by rules, and the execution of rules. As shown in the diagram, the rule engine is itself a Java object. The application consists of the application objects and

the rule engine object. The rules are loaded into the rule engine by invoking a method on the rule engine object and passing the rules in a file, stream, XML representation or some other form to the rule engine. In order to evaluate the rules the rule engine must have visibility to the application objects referenced by rules. To provide this type of visibility, the object data members and methods referenced by rules are usually specified as public. The objects referenced by the rules are introduced to the rule engine through the rule engine's assert method. Once the objects have been asserted to the rule engine the rule engine maintains a set of references to the objects. The fire rules method of the rule engine is invoked to cause the rule engine to evaluate all of the loaded rules. When the conditions of rules are met, the rules are executed or fired, causing their action statements to execute. The action statements of rules can modify the rule engines referenced objects thereby causing more rules to become eligible to fire. The process continues until no more rules are eligible to fire at which time control returns to the application statement following the fire rules method invocation.

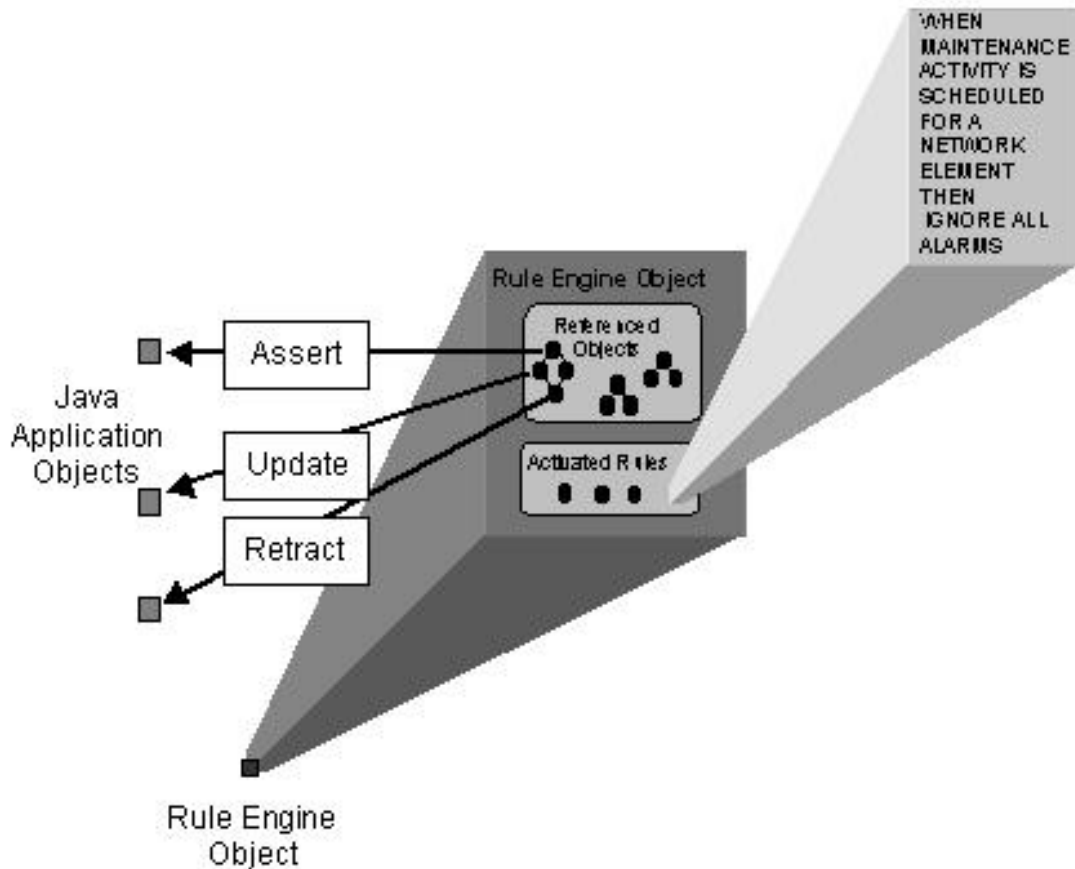


Figure 3. Rules Execution

4.2. JSR-094

The international Java community develops and evolves Java(tm) technology specifications using the Java Community Process (JCP). The JCP produces high-quality specifications in "Internet time" using an inclusive, consensus building approach that produces a specification, a reference implementation (to prove the specification can be implemented), and a technology compatibility kit (a suite of tests, tools, and documentation that is used to test implementations for compliance with the specification). Once a JSR (Java Specification Request) is submitted and approved by the Executive Committee, a draft specification is developed and made available first for review by members of the Java Community, and then by members of the public at large. Based on feedback, the specification is revised and a final draft is

submitted to the Executive Committee for approval before it formally becomes part of the Java technology specifications.

JSR-094 (http://java.sun.com/jcp/jsr/jsr_094_ruleengine.html) defines a Java runtime API for rule engines targeting both the J2EE and J2SE platforms. The API prescribes an object model and a set of fundamental rule engine operations based upon the assumption that most clients will need to be able to execute a basic multi-step rule engine cycle. The fundamental rule engine operations to be supported are: parsing rules, adding objects to an engine, firing rules and getting resultant objects from the engine. The API will also support variations of this basic cycle that would occur in J2EE server deployments (multiple instances of rule engines, the sharing of rulesets between different instances of a rule engine, the ability to reset and reuse an instance of a rule engine, etc.)

A primary input to a rule engine is a collection of rules called a ruleset. The rules in a ruleset are expressed in a rule language. JSR-094 defines API support for parsing rulesets that have been authored in vendor-specific rule languages. Specifically, to help simplify the task of creating rule-authoring tools and because business-to-business exchange of rules is anticipated, the specification defines API support for parsing rulesets that have been authored in XML-based, vendor-independent rule languages. JSR-094 does not attempt to define these rule languages, but assumes that a parallel effort will specify an open, XML-based rule language that will end up in the form of an XML Schema registered at a site such as xml.org.

At the time of writing (February, 2001), the draft specification for JSR-094 is being developed and has not yet been submitted for Community Review.

4.3. An XML Rule Representation for Java Rule Engines

As we saw in the last section, the API proposed by JSR-094 describes the external interface to the rule engine, but says nothing about the rule language itself. It simply prescribes that this "content language" be made available to the engine via XML, and provides some kind of "content handler" to process these XML rule documents. A rule language is still required in order to provide a Java rule engine with rules via this API, and the target rule engine must be able to parse and execute rules specified in

this rule language. This paper proposes a preliminary DTD describing a generic rule language (Simple Rule Markup Language - SRML) consisting of a subset of language constructs common to the popular forward-chaining rule engines. Because it does not use constructs specific to a proprietary vendor language, rules specified using this DTD can be executed on any conforming rule engine, making it useful as an interlingua for rule exchange between Java rule engines.

The ruleset is the root element of the SRML XMLdocument, and it consists of a list of rules. Rules have a condition part and an action part, and the condition part must have at least one condition. Conditions are composed of test expressions, and can be simple conditions or not conditions. Simple conditions can be bound to variables while not conditions cannot. The action part of a rule consists of actions, which can be variable declarations and assignments, as well as the traditional assert, retract and modify statements of rule languages.

The assert action adds an object to working memory. The retract action removes an object from working memory. The modify action modifies an object in working memory. Expressions appear throughout the language and can be assignable (variables or fields), constants (literals such as strings, ints, floats, booleans, etc.), arithmetic or boolean expressions.

Following is a preliminary DTD describing the structure of SRML:

```
<!-- SRML (Simple Rule Markup Language) DTD -->
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT ruleset (rule*)>
<!ATTLIST ruleset name NMTOKEN #IMPLIED>
<!ELEMENT rule (priority?, conditionPart, actionPart ) >
<!ATTLIST rule name NMTOKEN #REQUIRED>
<!ELEMENT priority (%expression;)>
<!ELEMENT conditionPart (%condition;)+>
<!ELEMENT actionPart (%action;)* >
<!ENTITY % condition "(simpleCondition | notCondition)">
<!ENTITY % action "(assignment | bind | assert | assertobj |
modify | retract)">
<!ELEMENT simpleCondition (%expression;)*>
<!ATTLIST simpleCondition className CDATA #REQUIRED
objectVariable NMTOKEN #IMPLIED>
<!ELEMENT notCondition (%expression;)*>
<!ATTLIST notCondition className CDATA #REQUIRED>
<!ELEMENT assert (assignment | bind)* >
<!ATTLIST assert className CDATA #REQUIRED>
<!ELEMENT assertobj (%expression;)>
```

```

<!ELEMENT retract (variable)>
<!ELEMENT modify (variable, (assignment | bind)+)>
<!ENTITY % expression "(%assignable; | constant | unaryExp |
binaryExp | naryExp)">
<!ELEMENT unaryExp (%expression;)>
<!ATTLIST unaryExp operator (plus | minus | not) #REQUIRED>
<!ELEMENT binaryExp (%expression;,%expression;)>
<!ATTLIST binaryExp operator (eq | neq | lt | lte | gt | gte)
#REQUIRED>
<!ELEMENT naryExp (%expression;)+>
<!ATTLIST naryExp operator (add | subtract | multiply |
divide | remainder | and | or) #REQUIRED>
<!ELEMENT assignment (%assignable;,%expression;)>
<!ELEMENT bind (%expression;) >
<!ATTLIST bind name NMTOKEN #REQUIRED>
<!ELEMENT constant EMPTY>
<!ATTLIST constant type (string | boolean | byte | short |
char | long | int | float | double | null) #REQUIRED value
CDATA #REQUIRED>
<!ENTITY % assignable "(variable | field)">
<!ELEMENT variable EMPTY>
<!ATTLIST variable name NMTOKEN #REQUIRED>
<!ELEMENT field (%expression;)?>
<!ATTLIST field name NMTOKEN #REQUIRED>

```

Here is an example of a rule expressed using this syntax:

Rule:

```
Discount
```

Description:

```
If the total purchase amount of a shopping cart is > 100$
Then Set the discount for the shopping cart to 0.1%
```

Traditional Syntax:

```
rule Discount { when { ?s:ShoppingCart(purchaseAmount > 100);
}
then { update ?s { discount = 0.1; } } }
```

Rule Markup:

```
<rule name="Discount">
```

```

<conditionPart>
  <simpleCondition className="ShoppingCart"
objectVariable="s">
    <binaryExp operator="gt"> <field name="purchaseAmount"/>
      <constant type="float" value="100"/>
    </binaryExp>
  </simpleCondition>
</conditionPart>
<actionPart>
  <modify>
    <variable name="s"/>
    <assignment>
      <field name="discount"/>
      <constant type="float" value="0.1"/>
    </assignment>
  </modify>
</actionPart>
</rule>

```

Here is an example of a slightly more complex rule marked up using this syntax:

Rule:

```
CrossSelling
```

Description:

```

If Purchase includes a Videotape in the Shopping Cart
AND The Videotape has a Soundtrack CD
AND The Soundtrack CD is not already in the Shopping Cart
Then Suggest the addition of the Soundtrack CD to the
Shopping Cart

```

Traditional Syntax:

```

rule CrossSelling { when { ?s: ShoppingCart(); ?cd: CD();
?vt: VideoTape(soundtrackCD == ?cd);
Purchase(shoppingCart == ?s; item == ?vt);
not Purchase(shoppingCart == ?s; item == ?cd);
} then { assert Suggestion { shoppingCart = ?s; item = ?cd; }
} }

```

Rule Markup:

```
<rule name="CrossSelling">
```

```

<conditionPart>
  <simpleCondition className="ShoppingCart "
objectVariable="s"/>
  <simpleCondition className="CD" objectVariable="cd"/>
  <simpleCondition className="VideoTape" objectVariable="vt">
    <binaryExp operator="eq">
      <field name="soundtrackCD"/>
      <variable name="cd"/>
    </binaryExp>
  </simpleCondition>
  <simpleCondition className="Purchase">
    <binaryExp operator="eq">
      <field name="shoppingCart"/>
      <variable name="s"/>
    </binaryExp>
    <binaryExp operator="eq">
      <field name="item"/>
      <variable name="vt"/>
    </binaryExp>
  </simpleCondition>
  <notCondition className="Purchase">
    <binaryExp operator="eq">
      <field name="shoppingCart"/>
      <variable name="s"/>
    </binaryExp>
    <binaryExp operator="eq">
      <field name="item"/>
      <variable name="cd"/>
    </binaryExp>
  </notCondition>
</conditionPart>
<actionPart>
  <assert className="Suggestion">
    <assignment>
      <field name="shoppingCart"/>
      <variable name="s"/>
    </assignment>
    <assignment>
      <field name="item"/>
      <variable name="cd"/>
    </assignment>
  </assert>
</actionPart>
</rule>

```

5. Part V: Conclusion

As we have seen throughout this paper, much thought has gone into the creation of an *interlingua* for the exchange of knowledge across disparate systems. Several XML-based rule languages are being developed and proposed to facilitate the

sharing of rules across the Internet. With the acceptance of JSR-094 by the Java Community, we will begin to see these XML-based languages put to practical use. As a result, business logic will become portable, leading to significant reduction in the average time-to-market of software applications, the cost of development and the maintenance workload of application programmers.

Bibliography

[01] Artificial Intelligence: Structures and Strategies for Problem Solving (G.Luger, W.Stubblefield)

[02] Logic, Language and Meaning (L.T.F. Gamut)

[03] <http://alphaworks.ibm.com> - IBM CommonRules

[04] <http://www-ksl.stanford.edu> - Knowledge Sharing Initiative

[05] www.dfki.uni-kl.de/ruleml - the RuleML Initiative

[05] <http://www.blazesoft.com> - Blaze Advisor - Java Rule Engine

[07] <http://herzberg.ca.sandia.gov/jess/> - JESS - Java Rule Engine

[08] <http://www.ilog.com> - JRules - Java Rule Engine

[09] <http://www.pst.com> - OPS/J - Java Rule Engine

[10] http://java.sun.com/jcp/jsr/jsr_094_ruleengine.html - JSR-094 Description

Biography

Margaret Thorpe

Product Manager

ILOG, Inc.

Gentilly

France

Email: mthorpe@ilog.fr

Margaret Thorpe - Margaret Thorpe is currently Product Manager of Business Rule Components at ILOG, a software component vendor based in Paris, France. Previously she was the founder and President of Infinite Intelligence, Inc., a consulting firm specializing in the application of advanced techniques and technologies to the solution of complex business problems. She has over fifteen years of experience designing and developing intelligent systems for manufacturing, sales, finance & systems management. Margaret was an early proponent of "business rules" as a knowledge representation technique and has extensive experience modelling, specifying and implementing business rules for clients in the mortgage, mortgage insurance, and mutual fund industries. She has published a dozen papers and articles on the topics of business rules, database management performance, and expert systems tools.