

The Java Community and Rule Engine Standards

Daniel Selman

Specification Lead, Java Specification Request 94 ("The Java Rule Engine API")

ILOG, Inc.

Johan Majoer

Expert Group Member, Java Specification Request 94 ("The Java Rule Engine API")

Fair Isaac Corporation

Introduction

The number one request from the Java rules community is a standard business rules language (source: <http://www.javarules.org>). Developers view a standard rule language as a key enabling technology, allowing them to build tools and applications that can generate and manage rules, and execute them on multiple rule engines. A standard rule language breaks the dreaded "vendor lock in" and ensures that the considerable investment required to distill knowledge as business rules can be moved between execution environments. It also allows JSR-94, the Java Rule Engine API, to evolve to offer much more interesting services related to the semantics of execution and the internal structure of rules and rulesets.

Underlying the request for a standard business rule language lurks the more difficult issue of semantic interoperability between rule engines. It is clearly not enough to have a standard XML Schema for business rules, and then have the rules execute differently on two rule engines! The difficulty of defining semantic interoperability is compounded by the diversity in rule engine products: ranging from RETE/AI/CLIPS heritage forward-chaining engines, through XML processing engines to decision trees and so forth.

Consequently a major prerequisite for deriving significant value from a standard rule language is resolution of two major open questions: "What is a rule engine?" and "How does a rule engine execute rules?"

The Java rule engine space is still very dynamic, with new vendor products and projects appearing monthly. There are approximately 25 commercial products and 18 open source software projects in the Java rule engine space (source: <http://www.javarules.org>).

Java Community Process JSR-94, the Java Rule Engine API

JSR-94 is a completed standard and defines a number of interfaces that a Java developer can use to interact with a Java rule engine. It does not attempt to define a standard rule language however and hence has a limited practical impact. The specification provides a standard API for the following operations:

- creating a stateful interaction with a rule engine
 - adding and removing Java objects to/from the engine
 - invoking the engine and retrieving result objects
- creating a stateless interaction with a rule engine
 - invoking the rule engine with input objects and retrieving result objects
- deploying an executable set of rules from a variety of sources and registering them for execution

- undeploying rules

A number of vendors currently implement the JSR-94 specification. These include:

- ILOG JRules
- Fair Isaac Blaze Advisor
- Yasutech QuickRules
- Computer Associates, CleverPath Aion
- Drools
- OpenRules
- Sandia Labs, Jess
- Pegasystems PEGARules

A useful parallel can be drawn between JSR-94 and the Java Database Connectivity (JDBC) API. The JDBC API is also comparatively simple and defines just 23 classes/interfaces. The power of JDBC for Java developers comes largely from the underlying standardized SQL and relational model.

Unfortunately there is no equivalent of SQL or the relational model in the rule engine space. This is discussed in greater detail in the section below.

The Opportunity

Business rule modeling, management, exchange and execution clearly has the potential to reach a very broad market, and speaks to a generic business need. For comparison, the world-wide relational and object-relational database management systems (RDBMS) software market is expected to grow to nearly \$20 billion USD by 2006 (source, <http://databasedadvisor.com>). In 2004 the OLAP market alone was estimated at \$4 billion USD (source, www.olapreport.com).

In contrast, the business rules market in 2003 was estimated by IDC to be \$124 million USD.

While the comparison between the data management market (particularly the historical transition from proprietary products to SQL-based products) is enlightening, it is also clear that different market dynamics are at work. For example, the data management market before SQL standardization was considerably larger than the current business rules market.

The Challenges

The biggest challenge to standardization is the lack of a de facto or de jure standard model for rule engines (the equivalent of the relational model for RDBMS). Most of the current major rule engines have a shared heritage in the RETE algorithm and the approach to rule execution championed through the NASA CLIPS project. However, vendors have added many extensions to the basic CLIPS functionality. The pace of vendor innovation is still strong, with new management and runtime concepts arriving regularly.

The rule engines in the Java space are very diverse in functionality and concepts. To summarize, the table below lists some of the functionality in the Drools, Blaze and JRules forward-chaining RETE rule engines. No doubt including other engines, further from the RETE/CLIPS root, would increase the diversity of concepts even further.

A review of some rule engine runtime concepts

The table below is intended to show at a high-level the diversity of execution concepts across three Java

rule engine products: ILOG JRules, Fair Isaac Blaze Advisor and Drools.

Concept	Discussion
Working Memory	Global storage area shared by a group of rules
Agenda Filter	Procedural code that can prevent certain rules from firing
Fact Handle	A persistent identifier for a fact object in working memory
Working Memory Event Listener	Allow programmers to be notified when changes occur to the contents of working memory
Application Data	A mechanism to pass input data and context to rules
Rule	Typically a class that encapsulates the properties below.
- <i>Conditions</i>	The Boolean tests that determine when a rule should be fired
- <i>Declaration</i>	The <i>signature</i> of a rule
- <i>Consequence/Actions</i>	The procedural code that should be invoked when a rule is fired
- <i>Documentation</i>	A human-readable description of the rule
- <i>Duration</i>	The duration or activation-date and expiration-date for the rule
- <i>Name</i>	The name of the rule, often this must be unique
- <i>Salience</i>	A numeric value that determines the order in which the rule will be fired
Task Runner	An API that allows a given collection of rules to be evaluated, within a ruleset
Tool	A callback interface that can be registered with the engine to receive notifications
Function	A procedural function that can be invoked from the conditions or actions of a rule
Ruleset Parameter	The signature of a ruleset, including in, out and inout variable declarations
Task	A logical grouping of rules within a ruleset that perform a business function
Rule Instance	An activated rule that has had variables bound to values
Event	A mechanism to automatically activate rules based on notifications arriving through an event channel
Ruleflow	An orchestration mechanism that controls tasks within a ruleset (or across a ruleset) and implements a procedural control flow
Event rules	Rules that are activated when an event occurs
Collection support	Rule conditions and actions that can reason on collections of objects
Wait/until	Causes rule evaluation to block until a condition becomes true or an event is raised

Concept	Discussion
try/catch and exception handling	Built in rule engine support to handle exceptional error conditions and react appropriately
select/dynamic select	A mechanism to select at runtime which rules should be active based on their properties
priority (static and dynamic)	A mechanism for saliency where priority can be relative or can be calculated at runtime using a function
logical objects (truth maintenance system)	An object that is only present in working memory while a given condition is true
isknown, isunknown	A mechanism that allows rules to determine whether a required variable has been assigned a value
firinglimit	A declarative mechanism to limit the number of rules fired within a given task
Object Model (BOM/XOM)	The model upon which rules are authored (Business Object Model) and the model upon which rules are executed (Executable Object Model). The models may be implemented using the native platform object model (Java, C#) or may be proprietary internal models (named slots etc) .
Sequential Mode, RETE mode	Execution algorithms. RETE provides full forward-chaining based inferencing while sequential evaluates a list of rules against input objects
Decision Tables	A business or execution level artifact that describes a decision table. In practice this may also be executed as a collection of rules
Decision Trees	A business or execution level artifact that describes a decision tree. In practice this may also be executed as a collection of rules

Defining a Standard Evaluation Model

The major challenge to seamless interoperability of rules across vendors is to define a common evaluation model. Such an endeavor should not be underestimated, as the rule engine vendors continue to innovate and new vendors are entering the market.

For comparison it is useful to compare with some other successful specifications:

- Java Virtual Machine Specification (SUN) ~500 pages.
- SQL92: (IBM) ~600 pages
- C Language Standard (ATT) ISO/IEC 9899:1999 ~500 pages

Currently, it would appear that the semantic model with the widest adoption is based on production rules, evaluated using a forward-chaining rule engine. This may be a useful basis for standardization. However, the newer sequential mode is semantically simpler and growing in popularity for some business rules applications. However, not including support for more diverse engines creates the possibility of a standard that may not be broad enough for widespread acceptance.

Appendix - Learning from the SQL Standard

This section is a summary of the paper “*The Essential Paradigm for Successful Information Technology Standards*” by Michael M. Gorman.

Gorman cites three important developments that lead to the success of SQL:

- A significant market share vendor community
- Publicly developed, available, and evolving standards
- Enforced conformance tests

In the software industry there has been no other single class of software that has been as successful as SQL DBMSs. SQL DBMS operates on every hardware platform, through every commonly available operating system, and is managing data of every possible type across industries world wide.

Starting in the early 1970s, independent DBMS vendors arose. Each vendor both specified and implemented data management very differently. Companies or Federal agencies were commonly known as System 2000 shops, Total shops, IDMS shops, or IMS shops. Because of the significant differences among DBMSs, the cost of conversion from one DBMS to another was prohibitive, and the ability to accomplish data sharing was virtually nil.

By 1985 there were at least six different sources for DBMS products all based on the same underlying relational data model. The significant market share vendor community had formed. Because there was such a significant market share vendor community for just one DBMS data model, all other previously significant market share vendors who had supported their own proprietary data model based DBMSs started to wane. In 1984 relational data model DBMSs accounted for less than 20% of the installed licenses. By 1990, they were 80% of the installed licenses. As of 1998, well over 95% of the DBMSs conform to the relational model.

Relational DBMSs operated on every brand of hardware and every major operating system. Because there were multiple DBMSs vendors, quality, performance, features, and competition increased while prices decreased.

At the beginning, the significant market share vendor, IBM, was essential to begin the innovation. In the end, there was a community of significant market share vendors all of whom implemented essentially the same features and facilities. Users were then free to buy from the lowest price, highest quality and performing vendor with the largest quantity of features.

End users were not locked into any one vendor. Databases could be commonly defined, data could be commonly prepared and stored in different vendor databases. Training was able to be accomplished by universities. Today, there is not a single university computer science department that does not teach database management and the relational database computer language, SQL.

During the 1965-1985 era, there were no ANSI data management standards organizations. Consensus documents were however built, but because there were no conformance tests and certifications, vendors implemented these consensus documents their own way. They were like “standards homonyms,” wherein functionally they “sounded” and maybe had verbs that were “spelled” the same way but the functions meant something different at the level that defines portability.

During the second era, 1986 through 1998, IBM’s SQL language was transformed from a de facto standard into a de jure standard. Two critical events supported world-wide acceptance of SQL versus all previous non-relational DBMSs.

- The ANSI data management standard was required to have both a data definition and a data manipulation language component so that it could be both complete and be tested.

- Congress, through public laws such as the Brooks Bill, required that Federal Government agencies procure data management systems only after they were first certified by the National Institute of Standards and Technology (NIST) as conforming to the ANSI standard.

In 1986, DBMSs supported only simple databases that were millions of characters in size. By 1998, DBMSs were supporting advanced data structures, embedded processes, sound, video, and graphics based databases that approached tens of Billions of characters in size.

It was only after 1986 and after the SQL conformance tests were set into place that most Federal agencies began the march to a single class of DBMS. that is, those DBMSs that conformed to the SQL language. Once there were conformance tests, agencies could count on their ability to define databases and data that could be shared. Conformance tests, backed by the "Conform or You Can't Sell" policy *made* all the vendors develop to the SQL standard.

Since 1986, SQL DBMS vendors have rapidly increased. Features have grown almost without measure, and prices have dropped dramatically.

During the mid 1980s through the mid 1990s, the U.S. Government expended considerable sums of money (about \$600K USD per year) to ensure that Federal agencies were procuring standards-conforming products. The practical effect of that procurement rule was to open the procurement process to all those who could “prove” that they conformed to the standard.

Whenever NIST built a test that discovered that different DBMSs produced different results, NIST attempted to informally resolve the semantic discrepancies. For those discrepancies that were not able to be resolved, NIST formally brought the dispute to the standard’s forum for resolution. This NIST role in resolving disputes prior to the “gold” release of data management system products was invaluable to the millions of DBMS end users.

NIST:

- Had a reputation for complete impartiality and independence
- Had the financial backing through which it adequately supported the development of comprehensive batteries of tests
- Had the technical excellence through which it developed a testing scenario that was comprehensive, cost effective, and considered valid and above reproach
- Represented Government consumers, that is, Federal, State, and Local government agencies that hold through procurement regulations that made certification a prerequisite for procurement.