WHITE PAPER

Applying Rule Markup Language in the Military Space Domain

September 2003

Suzette Stoutenburg

MITRE

Air Force Center
Colorado Springs, Colorado

Contents

1. Introduction

1.1. Background

Commercial companies and government organizations have embraced a web based approach to meeting critical business and mission requirements. The new Strategic Technical Plan drafted by the Air Force Electronic Systems Center (ESC) states that "the objective future is one in which systems are made interoperable by adoption of network centric, web-enabling and open architecture technology."[1]

To realize this future vision, it is imperative that exploration of emerging technology continue, with the goal of determining the value and applicability of each new advance to critical government missions. To that end, the Strategic and Nuclear Deterrence Command and Control System Program Office within ESC established the Web Way Ahead effort in 2000. This is a multi-year effort to support research and experimentation with new advances in technology to evolve existing mission systems toward interoperability and network centric processing. The focus to date has been on applications of eXtensible Markup Language (XML)[2] and web-based security.

"The objective future is one in which systems are made **interoperable** by adoption of **network centric**, **web-enabling** and **open architecture** technology."

- ESC, 2001

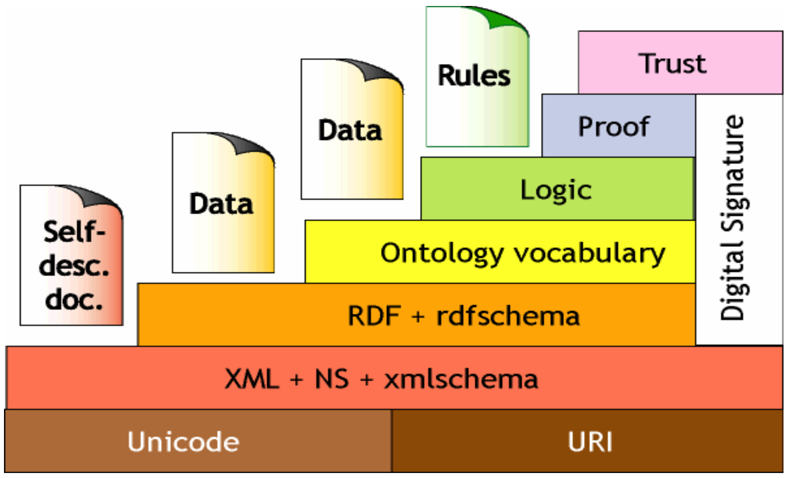
As part of the 2003 Web Way Ahead effort, a study was commissioned to investigate Rule Markup Language (RuleML)[3] to determine its applicability to interoperability in the military space domain. The purpose of this paper is to document the results of this study.

1.2. Why explore Rule Markup Languages?

Rules are fundamental to everyday problems and are especially important in applications that are web-based. Rules specify how a business is run, decisions are made, information is shared, and defense maneuvers are deployed. As XML is embraced as the international standard for web-based data exchange, the need for XML-based rules is becoming increasingly important. A standard for XML-based rules is needed to realize the concept of a Semantic Web[4], and will most likely be an essential component of the future World Wide Web, as shown in the Semantic Web diagram by Tim Berners-Lee, provided in Figure 1[5]. Therefore, understanding how XML-based rules may be applied to help realize the vision for web-based data exchange across the military enterprise is crucial.

Additionally, there is significant business value to be gained by implementing rules. Isolating rules from application code allows changing of rules without software modifications, providing organizations the agility to adapt to changing mission needs. Application code can be reused across different rule domains, and rules can be reused across different application domains. The efficiencies and flexibility to be gained through the use of rules make the Rule Markup Language technology worth further investigation.

[1] Air Force Electronic System Center, 2001 Strategic Technical Plan
[2] XML Home Page, http://www.w3.org/XML/
[3] Rule Markup Language Home Page, http://www.dfki.uni-kl.de/ruleml/
[4] Semantic Web Home Page, http://www.w3.org/2001/sw/
[5] Semantic Web on XML, Tim Berners-Lee, XML 2000, http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html



From Tim Berners-Lee XML 2000 *Semantic Web* talk.
Available at: http://www.w3.org/2000/Talks/1206-xml2k-tbl/

Figure 1
Rules in the Context of the Semantic Web

1.3.What is RuleML?

RuleML is an XML-based markup language that is intended to support rule exchange and interoperation across disparate domains. RuleML allows rules to be expressed as modular components in a declarative way, and uses distinct, standard XML tags to define a rule base, composed of facts and rules.

Rules expressed in RuleML are not intended for direct execution; instead, they can be translated to any inference engine language, such as Java Expert System Shell (Jess)[6], Agent Building and Learning Environment (ABLE) Rule Language (ARL) [7], LISP, or Prolog. Therefore, RuleML is independent of the inference engine used to implement an application, and in fact, is designed to allow exchange of rules between different engines.

The Rule Markup Initiative[8] was a result of the Pacific Rim International Conference on Artificial Intelligence 2000. A team of researchers and practitioners came together and took initial steps toward defining a standard Rule Markup Language based on XML that supports forward and backward chaining as well as inferential, transformational and trigger/reaction oriented tasks.

For more information on the history and details of RuleML, see the RuleML Home Page[9].

1.4. Acknowledgements

This study would not have been possible without the continuing support and vision of Mr. Michael Caracillo, Integration Director of the Strategic Command and Control System Program Office within ESC and Mr. John Shottes, MITRE Executive Program Manager of the Web Way Ahead effort. The RuleML investigation was completed under the outstanding direction of Dr. Nancy Reed and Ms.

[6] Java Expert System Shell (Jess), Sandia National Laboratories, http://herzberg.ca.sandia.gov/jess/
[7] Agent Building and Learning Environment (ABLE), IBM alphaworks, http://www.alphaworks.ibm.com/tech/able
[8] Rule Markup Initiative, http://www.dfki.uni-kl.de/ruleml/
[9] Rule Markup Language Home Page, http://www.dfki.uni-kl.de/ruleml/

Mary Pulvermacher of the MITRE Corporation. Domain expertise and insight was provided by Mr. John Wilson, and outstanding support in the area of eXtensible Stylesheet Language Transform (XSLT) development was provided by Ms. Karen Fox, both of MITRE.

Mr. Said Tabet[10] and Mr. Harold Boley[11], the founders of RuleML, were very supportive and provided guidance on the validity of options to resolve interoperability issues. An XSLT to transform RuleML to Jess[12] was developed by Mr. Tabet, and was made available for our use on the RuleML Home Page. For more information on the work of Mr. Tabet and Mr. Boley, see their personal websites. Example RuleML files provided on the RuleML Home Page were very useful in this study as well.

[10] Mr. Said Tabet Home Page, http://home.comcast.net/~stabet/
[11] Mr. Harold Boley Home Page, http://www.dfki.uni-kl.de/~boley/
[12] RuleML to Jess XSLT, http://www.dfki.uni-kl.de/ruleml/jess/RuleMLTransform.xsl

2. Approach

2.1. Introduction

The purpose of this section is to document the approach taken in the study. The goals and methodology are described, along with the rule domain, inference engines and selected tools.

2.2. Goals

The goals of the ***Applying RuleML in the Military Space Domain*** task are as follows.

- Investigate RuleML technology.
- Determine if RuleML supports interoperability.
- Determine if RuleML will be useful in the military space domain.

The high level plan is shown in Figure 2.

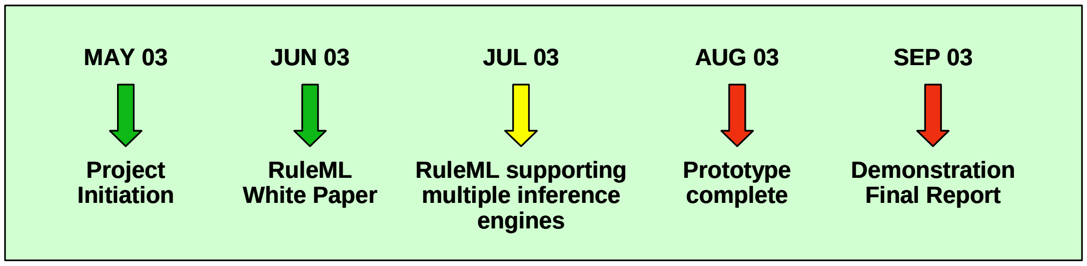


Figure 2
High Level Plan for RuleML Study

2.3. Methodology

The first step in the study was to research what RuleML is and how it is being used today. We studied the RuleML website, including the definition and design of the language. We reviewed examples and experimented with developing rules in RuleML. The result of this investigation was a RuleML White Paper that was delivered as part of this project in May 2003.

The next step was to define an approach that would determine if and how RuleML supports interoperability. To accomplish that, the following steps were identified.

- Identify a set of rules to capture.
- Define the rules in RuleML.
- Select inference engines and languages in which to execute the rules.
- Transform the rules in RuleML to one inference engine language using XSLT.
- Transform the rules in RuleML to a second inference engine language using XSLT.
- Execute the rules against a set of RuleML facts.
- Compare the results to see if the rules in each inference engine result in the same conclusions.
- Automate the process in a demonstration, complete with Rule Distribution objects and web page access to view rules and results. The demonstration is described in more detail later in this section.

The architecture of the overall approach is depicted in Figure 3.

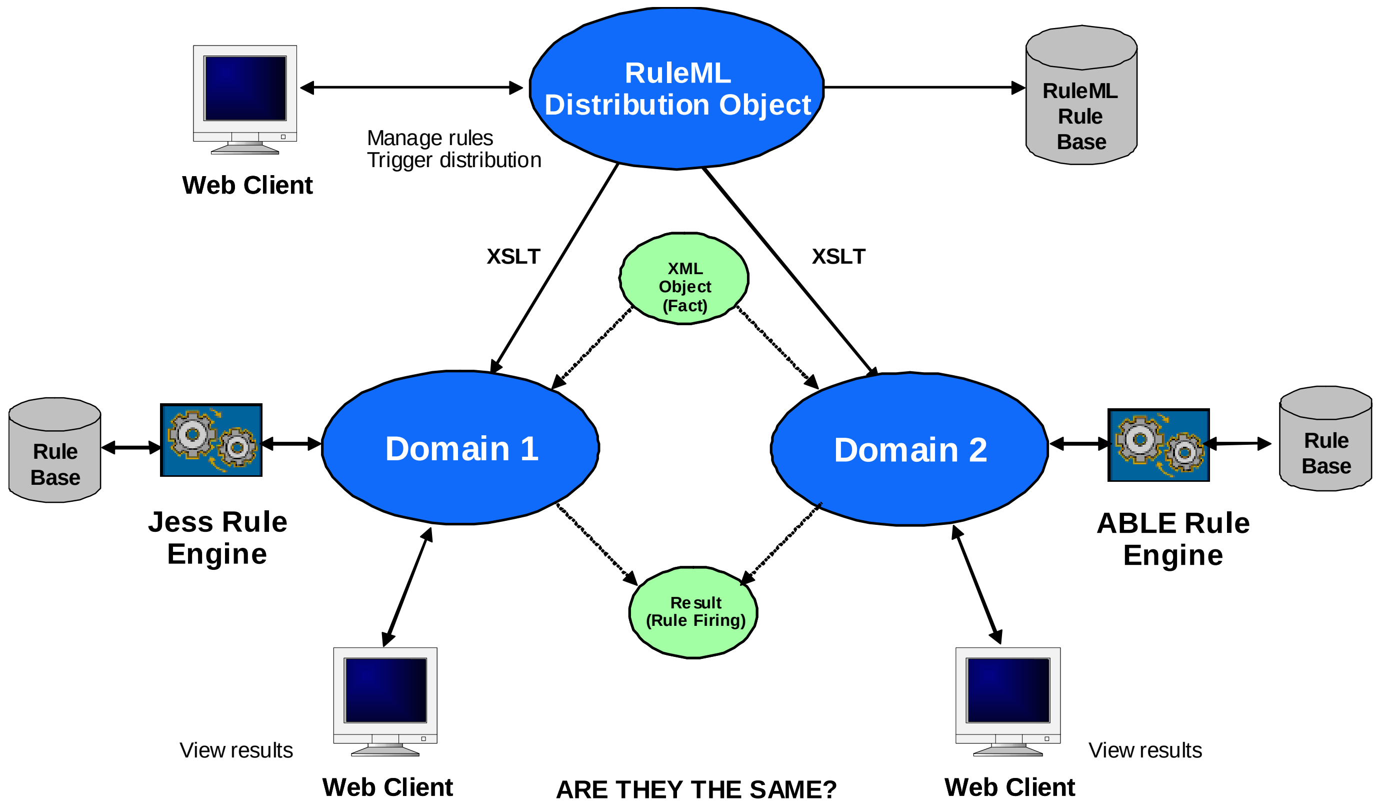


Figure 3
RuleML Interoperability Study Architecture

2.3. Rule Domain

To determine the types of rules to capture, we met with domain experts in the space surveillance arena. The goal was to pick simple rules at first to see if interoperability could be demonstrated without the complication of rule complexity. If successful, we would attempt to capture more complex rules and test the results. We also wanted to keep the demonstration unclassified.

We selected a small set of rules that define how to classify Satellite Element Set messages. These messages are used by the United States Air Force to share information on orbital parameters of all trackable objects that are orbiting the earth. During the course of our research, we found that RuleML could not support the specification of these rules. The details of this finding are covered in section 3.2. We then selected a different set of rules that determine the releasability of Satellite Element Set messages. These are slightly different than determining the classification and are described in more detail in section 3.3.

The rules to classify and determine releasability of Element Set messages are in the form of inference rules, in which a premise is tested, and if true, facts specified in the conclusion become part of the fact base. Inference rules are also referred to as deduction rules or derivation rules. An excellent description of rule types can be found in the following presentation by Mr. Tabet and Mr. Boley: http://www.dfki.uni-kl.de/ruleml/ruleml-krdtd/sld001.htm.

2.4. Inference Engines

The first inference engine selected was the Sandia Laboratories developed Java Expert System Shell (Jess) which uses the CLIPS[13] language. Jess was selected because we were already familiar with it and it easily integrates with Java. CLIPS is a LISP-like language that employs symbolic reasoning in a forward or backward chaining approach.

Next, we wanted to select an inference engine that was very different from Jess. That excluded symbolic reasoning languages, such as Prolog and LISP, since these languages are very similar in structure to CLIPS. We decided on the Agent Building and Learning Environment (ABLE) developed by IBM Alphaworks. ABLE is a framework for developing software agents, employing a Java-like language, Rules Editor and Machine Learning component. ABLE also supports XML through its ABLE Rule Markup Language (ARML).

If it could be shown that rules captured and distributed in RuleML execute uniformly in two very different inference engines, we believed its support of interoperability would be firmly established. If this was not found to be the case, then we planned to explore solutions and/or propose enhancements to achieve interoperability using RuleML.

2.5. Demonstration

To clearly demonstrate the results of the study, we developed a RuleML Interoperability Demonstration. The demonstration is web-based, and allows the user to view RuleML files and trigger the transformation of RuleML to Jess and ABLE languages. Built using Java Server Pages[14] and Java Beans[15], the user can also specify facts dynamically and trigger the execution of rules in each engine. The results from each engine can then be viewed side by side, so that the user can determine if Jess and ABLE executed the RuleML specified rules in a uniform manner.

2.6. Tools

Tools used in this study include XML Spy[16] for RuleML editing, validation and transformation. Xalan version 2.5[17] was used as the transform engine, both through XML Spy and with direct calls from Java beans and batch files.

To transform RuleML to Jess, we made use of an XSLT developed by Mr. Said Tabet[18], which is available on the RuleML Home Page. We created a new XSLT to transform RuleML to ABLE.

Initially, compilation and execution of the rules was accomplished by using the Jess and ABLE rule engines directly. Later, we automated the process in the RuleML Interoperability Demonstration, described above. Apache Tomcat[19] was used as the web server in support of the demonstration.

RuleML version 0.8[20] was evaluated in this study.

[13] CLIPS Home Page, http://www.ghg.net/clips/CLIPS.html
[14] Java Server Pages, http://java.sun.com/products/jsp/
[15] Java Beans, http://java.sun.com/products/javabeans/
[16] XML Spy Home Page, http:/www.xmlspy.com/
[17] Xalan Home Page, http://xml.apache.org/xalan-j/
[18] RuleML to Jess XSLT, http://www.dfki.uni-kl.de/ruleml/jess/RuleMLTransform.xsl
[19] Apache Tomcat Home Page, http://jakarta.apache.org/tomcat/
[20] RuleML 0.8 Schema, http://www.dfki.uni-kl.de/ruleml/inxsd0.8.html

3. Results

3.1. Introduction

Several iterations were necessary before a single RuleML file could be successfully executed in Jess and ABLE. The purpose of this section is to describe the major iterations, including how the rules were built, translated and interpreted by each engine.

3.2. Iteration 1 Element Set Message Classification Rules

Rules for classifying Element Set Messages were easily captured in RuleML. These rules primarily involved checking the range of values of a field in the message. These rules were successfully transformed to Jess, but did not fire correctly. We found that the "<" and ">" operators were treated as symbols in Jess, not as mathematical operators.

Next, we looked for a way to successfully express comparison of values in Jess, before attempting to capture the rules in RuleML. We built a set of rules using Jess test patterns. But, we were unable to find a way to represent test patterns in RuleML. Using templates in Jess is a good way to represent message data, but we abandoned that as well, once we determined that RuleML 0.8 did not support Jess templates.

We were unable to find a solution in which comparison rules could be captured in RuleML and then transformed and successfully executed in Jess. Therefore, we decided to change to a rule set that would be easier to represent in a markup language that is declarative, supports symbolic reasoning, and does not require mathematical operators. We selected a rule set that notionally specifies Space Message Releasability rules to move information between multi-level security domains.

3.3. Iteration 2 Space Message Releasability Rules

Rules for classifying Space Message Releasability were simplified for use in this study, and were easily captured in RuleML. Rules were structured around the following variables:

- **Source** of each Satellite Element Set (elset) message, captured in the **source** variable
- **Destination** of each elset message, captured in the **destination** variable
- **Classification** of each elset message, captured in the **elsetMessage** variable

Based on the value of each of these, the **releasability** of each message can be determined.

Therefore, the rules primarily involved checking values of variables. These rules were successfully transformed to Jess, and executed properly.

Since we determined in iteration 1 that the contents of a message composed of fields could not be captured easily in RuleML, we decided to change the use of the **elsetMessage** variable. In iteration 2, the **elsetMessage** variable (marked with the RuleML tag <var>) was used to capture whether the message was "classified" or "unclassified". We assumed that a Java object would manipulate each incoming message and send the information (fact) in the proper format to the inference engine. The **source** variable was used to identify the source of the message as being "trusted" or "untrusted". The **destination** variable captured the security level of the domain to receive the message. Based on all these variables, **releasability** of the message could be decided.

The releasability of the message was represented as an individual value (marked with RuleML tag <ind>) that would be paired with the value true or false. Releasability was not represented as a variable in this iteration, but instead more as a descriptor of the meaning of the fact. This approach to designing the rules closely followed the example in the Discount Rule Base[21], provided from the RuleML Home Page.

Note also that initially, the **elsetMessage** and **destination** variables had some overlap in the potential range of values, such as the value of "unclassified". This resulted in undesired rule firings in which rules about destinations fired on facts about elset messages, and rules about elset messages fired on facts about destinations. Therefore, we found it necessary to include amplifying information around each atom in the premise to avoid undesirable rule firings. So, the individual "message" (**<ind>message</ind>)** was added in the rule premises with every occurrence of the **elsetMessage** variable to indicate that it was a message. These <ind> tags added meaning to each premise, because when translated to Jess, each would result in an additional atom that would need to be matched in the symbolic reasoning engine. This successfully avoided the undesired rule firings observed early in iteration 2.

Next, the new rule set was transformed to ABLE. However, we found that the resulting code would not execute in ABLE because ABLE did not recognize the translated <ind> tags. We considered how we might modify the XSLT, but postponed that approach, thinking we might be able (and perhaps should) develop rules that don't involve <ind> tags that are used as descriptors. The use of <ind> tags as descriptors will only work in a symbolic reasoning engine. Therefore, we redesigned the RuleML representation of the Releasability rules to avoid use of descriptor <ind> tags in iteration 3.

3.4. Iteration 3 Modified Releasability Rules

The Space Message Releasability rules were modified in iteration 3 to be simpler. The clarifying <ind> tags were removed, and to avoid undesirable rule firings, the range of values for the elsetMessage and destination variables were changed to be exclusive. Releasability was no longer represented as a descriptor (<ind>) to the elsetMessage variable; instead **releasability** was used as the name of the variable that would represent the value of "true" or "false", indicating the results of the rule firings.

These rules were transformed to ABLE and successfully executed.

Iteration 3 rules were transformed to Jess, but did not work properly in Jess. Instead, execution of the transformed rules caused an exception to be thrown. Upon inspection of the Jess rules, it was clear that the variable **releasability**, as represented in the rules, had no prior value or binding in the premise. Therefore, when the rule fires, there is no value that can inferred.

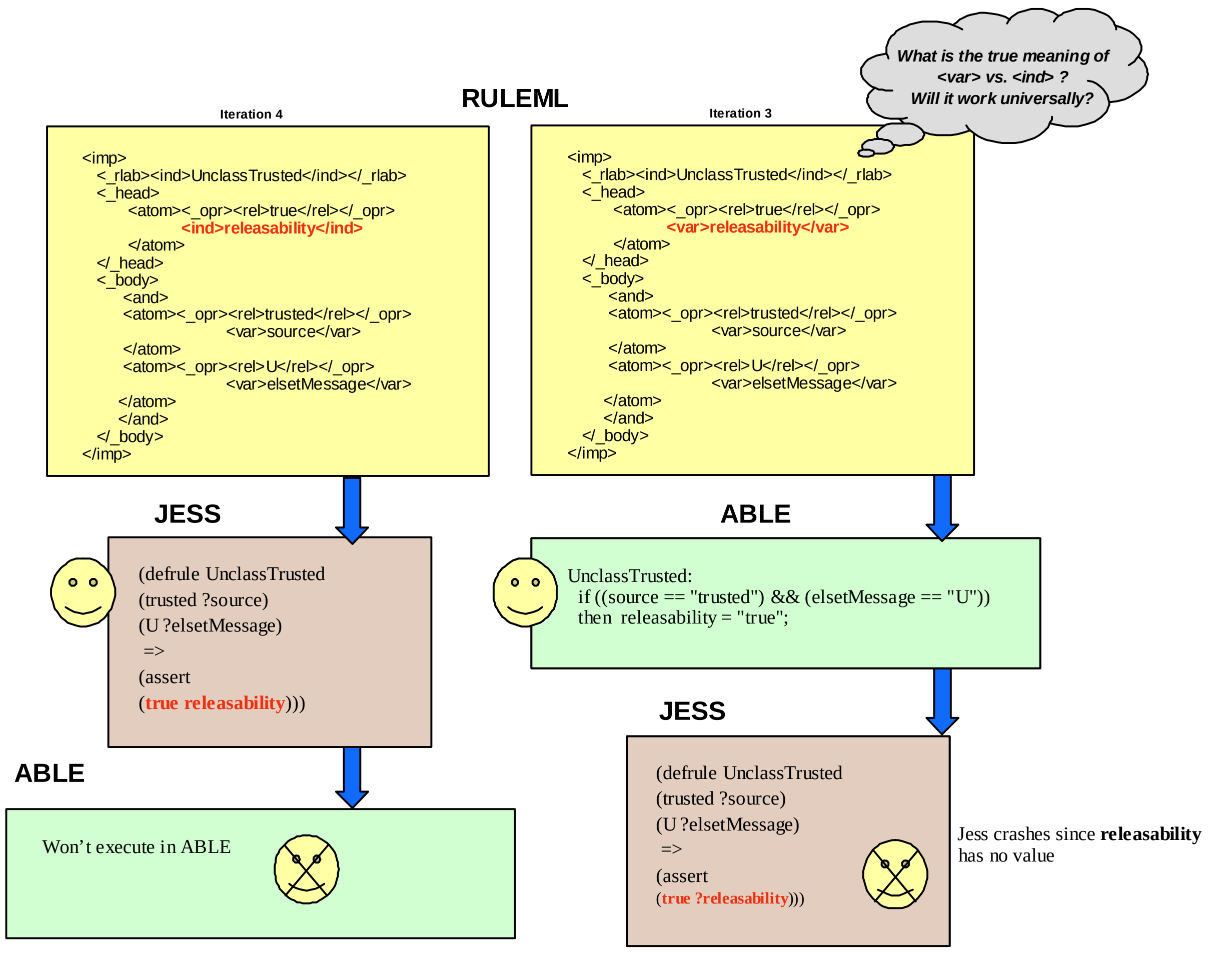
Note that straightforward assignment of values to variables is possible in Jess using binding. However, we found that RuleML does not support binding of variables.

3.5. Iteration 4 Releasability Rules with modified Releasability variable

In iteration 4, the Space Message Releasability rules were modified so that the **releasability** variable was represented with an <ind> tag. As expected, transformed Jess rules executed properly but transformed ABLE rules did not.

The fundamental interoperability problem found here is in the use of variables between reasoning engines. In CLIPS and other symbolic reasoning languages, variables are placeholders that take on different values that are then pattern matched against facts. The variable name is replaced with its value during pattern matching, and is not used to reference the value after matching. In ABLE, variables are Java-like, and can only be referenced by name. This problem is depicted in Figure 4.

[21] Discount Rule Base in RuleML, http://www.dfki.uni-kl.de/ruleml/exa/0.8/discount.ruleml


RULEML
Iteration 4
Iteration 3
What is the true meaning of <var> vs. <ind> ? Will it work universally?
JESS
ABLE
JESS
ABLE
Won't execute in ABLE
Jess crashes since **releasability** has no value

```
<imp>
  <_rlab><ind>UnclassTrusted</ind></_rlab>
  <_head>
      <atom><_opr><rel>true</rel></_opr>
          <ind>releasability</ind>
      </atom>
  </_head>
  <_body>
     <and>
     <atom><_opr><rel>trusted</rel></_opr>
                 <var>source</var>
     </atom>
     <atom><_opr><rel>U</rel></_opr>
                 <var>elsetMessage</var>
    </atom>
    </and>
  </_body>
</imp>
```

```
<imp>
  <_rlab><ind>UnclassTrusted</ind></_rlab>
  <_head>
      <atom><_opr><rel>true</rel></_opr>
          <var>releasability</var>
      </atom>
  </_head>
  <_body>
     <and>
     <atom><_opr><rel>trusted</rel></_opr>
                 <var>source</var>
     </atom>
     <atom><_opr><rel>U</rel></_opr>
                 <var>elsetMessage</var>
    </atom>
    </and>
  </_body>
</imp>
```

```
(defrule UnclassTrusted
(trusted ?source)
(U ?elsetMessage)
 =>
(assert
(true releasability)))
```

```
UnclassTrusted:
  if ((source == "trusted") && (elsetMessage == "U"))
  then  releasability = "true";
```

```
(defrule UnclassTrusted
(trusted ?source)
(U ?elsetMessage)
 =>
(assert
(true ?releasability)))
```

Figure 4
Fundamental Interoperability Problem

We modified the ABLE XSLT to translate <ind> tagged atoms to variables in ABLE. This did work for our rule set, and allowed the RuleML generated in Iteration 4 to work in both Jess and ABLE. However, we showed that this does not work in general. We downloaded the Discount Rule Base from the RuleML Home Page and found that ABLE did not translate properly, since the use of the <ind> tagged atoms as descriptors only work in symbolic reasoning engines.

A summary of all iterations along with findings is provided in Table 1.

Table 1
Summary of Iterations and Results

Iteration	Rule Set	Finding
Iteration 1	Rules to classify Element Set messages in RuleML.	RuleML treats mathematical operators like symbols, making mathematical operations difficult. Decided to change to a rule set that better fits symbolic reasoning.
Iteration 2	Rules for releasability of Element Set messages. Added <ind> tags to capture semantics around variables.	Sharing range of values across Jess variables can lead to rule misfires. Worked in Jess, but not in ABLE. <ind> tags were not easily translated to ABLE.
Iteration 3	Removed semantic <ind> tags and changed range of values for each variable to be distinct. Treated "releasability" like a variable, using <var> tag.	Worked in ABLE but not in Jess. Concept of variables are fundamentally different in Jess (pattern-matching) vs. ABLE (Java-like)
Iteration 4	Changed releasability back to <ind>. Re-evaluated the meaning of <ind>. Modified ABLE XSLT to translate <ind> to variables.	Worked in Jess, not in ABLE. Worked in Jess and ABLE for this rule set. Tested with additional rule sets from web site, but they did not transform properly in ABLE. True interoperability was violated.

3.6. Summary of Findings

The following points summarize our key findings.

- RuleML supports symbolic reasoning engines very well.
- RuleML treats mathematical operators like symbols, making mathematical operations difficult. We were unable to express rules using ">" or "<" operators in RuleML that execute properly in the inference engines.
- ***Getting one RuleML file to execute in multiple languages is difficult, due to differences in fundamental processing between symbolic reasoning and Java-like languages.***
    - ***The use of variables and "individual" values in RuleML can result in non-uniform execution of rules.***

Other findings include the following.

- RuleML 0.8 does not support NOT.
- RuleML 0.8 does not support templates.
- RuleML does not use XML attributes widely, while ABLE Rule Markup Language relies heavily on attributes.
- ABLE requires that variables are declared up front, while RuleML has no concept of variable declaration.
- RuleML does not support binding of variables or retracting facts inside a rule premise.

We also found that building rules that fire properly is tricky in any inference engine. Rules must be designed so that data can be captured as facts that trigger the firing of appropriate rules. In the absence of robust tools, developers with expertise in rule design and development will be required to develop viable rule bases.

3.7. Solution Options

One option to resolve the interoperability problem between Jess and ABLE would be to modify the RuleML to ABLE XSLT to translate <ind> tagged items to variables in ABLE. This did result in interoperable rule sets in some cases, but did not work universally, as observed in Iteration 4 results.

Another possible solution would be to add tags to RuleML that allow specification of the different uses of <ind> and <var>. For example, <ind> can be used in two different ways: as a descriptor in a premise or as a literal value that holds a value of a variable. Here is an example of using <ind> tags as a descriptor, taken from RuleML developed as part of this study.

```
<atom>
   <_opr><rel>true</rel></_opr>
   <ind>releasability</ind>
   <var>elsetMessage</var>
</atom>
```

Here is example RuleML, showing how <ind> can be used to contain a literal value that represents the value of a variable. This example is taken from the RuleML Home Page.

```
 <atom>
   <_opr><rel>discount</rel></_opr>
   <var>customer</var>
   <var>product</var>
   <ind>"5.0 percent"</ind>
</atom>
```

Perhaps these two uses should be distinguished with different tags or different attributes, then translated accordingly to the destination language. This approach warrants further investigation.

Rules could be designed so that the use of <ind> and <var> is not an issue, such as what was done in Iteration 4. This is not ideal, but will probably be necessary to some extent if Java-like inference engines are to be supported by RuleML. Robust tools could be developed to hide the design details of RuleML and enforce standards for rule design. A novice rule developer could use a GUI to specify rules in a simple way, then the design of the actual marked up rules could be created behind the scenes such that no confusion occurs between symbolic reasoning and Java-like inference engines. If these approaches are not found to be viable, it may be necessary to accept that RuleML, in its current state, supports symbolic reasoning engines only.

Finally, we hope to investigate whether an ontology could be implemented along with RuleML rule bases to clarify any differences in translation to inference languages.

The proposed solutions to the interoperability problems observed are provided in Table 2.

Table 2
Summary of Solution Options

Solution	Result	Advantages/ Disadvantages	Conclusion
Develop custom XSLTs for each inference engine domain.	This was successfully demonstrated in Iteration 4. However, this did not work in general, as other RuleML rule bases downloaded from the internet would not execute properly.	Could lead to different interpretation of rules. May not solve the problem in general.	While writing custom XSLTs will be required for each inference engine type, writing custom XSLTs within the same language domain should be avoided. Overall rating: POOR
Add tags to be able to interpret <ind> vs. <var> properly; distinguish between multiple uses of <ind> using specific tags.	Needs further evaluation and collaboration with RuleML founders.	Could violate the declarative nature of RuleML.	Overall rating: TBD
Hide RuleML details with GUI tools; tools manage necessary differences in RuleML to fit domain, depending on rule engine.	Needs further evaluation and collaboration with RuleML founders.	Allows one specification of rules, with details hidden. Could violate the declarative nature of RuleML.	Overall rating: Promising
Write rules carefully so that the <ind> and <var> interpretation is not an issue.	Under investigation.	Rule development should not be limited by the inference engine language.	Overall rating: Not ideal, but probably necessary if RuleML will support Java-like inference engines.
Acknowledge that RuleML supports symbolic reasoning languages and not Java-like languages.	Needs further evaluation and collaboration with RuleML founders.	Limited interoperability of RuleML. Can symbolic reasoning engines perform under high load?	Overall rating: Not ideal, but again, potentially realistic.
Investigate use of ontology to eliminate confusion.	Under investigation.	Would allow RuleML to remain independent of inference engine.	Overall rating: Promising

4. Conclusions

We fully support the notion of capturing and sharing rules in an XML-based language. In general, we found RuleML to be an excellent start at defining a standard for which to specify XML-based rules. RuleML clearly supports symbolic reasoning engines very well. It is important to remember that this standard is in its infancy, and extensions and a new version are under development.

Building rules that fire properly is not a simple task in any inference engine. Rules must be designed so that data can be captured as facts that result in the triggering of appropriate rules. There is a need to develop robust tools to support the design and implementation of domain rules. Ideally, these tools should hide the details of the rule base implementation as much as possible, especially if wide use of an XML-based rule language is to become a reality.

We believe that a set of standards for rule design should be developed and published along with the RuleML documentation. Tools to support RuleML development should enforce the standards.

RuleML will have to be extended in some way to allow for use of mathematical operators since this is a clear need in any commercial or government application. This could be accomplished with new RuleML tags. RuleML 0.8 does not support NOT, which will also be needed in the future. During collaboration with the RuleML founders, Mr. Said Tabet and Mr. Harold Boley, we found that much of this work is already underway. There is also an extension to RuleML, called Object Oriented RuleML[22], which does support message templates.

There are features of certain languages that are not supported by RuleML, such as the binding of variables in the rule premise which is widely used in Jess. According to the founders, this omission is by design, since they want to preserve the declarative nature of RuleML. We don't see this as a weakness, but it will require programmers with expertise in a given language to think differently when designing and implementing rules in RuleML.

The biggest challenge we identified is that the use of variables and "individual" values in RuleML can result in non-uniform execution of rules. This problem is due to the fact that variables can be fundamentally different between some inference engines, and that <ind> tagged atoms can be used in different ways. We plan to continue our collaboration with the RuleML founders to develop solutions to this problem.

[22] Object Oriented RuleML, http://www.dfki.uni-kl.de/ruleml/indoo/

5. Recommendations

We are seeking support for MITRE participation in the continuing work to develop a standard Rule Markup Language. The solutions proposed in section 4 should be investigated further with the RuleML founders so that modifications and extensions to RuleML can be defined.

Continued collaboration with RuleML Founders on solutions and tools.

It is clear that a robust set of tools will be needed to support rule development in RuleML. Standard XSLTs will be needed for use with each inference engine language. Establishing tools and standards will be important as RuleML development continues. MITRE support in the evaluation of these tools could benefit the overall effort to standardize rule development.

We also are seeking support for the following follow-on studies.

Introduce a **web service language** as a third domain.

We would like to introduce a **web service language**, such as Water, as a third domain in the study. We would test to see how rules are translated from RuleML to Water and determine if there are any new issues discovered with a language that is very different than Jess and the Java-like ABLE.

We believe that **more complex** and other types of rules, such as **Reaction Rules,** should be implemented and tested. This effort could uncover new issues not discovered with the simple deduction rules used in this study. We would like to investigate **other rule markup languages** to see if there are best practices that could be applied to RuleML.

The **performance of rule engines** will be very important in the Semantic Web of the future. We recommend a performance study in which a number of inference engines are evaluated to determine which is the most viable under different conditions encountered in the military domain. In particular, which engine performs best under high load scenarios? Would the findings of such a study drive what the canonical rule markup language should be?

Apply additional **Semantic Web technologies**

We recommend that an **ontology** be developed to capture the differences in the use of variables and other components of RuleML between engines, as shown in Figure 5. We could then test to see if the ontology can mitigate the interoperability problem discovered in this study. Also, an ontology to describe the data between the tags should be developed in conjunction with a RuleML rule base to **simulate a Semantic Web** in a military domain. The lessons uncovered in such a study could be widely applied to today's military systems under development.

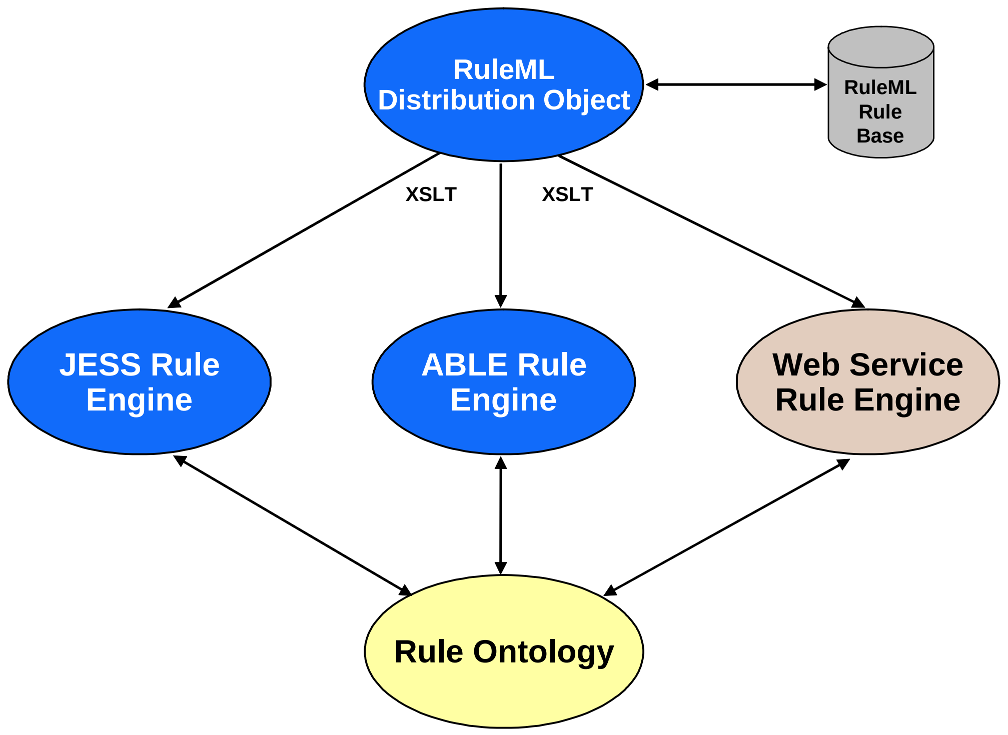


Figure 5
Applying Additional Semantic Web Technologies

6. References

1. **RuleML Home Page**
   http://www.dfki.uni-kl.de/ruleml/
2. **Object Oriented RuleML**
   http://www.dfki.uni-kl.de/ruleml/indoo/
3. **Jess Home Page**
   http://herzberg.ca.sandia.gov/jess/
4. **Agent Building and Learning Environment**
   http://www.alphaworks.ibm.com/tech/able

7. Acronyms

ABLE	Agent Building and Learning Environment
API	Application Programming Interface
ARML	ABLE Rule Markup Language
Elset	Element Set
ESC	Electronic Systems Center
GSIX	Guarded Sharing of Information using XML
GUI	Graphical User Interface
Jess	Java Expert System Shell
NS	Namespace
PKI	Primary Key Indicator
RDF	Resource Description Framework
RuleML	Rule Markup Language
URI	Universal Resource Indicator
XML	eXtensible Markup Language
XSLT	eXtensible Stylesheet Language Transform