

Carnegie Mellon
Master of Information Systems Management

Rules Engine
Independent Study
Fall 2003

Date: December, 17 2003

Prepared by
Punit Kaishap

Instructor
Prof. Eric Nyberg

Table of Contents

1 ABSTRACT	2
2 INTRODUCTION.....	3
2.1 Rules.....	3
2.2 Rules Engine	4
2.3 Features of a Rules Engine.....	5
3 RULES ENGINE MARKET.....	7
4 PROTOTYPES	10
4.1 Prototype 1	10
4.2 Prototype 2	13
5 CONCLUSION.....	17
6 SOURCES.....	19

1 Abstract

I have drawn inspiration for this paper from my summer internship with U.S. Steel. I worked in a team that developed a Java based Rules Engine used to validate orders placed at U.S. Steel plants worldwide.

Although this paper is about Rules Engines I have concentrated on the application of Rules Engines as a technology solution for automating business processes. Thus, you will find that I have made a number of references to a Business Rules Engine in this paper.

I start the paper by providing an introduction to Rules Engines and its components. The next section of this paper concentrates on the Rules Engine market. In addition I have compared two Rules Engines based on factors like rule lifecycle management, flexible deployment, performance and extensibility.

A very important part of this paper is my work regarding two prototypes. The first prototype shows how XSL and XML can be used to develop a simple Rules Engine. The second prototype consists of a database model and a design document. Both these prototypes highlight the benefits of developing a rules based system using an event condition paradigm.

I conclude the paper by stating some key points that explain what I have learned about Rules Engines and have tried to state what its future may be like in the technology powered business world of tomorrow.

2 Introduction

2.1 Rules

Many enterprise software systems today already include the concept of rules. Most times, these rules are directly implemented in code and are difficult to adapt to a changing business landscape. When these business 'rules' are coded using normal programming techniques, modification and maintenance of the logic can become difficult.

Example of simple business rules:

When a customer of a bank applies for a loan he must have a saving account with a balance exceeding \$5,000 otherwise **reject** the loan application.

In a steel company a customer can only place an order of a minimum of 100 steel coils. If an order contains less than the minimum quantity **cancel** the order.

Example of complex business rules:

When someone is selling a product desired by another person for a price less than or equal to the price the other is willing to pay **then** notify both the parties.

When the interest rates in the United States fluctuate check the interest rate fluctuation in the Asian continent.

Three scenarios:

- If the fluctuations balance each other out in the long term **then keep** investments levels in both the continents the same.
- If the fluctuations in the long run are in favor of United States **then reduce** investments in the Asian continent and increase investments in the United States.
- If the fluctuations in the long run are in favor of the Asian continent **then reduce** investments in the United States and increase investments in the Asian continent.

Long term is defined as a period of 5 years.

All the above rules contain an event that is followed by an action. Such a rule structure is used in systems that need to respond to changes in policy as quickly as the enterprise makes decisions.

Rule types

1. Declarative Rules

A set of rules that compute values or enforce constraints as other properties change, ensuring that data remains consistent

2. Decision Tree Rules

A family of rules that conduct fact-based forward chaining to execute "if-then" types of logic

3. Process Rules

These rules manage the receiving, assignment, routing and tracking of work

Benefits of Business Rules

- Business rules are easy to change

A business user can simply add, modify or delete rules without re-thinking program logic or sequence of execution

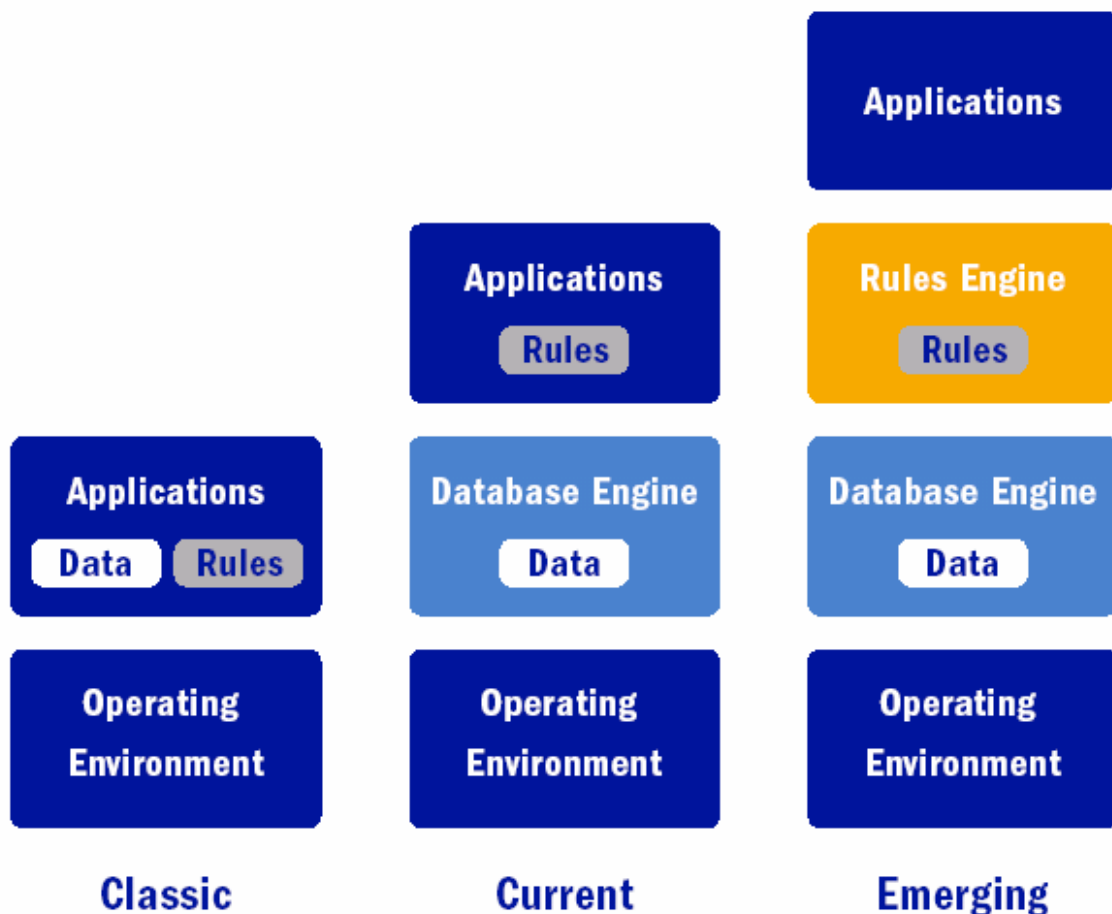
- Business rules are easy to understand
Business rules are normally represented in an easy to understand form. If rules are integrated with a Business Process Management solution then rules are normally represented as part of a Graphical User Interface.
- Business rules can manage complexity
In procedural coding, the programmer is responsible for correct logic flow which normally limits the complexity. On the other hand a Rules Engine takes the responsibility for computing the right action for a set of conditions, with no limits on complexity.

2.2 Rules Engine

In traditional applications, business rules and procedures were encoded in the source code of an application. However, this approach has two problems:

- a) It is expensive and slow to change as the business changes; and
- b) It is hard for the business user to represent business processes in the form of program logic / code.

Business Rule Engines take a very different approach. The rule engine is an algorithm that ensures that for any set of facts, all applicable rules are applied efficiently and consistently to compute the correct action according to the rules. The business users can create and edit the rules any time business changes.



(Source:

http://www.pegacom/ProductsServices/EnterpriseBPMTechnology/prpc/prpc_overview/documents/BREBrochureFINAL.pdf)

The above diagram depicts the Rules Engine as a component that powers applications by acting on data based on events received. The important point is that a Rules Engine solution is especially useful in system integration where the existing enterprise applications do not have to change.

2.3 Features of a Rules Engine

This section will help the reader understand the underlying concepts that form a Rules Engine.

Rule Centric Approach

A good rules engine allows the business logic of a system to be specified external to the system itself. Without a rules-based approach, rule code could be potentially hard coded and possibly replicated throughout an application within if/then blocks. Such an approach could cause the code not to be reused or maintained easily.

Knowledge base

A rules-based system maintains a collection of facts. This collection is known as the knowledge base. It somewhat has the same representation as a relational database, especially in that the facts must have a specific structure. Simply, a knowledge base is data that represents the state of an evolving situation.

Rules base

Rules represent the heuristic knowledge of a human expert in some domain. A rules base states how certain categories or facts are to be connected with other categories of facts.

Forward chaining

Depending on the algorithm used, some Rules Engine implementations use the concept of forward chaining. For example, the central concept of Jess is forward chaining.

Forward chaining is defined as a data driven technique used in constructing goals or reaching inferences derived from a set of facts. In addition, forward chaining is normally only possible when the number of outcomes is small. When there are too many conclusions, forward chaining becomes too inefficient and time consuming to be of any use.

Example:

Consider the following set of rules:

Rule 1: If A and C	Then F
Rule 2: If A and E	Then G
Rule 3: If B	Then E
Rule 4: If G	Then D

Suppose you need to prove that D is true given that A and B are true. The first iteration fires Rule 3 and determines that A, B and E are true. The second iteration fires Rule 2 (A, B, E, and G are true) and then Rule 4 (A, B, E, G and D).

This is the method of forward chaining, where one proceeds from a given situation toward a desired goal, adding new assertions along the way.

Backward chaining

Again, depending on the algorithm used, some Rules Engine implementations use the concept of backward chaining.

Backward chaining is a method where one starts with the desired goal, and then attempts to find evidence for proving the goal.

Example:

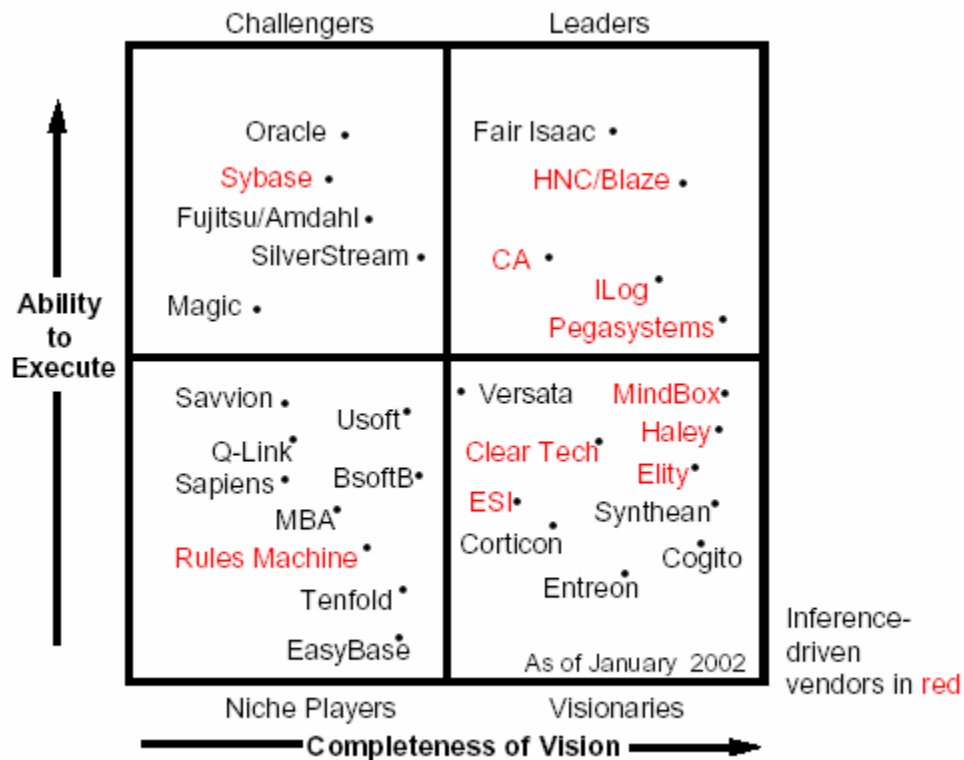
Consider the following set of rules:

Rule 1: If A and C	Then F
Rule 2: If A and E	Then G
Rule 3: If B	Then E
Rule 4: If G	Then D

Like in the previous example, we are trying to prove that D is true. First find a rule that proves that D is true. Rule 4 does so. This provides a sub-goal – to prove that G is true. Rule 2 proves that G is true. Since it is already known that A is true the new sub-goal is E. Now Rule 3 proves that E is true. Thus the next sub goal is B. However, we now know that B is true as it is one of the assertions. Therefore, E is true, which implies G is true, which in turn implies that D is true.

Unlike forward chaining, backward chaining is useful in situations where the quantity of data is potentially very large.

3 Rules Engine market



Source: Gartner Research

Gartner rates vendors in two areas: vision and ability to execute. The Visionaries are those companies that have a strong technology infrastructure, are market leaders and have a number of investment participants. The companies that are characterized as executors are those that have an efficient and easy installation packages, strong service and support and an experienced management team.

I have gone further to characterize some of the prominent commercial and open source Rules Engines present in the market today.

Commercial Rules Engines

- Advisor
- Haley's
- Jess (Java Expert System Shell)
- JRules
- ILog

Open source Rules Engines

- CLIPS
- Drools
- Mandarax

For this study an analysis of the following two Rules Engines has been done.

Rules Engine 1: ILog Rule Engine

The ILog Rule Engine has the following characteristics:

Rule life cycle management

ILog RuleKit provides a complete set of tools for developing and managing business rule applications. The main features of this system are:

- Central rule management and storage with Business Rule Repository
- Create, edit and debug rules using a graphic environment provided by the Rule Builder
- Supports a predefined set of easy-to-use business rule languages with English-like syntax
- Centrally store business rule metadata in the Business Rule Repository. This repository allows rules to be persisted as files or tables in a database.

Flexible deployment

- ILog Business Rule components integrate seamlessly with Java and C++ as well as with J2EE, XML and Web Services architecture
- Embed JRules in Enterprise Java Beans
- Dynamically process XML schemas, objects and rules
- Access any JDBC-compliant database
- Deploy as a Web service and invoke Web services

Performance

- Designed using algorithms that are specially optimized for object-oriented systems
- Maximize performance and scalability with compiled rules
- Sequential execution

Extensibility

- Customizable language
- Open API

Rules Engine 2: Jess (Java Expert System Shell)

Rule life cycle management

Rules can be managed in many ways: command line application, GUI application, servlets or applets. For the representation of rules Jess uses the concept of forward chaining. In Jess after a rule has been executed it never has to assert the fact again.

Flexible deployment

Jess can be flexibly deployed as an Applet or as an Application.

Jess as an applet:

As an applet, Jess can be used to answer general question and answer situations by simply embedding the applet class on a web page.

As mentioned

Jess as an application:

Jess can be used in two ways. First, it can be a rules engine (a program that efficiently applies rules to data). A rule-based program can have hundreds or even thousands of rules and Jess will continually apply them to data that is maintained in the form of a knowledge base.

Performance

Jess's rule engine uses an improved form of a well-known algorithm called Rete to match rules against the knowledge base. Jess is actually faster than some popular expert system shells written in C. Jess is a memory intensive algorithm because Rete explicitly trades space for speed.

Extensibility

Jess is also a general purpose programming language that allows access to all Java classes and libraries. It is for this reason that Jess is also frequently used as a dynamic scripting or rapid application development environment.

Another important point is that Jess works in real time. While Java code generally must be compiled before it can run, a line of Jess code is executed immediately as it is being typed. It is also easy to extend the Jess language with new commands written in Java or in Jess itself.

4 Prototypes

4.1 Prototype 1

Note

The code for this prototype was downloaded from the Java Pro website. I modified the code to demonstrate a simple approach to implement a Rules Engine.

Description

This simple implementation is based on the event-condition-action paradigm.

- Data is represented in XML
- Rules are XSL templates imposed on the data
- Rules are triggered by events from XML data

The technologies used are Java, XML and XSL. By using a small set of technologies, the development and maintenance of such a system becomes easier. It is important to note that Java can be replaced by language that can communicate with XSLT. Switching the language would require the developer to make only minor code modifications.

XSLT is an XML based language for transforming XML source into something known as a result. The result can be another XML file or an HTML file or a WML file. However, for this prototype we will use XSLT in a way that is was not designed for. To some extent the approach exposes some of the logic programming flavor of XSL.

XML data

The data utilized by the XSLT Rules Engine should be formatted as XML data. Third party tools can be used or in-house software can be developed to convert data into XML form. Here is a code snippet of the XML data used in this prototype:

There are two XML files used in this prototype. However, I have discussed on one example in this case.

(File location: /data/data1.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<data1>
  <account number="111111" type="0">
    <open_date>20021009</open_date>
    <amount>990</amount>
  </account>
  <account number="111112" type="0">
    <open_date>20030309</open_date>
    <amount>30000</amount>
  </account>
</data1>
```

The data has account information for two accounts of a bank. To demonstrate how easily rules can be represented as XSLT documents we can apply to following rules:

- 1) If cash amount < 1000 then action is **warning**
- 2) If cash amount > 20000 then action is **invite to invest**
- 3) If account is opened in 2003 then action is **promotion message**

XSLT rule specification

The above mentioned rules are included within XSL templates. You will never find conditional statements with <, > or = in your application code. Hence if business rules change the application code does not have to change.

XSLT contains pattern matching and branching features that enable business rules processing.

The following code snippet shows how a rule is represented:

(File location: rulesets\ruleset1)

```
<xsl:for-each select="amount">

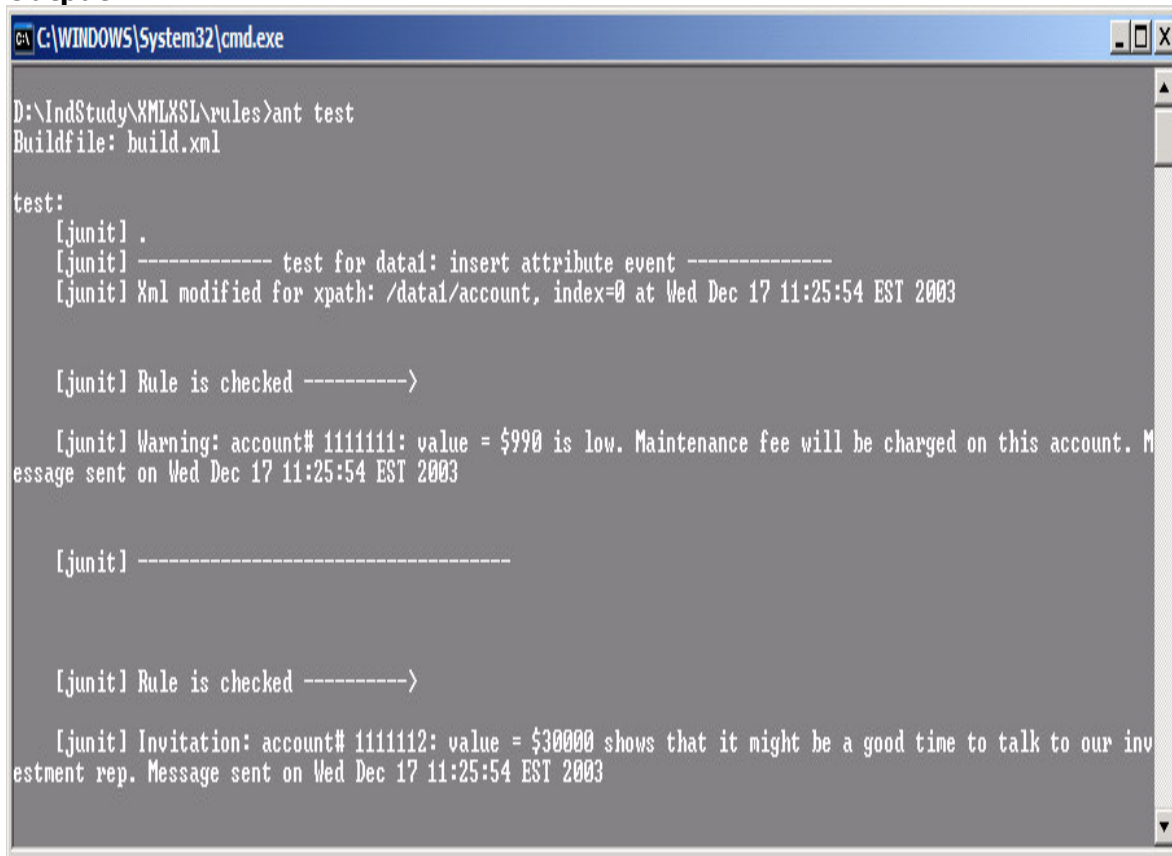
  <xsl:variable name="account_number">
    <xsl:value-of select="../@number"/>
  </xsl:variable>

  <xsl:variable name="amt">
    <xsl:value-of select="number(.)"/>
  </xsl:variable>
  <xsl:call-template name="rule1">
    <xsl:with-param name="amount" select="$amt"/>
    <xsl:with-param name="acc_num" select="$account_number"/>
  </xsl:call-template>
  <xsl:call-template name="rule2">
    <xsl:with-param name="amount" select="$amt"/>
    <xsl:with-param name="acc_num" select="$account_number"/>
  </xsl:call-template>
</xsl:for-each>
```

This is the driver program. It starts of by matching the element, "amount". For each element "amount" it selects the value of the attribute "number" that belongs to the element "account_number". It then calls rule 1 and rules 2 and passes the parameters amount and account number with each call.

The files, ruleset_1.xsl and ruleset_2.xsl contain an action for each event. The line:
<xsl:value-of select="action:sendMsg2Stdout(concat('Warning: account# ', \$acc_num, ' : value = \$', \$amount, ' is low. Maintenance fee will be charged on this account. Message sent on ', \$time), \$index)" />
sends a warning message to the user if cash amount is less than 1000.

Output



```
C:\WINDOWS\System32\cmd.exe

D:\IndStudy\XML\XSL\rules>ant test
Buildfile: build.xml

test:
[junit] .
[junit] ----- test for data1: insert attribute event -----
[junit] Xml modified for xpath: /data1/account, index=0 at Wed Dec 17 11:25:54 EST 2003

[junit] Rule is checked ----->

[junit] Warning: account# 111111: value = $990 is low. Maintenance fee will be charged on this account. Message sent on Wed Dec 17 11:25:54 EST 2003

[junit] -----

[junit] Rule is checked ----->

[junit] Invitation: account# 111112: value = $30000 shows that it might be a good time to talk to our investment rep. Message sent on Wed Dec 17 11:25:54 EST 2003
```

Rule 1

The above output shows that for account number 111111, the account balance of \$990 is low and that a maintenance fee will be charged on this account.

Rule 2

For account number 111112, the account balance of \$30000 is greater than \$20000 hence the message is to encourage the user to talk with an investment representative in order to invest more.

Instructions to run the code

File structure:

Folder data	Contains two XML files: data1.xml and data2.xml
Folder lib	Contains rules.jar that is created by the build file
Folder libxml	Contains junit and xalan jar files needed at both compilation and run time
Folder rulesets	Contains two rule sets that in turn contain XSL files
Folder src	Contains the Java source files
Folder test	Contains the test client
Build.xml	Build file used by ANT
README.txt	Contains instructions to run the application – I have modified the run instructions from the original version

An installation of Jakarta ANT is required. I used apache-ant-1.5.2.

Click here to get source code: [Link](#)

4.2 Prototype 2

Description

This prototype is to demonstrate how rules can be represented in a Relational Database Management System. In this case the RDBMS is Oracle.

Steps to understand this prototype

1. Study the SQL scripts
2. Refer to the notes mentioned under the "Database structure" section
3. Study how rules are represented using regular expressions
4. Study the relationship diagram that shows how rules stored in the database are represented as a decision tree

My work at U.S. Steel involved developing a Rules Engine to validate XML based orders. The important point here is that this prototype paves the way for building a rules-based system for any enterprise system that needs validation.

List of technologies that would be used for the final implementation

- XML: Transactions are represented in XML form. Click [here](#) to see a sample.
- Relational database: Rules, Rule Groups and Rule Control data are stored in the database
- JAXB: Used to parse the XML data
- Java Web Components: JSP (presentation logic) and Servlets (business logic)
- JavaScript: Dynamic representation of rules in the form of a decision tree
- Tomcat server: Servlet container + web server

SQL files

Table name	
TSB_EC_RULES	link
TSB_EC_RULES_GROUPING	link
TSB_EC_TRANSACTION_FORMAT	link
TSB_EC_RULE_ELEMENTS	link
TSB_EC_ELEMENT_GROUPS	link
TSB_EC_ELEMENT_TRX_XREF	link
TSB_EC_RULES_CNTL	link

Database structure

This section describes each table and lists any relationships.

a. TSB_EC_RULE_ELEMENTS

Description

This table defines elements and their details like: PK_ELEMENT_ID, ELEMENT_NAME. These elements are created by the Code Generation system

Relationships

None

b. TSB_EC_TRANSACTION_FORMAT

Description

This table defines Transaction format ids and their details like: APP_CD, SUB_APP_CD, RULE_TRX_NAME, FK_ELEMENT_ID

Relationships

TSB_EC_RULE_ELEMENTS.PK_ELEMENT_ID::FK_ELEMENT_ID - One to many

c. TSB_EC_RULES

Description

This table defines rules and their details like: PK_RULE_ID, EXPRESSION_VALUE, EXPRESSION_FORMAT. These rules are used by 1 or more transactions.

Relationships

None

d. TSB_EC_ELEMENT_GROUPS

Description

This table defines those elements (DataTNDs) that need to be accessed remotely.

Relationships

None

e. TSB_EC_ELEMENT_TRX_XREF

Description

Links the elements with their transaction format ids.

Relationships

- TSB_EC_ELEMENT_GROUPS.PK_ELEMENT_GROUP_ID::FK_ELEMENT_GROUP_ID -

One to many

- TSB_EC_TRANSACTION_FORMAT.PK_TRX_FORMAT_ID::FK_TRX_FORMAT_ID -

One to many

f. TSB_EC_RULES_GROUPING

Description

This table defines a group of rules. The rules are evaluated in a linear fashion depending on the condition, AND / OR

Relationships

- TSB_EC_RULES.PK_RULE_ID::FK_RULE_ID - One to many

- TSB_EC_ELEMENT_GROUPS.PK_ELEMENT_GROUP_ID::FK_ELEMENT_GROUP_ID -

One to many

- TSB_IT_ERROR_CD_HEADER.PK_ERROR_CD_ID::FK_ERROR_CD_ID - One to many

g. TSB_EC_RULES_CNTL

Description

This table defines the rule control true.

ASSOCIATION	AND or OR
BRANCH_IND	T or F: Mark F if an error is to be recorded, i.e. if it is a leaf node and an error occurs it will be raised and the field isError will be equal to True
ERROR_BREAK_IND	Always F
PROCS_VALUE	T or F: Condition that determines the path of the down the tree
FK_TRX_FORMAT_ID	The rule control pointer that links the rule control tree the element

Relationships

- TSB_EC_RULES_GROUPING.RULE_GROUP_NUM::FK_RULE_GROUP_NUM
- TSB_EC_TRANSACTION_FORMAT.PK_TRX_FORMAT_ID::FK_TRX_FORMAT_ID

Types of Rules

Rules are represented in the form of regular expressions. The main aim of the Rules table is to make a list of reusable rules that can be linked in any combination in the Rules Grouping table.

I have listed a few types of rules. This is not an exhaustive list however it will give the reader an idea of how rules can be represented:

Type 1: match

Expression value: `m/\d+/`

Expression type: R – regular expression

Regular expressions are used when you need to match a pattern. For example to check for a valid time we use this regular expression:

`m/\b(((0-1)[0-9])|(2[0-3]))\.[0-5][0-9]\b/`

Type 2: comparison

Operator: `<, <=, >, >=, =, !=`

Expression value: A numerical value used as the comparison operand

Expression type: O.I

Rules that work with integers.

Example: Check if value is greater than 99999 - Operator: `>`, Expression value: 99999, Expression type: O.I

Type 3: date pattern match

Expression format: MM-dd-yy

Expression type: D.P

Rules that need to match a date pattern. Expression format is normally one of the following types:

`MM.dd.yyyy | MM/dd/yyyy | MM.dd.yy | MM-dd-yyyy | MM:dd:yy | MM:dd:yyyy`

Type 4: Not in

Operator: `,`

Expression value: User defined list of values delimited with a comma. This list contains values that are used for matching.

Expression type: NOT.IN

This rule is based on string comparison. It is used to check if a value does not match all the values in a given set of values (specified in Expression value that is separated with a delimiter which is specified under Operator).

Type 5: List.match

Operator: `e1.d1.e2.d2.e3.d3.e4.d4` (where e is element id and d is depth for 4 values respectively)

Expression type: List.match.1

This rule is a specialized version of List.match. We can use this type of rule when we want to match a specific node. Thus, we can specify the exact path of the node by listing the element id and depth of each node in the hierarchy.

For example:

```

<ActualDimensions>
  <Weight>
    <UOM>
      <CodeValue>99</CodeValue>
    </UOM>
  </Weight>
  <Weight>
    <UOM>
      <CodeValue>01</CodeValue>
    </UOM>
  </Weight>
  <Length>
    <UOM>
      <CodeValue>01</CodeValue>
    </UOM>
  </Length>
</ActualDimensions>

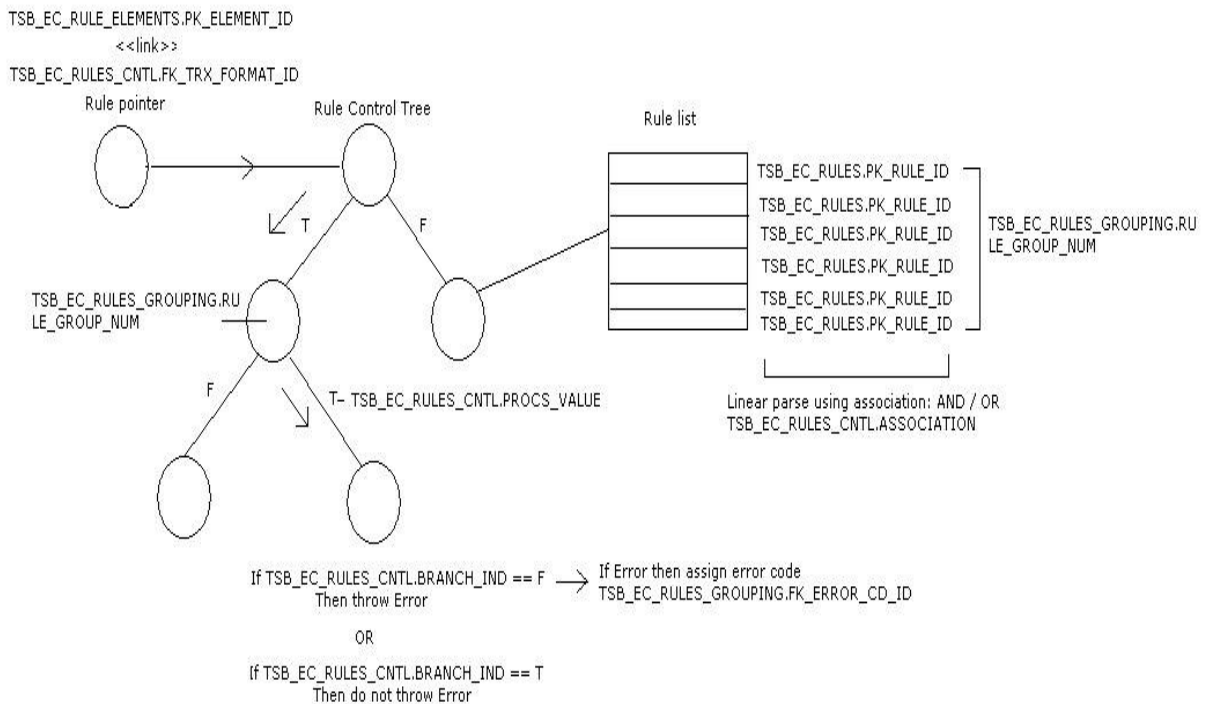
```

In the above example we want to check if the node UOM with CodeValue=01 exists. Thus, the operator in this case would look something like this:
65700.9.65200.10.65203.11
Where Element Ids = 65700, 65200, 65203 and depths = 9, 10, 11

Relationship diagram

This relationship diagram shows how rules stored in the database are represented as a decision tree.

MAPPING



5 Conclusion

Critical business processes are bottlenecked by costly, inefficient manual procedures that are too difficult to automate using conventional applications. Existing enterprise systems are unable to change to meet the evolving needs of the business. The results are increased costs, missed business opportunities and declining customer base.

1. Rules driven Business Process Management (BPM)

This is where Business Process Management plays an important role. BPM systems take out the complexity of automating business processes. Furthermore BPM is a software system that streamlines business processes by integrating them with existing systems.

The underlying framework of a BPM solution is a Rules Engine where the business rules can automate human decision making while leaving the existing enterprise infrastructure unchanged. A rule driven BPM system uses business rules engine technology to drive the business process. The main advantage of such a system is that a business analyst can go ahead and model the process by defining business rules rather than wait for a developer to code the logic of the task.

2. Artificial Intelligence Arena

As stated in the earlier section, there are many commercial rule-based engines in the market such as Advisor, Haley's, and Jess. These products come from the Artificial Intelligence arena and thus are designed for highly inference based situations. The applicability of such rule-based engines in enterprise information systems is not suitable as their capability often exceeds the business requirements.

I believe that a simple rule-base engine that runs on an event-condition based paradigm is strongly suited for enterprise information systems. Examples can be drawn from this paper: Prototype 1 and Prototype 2. Both these prototypes are based on an event-condition based paradigm. They are programmed using Java, XML and XSL and not with JESS or Prolog that are based on inference methods like forward chaining and backward chaining respectively.

The tradeoff of using rule-based systems based on the event-condition paradigm is the risk of excessive maintenance. However, such maintenance is necessary and unavoidable because business changes continuously and thus rules need to be updated. Does this mean that the development cycle of the rule-based engine will never end? The answer is that the development done by the programmer will end and the business user will take over. By providing a comprehensive User Interface with tools to create, modify and delete rules and to model decision trees, a rule-based system can evolve with changes in business.

3. Rules Engine applicability in business

Rule-based programming is widely used in the insurance and financial services industries where complex criteria needs to applied to large amounts of information. In the Artificial Intelligence field Knowledge based systems are a well known research topic. However, these systems have not been widely deployed in corporate information systems. A main reason may be because these systems depend on pre-built large domain bases to work. Creation and maintenance of these domain bases requires a large commitment of resources and time. Such a commitment is hard for

companies to make. However, this is where rules-based systems come in. Deployment of Knowledge based systems without an exhaustive domain description is feasible. The domain description can be incrementally built using condition-action rules. Business rules are in fact small pieces of knowledge about a business domain. The only issue with such rules is that they are widely scattered by different applications, often replicated and usually hard-coded. Thus they can end up being not reusable. A business rules architect can work around this problem by creating a business rules framework that can be used globally in all systems of a company.

This study has shown us the benefits and scope of a Rules Engine when applied to business. A growing trend today is using rule-based technology for Enterprise Application Integration. Today, most EAI vendors have rules capabilities that are either too simple to use or too complex to implement either directly or indirectly.

The business user of tomorrow needs to focus on finding a solution that has all or most the following attributes:

- Ease of use: GUI interface with decision tables for the business user
- Versatility: Easy integration with other vendors
- Rule management: Central rule management capability
- Inference capability: The workflow should be able to make its own decisions
- Performance: Rules Engines need to be efficient to process large amounts of data

The bottom line is that the Business Rules Engine is one of the major technologies that can help meet the fast changing technology world. There would be substantial growth in this sector in the near future.

6 Sources

Section	Source
Introduction	Drools guide (PDF file) http://drools.org/pdf/drools-guide.pdf
Introduction	Rules Power Technical white paper (PDF file)
Introduction	Pega Rules: http://www.pegacom/ProductsServices/EnterpriseBPMTechnology/prpc/prpcoverview/documents/BREBrochureFINAL.pdf
Introduction	Jess Online Manual: http://herzberg.ca.sandia.gov/jess/docs/61/
Rules Engine Market	Java Pro: http://www.fawcette.com/javapro/2003_08/online/xml_yboglaev_08_01_03/
Rules Engine Market	Jess Online Manual: http://herzberg.ca.sandia.gov/jess/docs/61/
Rules Engine Market	ILog white papers: http://www.ilog.com/products/jrules/whitepapers/index.cfm ilog_ds_rules.pdf
Rules Engine Market	http://www.kei.co.kr/technology/RBMS_Trend_Eng.pdf
Prototype 1	Java Pro: http://www.fawcette.com/javapro/2003_08/online/xml_yboglaev_08_01_03/
Conclusion	http://www.theserverside.com/resources/article.jsp?l=Jess
Conclusion	http://www.kei.co.kr/technology/RBMS_Trend_Eng.pdf
Conclusion	http://www.theserverside.com/resources/article.jsp?l=RuleBasedMDB