**T.C. DOĞUŞ UNIVERSITY**
**INSTITUTE OF SCIENCE AND TECHNOLOGY**
**COMPUTER AND INFORMATION SCIENCES MASTER PROGRAM**

# A RULE BASED EXPERT SYSTEM GENERATION FRAMEWORK USING AN OPEN SOURCE BUSINESS RULE ENGINE

**M.S. Thesis**

**Gökhan POLAT**
**2003097002**

**ADVISOR :**
**Doç. Dr. Selim AKYOKUŞ**
**Associate Professor Dr. Selim Akyokuş**

**DECEMBER 2006**
**İstanbul**

To my family

**ACKNOWLEDGEMENTS**

I would like to express special thanks to my supervisor Ass.Prof. Selim Akyokuş for his great support and encouragement. Without his help this thesis can not be completed. In addition, during master program and thesis study, my wife's patient and help was incredible. These type of supports was luck for my entire study.

**ABSTRACT**

Knowledge is key instrument for the deciding processes. On the other hand, for a deciding process, gathering knowledge and learning are very difficult phases. For this reason, in the last decades, studies are focused on the machine-learning systems and the expert systems for the most of the knowledge oriented areas, like academic, commercial, military and industrial areas.

In this thesis, a framework is developed for the rule base learning expert systems. Briefly, this framework will take a data set, induct the rules from this data set, construct an expert system according to inducted rules, and give a web based interface for testing new cases.

There are a lot of concepts in this study. Classification, decision tree, knowledge acquisition, ID3 algorithm, rule base systems, expert systems, rule engines, open source perspective are some of them. These concepts will be discussed briefly, after the discussion; framework will be explained with some examples.

Examples will show the reusability of the framework. Different data set can be applied the framework. But data set must be convenient to the ID3 decision tree algorithm. Other restrictions will be defined next sections. After constructing expert system new cases can be tested.

This framework has some principles:

- Java technologies are used
- Open source tools are used where needed
- Standardizations are applied where available

As a result of these principles, usability of the framework is dramatically increased.

## ÖZET

Bilgi, karar verme süreçlerinde anahtar araçtır. Öte yandan, bir karar verme sürecinin en zor evreleri bilgiyi edinme ve öğrenmedir. Bu nedenle, akademik, ticari, askeri ve endüstriyel alanlar gibi bilgi merkezli pek çok alanda çalışmalar makina öğrenmesi sistemleri ve uzman sistemlere odaklanmıştır.

Bu tezle bir kural tabanlı öğrenen uzman sistem çatısı sunulmaktadır. Kısaca bu çatı uygun data kümesini alır, bu kümeden kurallar çıkarır, bu kurallara göre bir uzman sistem kurar ve yeni durumları test etmek için web tabanlı bir arayüz verir.

Bu çalışma birçok konuyu kapsamaktadır. Sınıflandırmalar, karar ağaçları, ID3 algoritması, kural tabanlı sistemler, uzman sistemler, kural motorları ve açık kaynak kodlu yaklaşım bunlardan bazılarıdır. Bu konular kısaca açıklanacak, değerlendirmeler sonrasında çatı örnekler ile açıklanacaktır.

Örnekler çatının yeniden kullanabilirliğini gösterecektir. Çatı uygulama, farklı data kümeleri ile çalışabilmelidir. Ancak seçilen data kümeleri ID3 algoritmasına uygun olmalıdır. Diğer kısıtlamalar ileriki bölümlerde açıklanacaktır.

Çatı aşağıdaki temel prensiplere dayanmaktadır.

- Java teknolojileri kullanılmıştır
- Gerekli durumlarda açık kaynak kodlu araçlar kullanılmıştır
- Mümkün olduğunca standartlaştırma uygulanmıştır

Bu prensipler sonucunda, çatının kullanılabilirliği önemli ölçüde artmıştır.

**TABLE OF CONTENTS**

**LIST OF FIGURES**

**LIST OF TABLES**

**ABBREVIATIONS**

| | |
|---|---|
| **API** | Application Program Interface |
| **ID3** | Iterative Dichotomiser 3 |
| **J2EE** | Java 2 Enterprise Edition |
| **J2SE** | Java 2 Standard Edition |
| **JDK** | Java Development Kit |
| **JSR** | Java Specification Request |
| **LHS** | Left Hand Side |
| **OO** | Object Oriented |
| **OSS** | Open Source Software |
| **RETE** | NET (latin) |
| **RHS** | Right Hand Side |
| **XML** | Extensible Markup Language |
| **TDIDT** | Top-Down Induction of Decision Tree |

# 1. INTRODUCTION

Learning is very attractive concept for computer-based researches. There are a lot of methods for learning process. At the same time, a lot of result-based system uses these methods.

Learning capabilities are needed for intelligent systems that can remain useful in the face of changing environments or changing standards of expertise (Buchanan B.G., 1989)

Expert systems mostly need mentioned learning methods as a supporting system. In other words, this supporting system feeds the resulting expert system. Feeding methods are using some techniques to support expert systems , which are defined as;

- rule-based techniques
- inductive techniques
- hybrid techniques
- symbol-manipulation techniques
- case-based techniques
- qualitative techniques
- model-based techniques
- temporal reasoning techniques
- neural networks

In this thesis, rule-based techniques and inductive techniques are used to feed expert system. These two techniques are integrated as a new hybrid model. Before giving details of the new hybrid model, background of the expert system will be discussed.

An expert system's central goal is to help professional in the process of shifting from old implementation to modern approaches, based on latest technologies. An expert system assists the human designer by efficient encoding of expert knowledge and by reusing the available systems.

The use of expert systems in the speed-up of human professional work has been in two orders of magnitude with resulting increases in human productivity and financial returns. Last decade shows that a growing number of organizations shift their informational systems towards a knowledge-based approach. This fact generates the need for new tools and environments that intelligently port the legacy systems in modern, extensible and scalable knowledge-integrated systems (Pop D. and Negru V., 2003).

The most popular technique of knowledge acquisition is still done with an interaction with a human expert. A knowledge engineer, a person acquiring knowledge, interacts with an expert either by observation of the expert in action or by interview. As a result, rules are produced, first in plain English, later on in the coded form accepted by a computer. It is the responsibility of the knowledge engineer to acquire knowledge in such a way that the knowledge base is as complete as possible (Dobroslawa et al., 1995).

Classical expert system can also be explained in figured manner (Figure 1.1). As seen clearly on the explanations and the figure, human acts as a key role on the this picture.

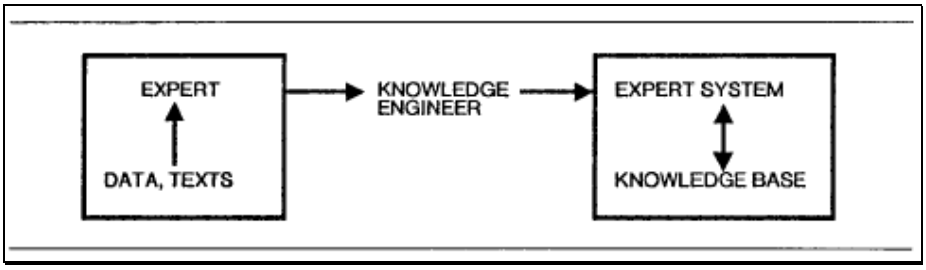



Figure 1.1 Classic transfer of expertise
(Buchanan B.G and Shortliffe E.H., 1984)

The process of working with an expert to map what he or she knows into a form suitable for an expert system to use has come to be known as knowledge engineering. We refer to the process of mapping an expert's knowledge into a program's knowledge base as knowledge engineering.

For the representation of knowledge in expert systems, a number of forms are used, such as: rules set (production rules, association rules, rules with exceptions), decision tables, classification and regression trees, instance-based representations, and clusters. Each representation has its advantages and drawbacks (Pop D. and Negru V., 2003).

Main idea of this study is; an expert system may be built by human by means of a rule set, which is the natural way for humans to understand the knowledge. But limited capability of the human causes a bottleneck on the expert system (Figure 1.2). On the other hand, a decision tool , which uses some data mining methods, can make the process much more easier.

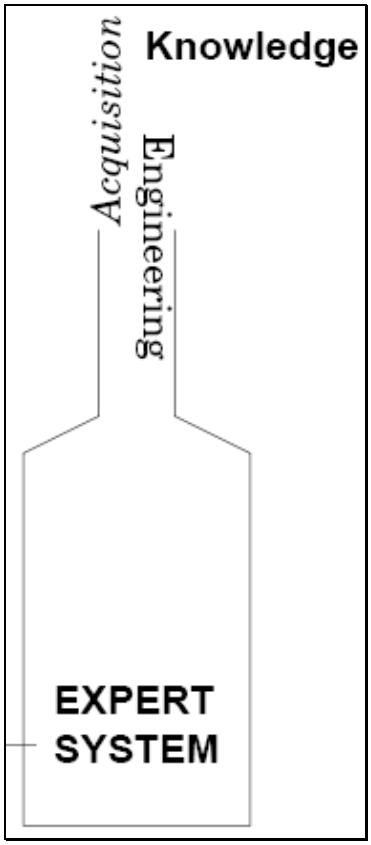



Figure 1.2 Bottleneck on the classic expert system
(Buchanan B.G and Shortliffe E.H., 1984)

The knowledge needed to drive the pioneering expert systems was codified through protracted interaction between a domain specialist and a knowledge engineer. While the typical rate of knowledge elucidation by this method is a few rules per man day, an expert system for a complex task may require hundreds or even thousands of such rules (Quinlan, J.R., 1985).

To avoid drawbacks of the knowledge-based systems, in this thesis, learning-based methodology is used. At this point, to be clear on the framework structure, summarized comparison of the knowledge-based and learning-based approaches is needed.

### 1.1. Labor Intensive Knowledge-Based Approach

Human experts construct a set of rules with which concepts can be identified in a text

Advantages :

- Human experience can be used to quickly distinguish good rules from bad ones

Disadvantages :

- Laborious, time-intensive development process
- Requires the availability of human expertise

### 1.2. Automated Learning Approach

Automated learning algorithms induce a model with which concepts can be identified in a an example data set . Learning requires :

- a goal-directed process of a system that improves the knowledge or the knowledge representation of the system by exploring experience and prior knowledge
- acquisition of new declarative knowledge
- development of motor and cognitive skills through instruction and practice
- organization of new knowledge into general effective representation
- discovery of new facts and theories through observation and experimentation
- a process of knowledge construction, not of knowledge recording or absorption

Advantages :

- There is no need for human experts
- Techniques are largely domain independent
- Exceptions are not likely to be overlooked

Disadvantages :

- (Large amounts of) example data are required to train most common machine learning algorithms
- Resulting model might not be easily understandable by human observer

## 2. GOAL AND ARCHITECTURE

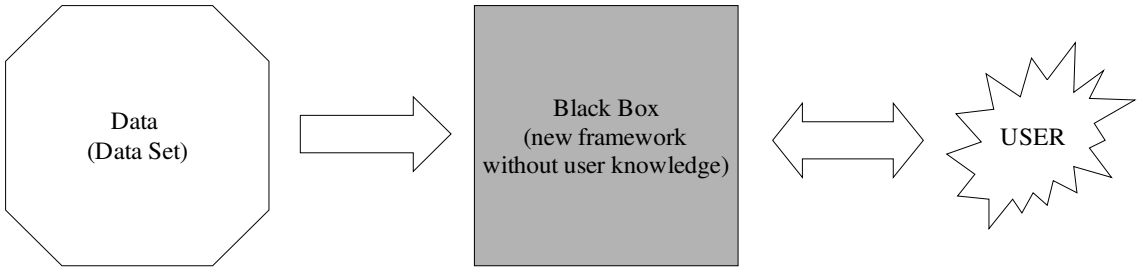



Figure 2.1 Brief representation of the framework

As explained in the introduction, an automated learning methodology for rule generation is more reasonable approach for expert systems. This approach has a superiority against to the knowledge-based systems besides some drawbacks. On the other hand, better observations and researches can produce large and good enough example data. One of the assumptions of this study is that; data should be reasonable.

This thesis presents a complete framework for constructing an expert system. This framework based on learning approach, can also be expressed as rule-based.
This expert system framework includes:

- Uses convenient data set
- Uses decision tree data mining method as a learning method
- Uses ID3 decision tree algorithm
- Automatic rule generation
- Uses JBossRule engine
- Generates rule file
- Generates web based interface for test new cases
- Uses open source products

Detailed architecture of learning expert system developed in this study is shown in Figure 2.2.

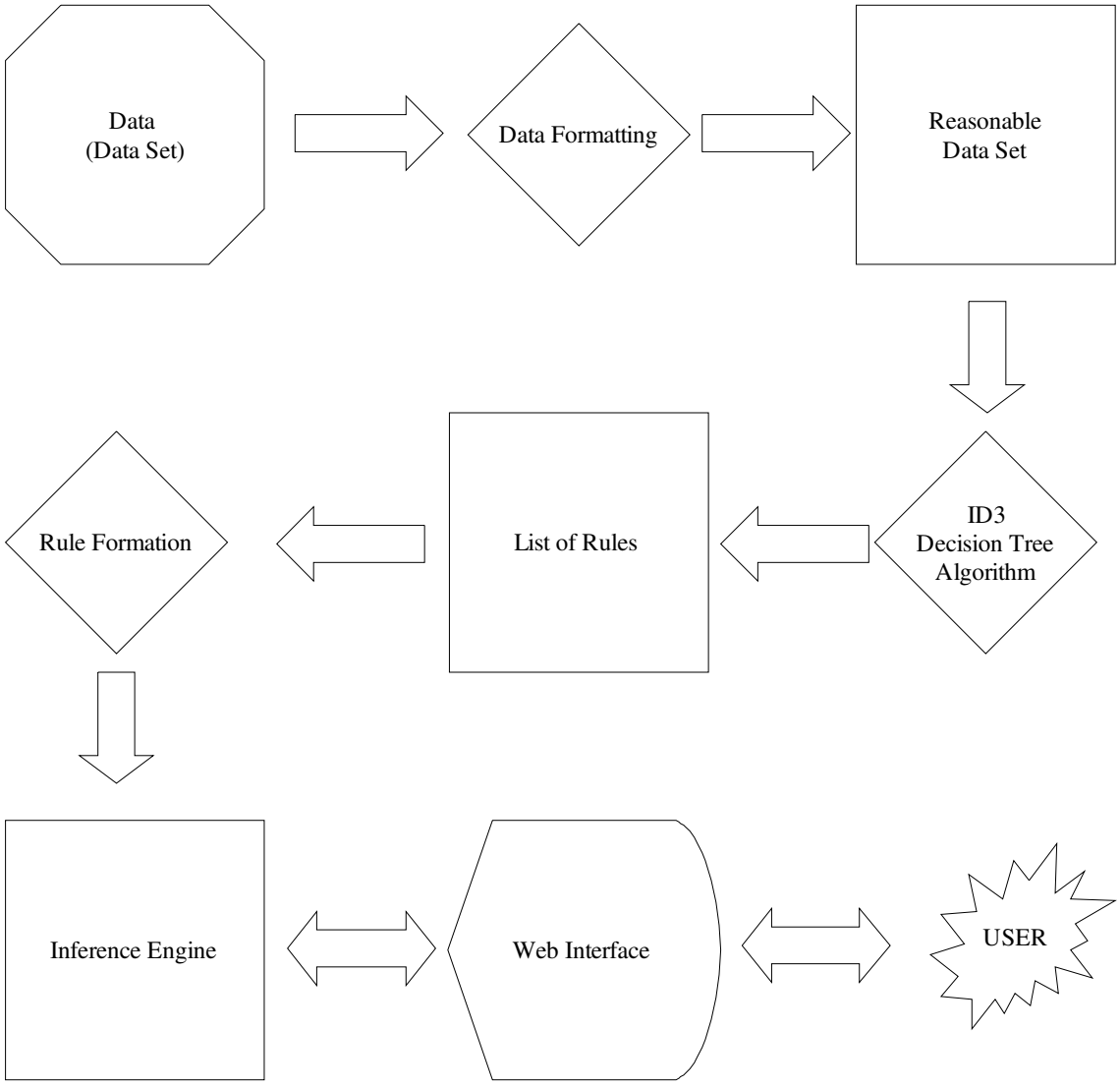



Figure 2.2 The architecture of the framework

## 3. MACHINE LEARNING - INDUCTIVE LEARNING

After the words about base of expert system and learning approach, machine learning and inductive learning should be explained briefly. Because the field of machine learning in concerned with the question of how to construct computer programs that automatically improve with experience, this concept should be clear.

Knowledge-based systems are relatively old structures. A newer paradigm, generally considered to be the machine learning approach, has attracted attention of researchers in artificial intelligence, computer science, and other functional disciplines such as engineering, medicine, and business. In contrast to Knowledge-based systems which acquire knowledge from human experts, machine learning systems acquire knowledge automatically from examples, i.e., from source data. Machine learning refers to a system capable of the autonomous acquisition and integration of knowledge. This capacity to learn from experience, analytical observation, and other means, results in a system that can continuously self-improve and thereby offer increased efficiency and effectiveness.

Knowledge acquisition is the transfer and transformation of problem-solving expertise from some knowledge source to a program. Learning from examples may automate much of the knowledge acquisition process by exploiting large data bases of recorded experience (Buchanan B.G and Shortliffe E.H., 1984).

To gain a knowledge, machine learning techniques, as rote learning, learning by being told, learning by analogy, learning from examples, and learning from discovery, have been studied extensively by AI researchers over the past two decades. Among these techniques, learning from examples, a special case of inductive learning appears to be the most promising machine learning technique for knowledge discovery or data analysis. It induces a general concept description that best describes the positive and negative examples.

Machine-learning approaches commonly used for classification include inductive-learning algorithms such as decision-tree induction and rule induction, instance-based learning , neural networks, genetic algorithms, and Bayesian-learning algorithms. Among the various machine-learning approaches developed for classification, inductive learning from instances

is the most commonly used method in real-world application domains. Inductive learning techniques are fast compared to other techniques. Another advantage is that inductive learning techniques are simple and their generated models are easy to understand. Finally, inductivelearning classifiers obtain similar and sometimes better accuracies compared with other classification techniques (Pham D. T. and Afify A. A., 2004).

Inductive learning has received considerable attention since the 1950s. Nowadays some approaches (eg. some growing toolkit of programs) can assist in knowledge acquisition (Buchanan B.G., 1989).

Induction refers to inference of a generalized conclusion from particular instances. Inductive learning techniques are used to automatically construct classifiers using labeled training data. Different inductive learning algorithms was developed, some of them are listed below;

- Decision Trees
- Find Similar (a variant of Rocchio's method for relevance feedback)
- Naïve Bayes
- Bayes Nets
- Support Vector Machines (SVM)

All methods require only on a small amount of labeled training data (i.e., examples of items in each category) as input. This training data is used to "learn" parameters of the classification model. (Dumais et al., 1998)

Conventional knowledge based system's inference mechanism is deductive. On the other hand learning systems use inductive structure. To understand difference between deduction and induction a table is constructed (Table 3.1). As seen from this table induction helps us for generalization. Whereas deduction goes from general to specific, induction, generates hypotheses, goes from specific to general .

Table 3.1 Deduction versus Induction

| **Deduction** | **Induction** |
|---|---|
| All humans are mortal. (Axiom) | Socrates is human. (Background K.) |
| Socrates is human. (Fact) | Socrates is mortal. (Observation(s)) |
| **Conclusion:** | **Generalization:** |
| Socrates is mortal. | All humans are mortal. |

Inductive-learning techniques can be divided into two main categories, namely, decision-tree induction and rule induction (Pham D. T. and Afify A. A., 2004).In this study, decision tree induction and rule induction are used as a composite inductive learning method. Decision tree method will be explained in detail in the next sections.

# 4. CLASSIFICATION RELATED BASIC CONCEPTS

As described previous sections in this thesis, decision tree algorithm is chosen as an inductive learning method which is the one of the best machine learning techniques. The inductive learning can be done with classification methods. Classification is one of the most important data mining tasks. In this chapter, classification will be explained briefly.

## 4.1. Brief Explanation

Classification is a key data mining technique whereby database tuples, acting as training samples, are analyzed in order to produce a model of the given data . Each tuple is assumed to belong to a predefined class, as determined by one of the attributes, called the classifying attribute. Once derived, the classification model can be used to categorize future data samples, as well as provide a better understanding of the database contents. Classification has numerous applications including credit approval, product marketing, and medical diagnosis (Kamber et al.,1997).

## 4.2. Types of Classification

A number of classification techniques from the statistics and machine learning communities have been proposed. These techniques are also called as classification algorithms.

Algorithms that classify a given instance into a set of discrete categories are called as classification algorithms. These algorithms work on a training set to come up with a model or a set of rules that classify a given input into one of a set of discrete output values. Most classification algorithms can take inputs in any form, discrete or continuous although some of the classification algorithms require all of the inputs also to be discrete. The output is always in the form of a discrete value. Decision trees and Bayes nets are examples of some of classification algorithms (Polumetla A., 2006).

This thesis focuses decision tree algorithm, because it fits the main goal of the study. On the other hand an other method, decision table will be explained and the differences between two decision methods will be described.

### 4.3. Decision Table

A *decision table* consists of a two-dimensional array of cells, where the columns contain the system's constraints and each row makes a classification according to each cell's value (Pop D. and Negru V., 2003). A decision table consists of a two-dimensional array of cells. Associated with each row in the array is a classification. A decision table can be viewed as a conjunction of row rules. An example of the decision table can be seen in Figure 4.1.

| **Conditions** | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1. Precipitation? | Y | Y | Y | N | N | N |
| 2. Temperature | Less than 50 | Between 50-70 | More than 70 | Less than 50 | Between 50-70 | More than 70 |
| **Actions** | | | | | | |
| 1. Heavy Jacket | X | | | X | | |
| 2. Light Jacket | | X | X | | X | |
| 3. No Jacket | | | | | | X |

Figure 4.1 An example of decision table
(Kolahi S., 2006)

### 4.4. Decision Tree

Decision tree are commonly used for gaining information for the purpose of decision-making. Decision tree starts with a root node on which it is for users to take actions. From this node, users split each node recursively according to the decision tree learning algorithm. The final result is a decision tree in which each branch represents a possible scenario of decision and its outcome.

In summary, the systems described here develop decision trees for classification tasks. These trees are constructed beginning with the root of the tree and proceeding down to its leaves. (Quinlan, J.R., 1985).

### 4.4.1. When to Consider Decision Tree

Decision trees are considered as an efficient technique to express classification knowledge and to use it. Their success is explained by their ability to handle complex problems by providing an understandable representation easier to interpret and also their adaptability to the inference task by producing logical rules of classification (Elouedi et al.,2000).

Decision trees are useful for automating decision processes that are part of an application program. Decision trees are used in a large number of applications, and the number continues to grow as practitioners gain experience in using trees to model decision making processes. Present applications include various pixel classification tasks, language understanding tasks such as pronoun resolution, fault diagnosis, control decisions in search, and numerical function approximation (Utgoff P.E., 1995).

Decision tree learning algorithm is suited when

- Instance is represented as attribute-value pairs. For example, attribute 'Temperature' and its value 'hot', 'mild', 'cool'. We are also concerning to extend attribute-value to continuous-valued data (numeric attribute value) in our project.
- The target function has discrete output values. It can easily deal with instance which is assigned to a boolean decision, such as 'true' and 'false', 'p(positive)' and 'n(negative)'. Although it is possible to extend target to realvalued outputs, we will cover the issue in the later part of this report.
- The training data may contain errors. This can be dealt with pruning techniques that we will not cover here.

## 4.5. Comparison

Table 4.1 Comparison of the models

| *Criteria* | **Structured English** | **Decision Tables** | **Decision Trees** |
|---|---|---|---|
| **Determining Conditions and Actions** | Second Best | Third Best | Best |
| **Transforming Conditions and Actions into Sequence** | Best | Third Best | Best |
| **Checking Consistency and Completeness** | Third Best | Best | Best |

The decision table and decision tree are essential tools for systems analysts. These decision aids are used by systems analysts in depicting conditional logic for programmers and in validating this logic with the user. In addition, many authors recommend the decision table and tree as useful aids in decision making (Subramanian G.H et al., 1989).

A rule can be defined by structured English words, a decision table and a decision tree. All three methods have some advantages and drawbacks. Table 4.1 shows that decision tree is the most effective method for defining rule according to specified criteria.

## 5. BUILDING DECISION TREE - ID3

Several methods have been proposed to construct decision trees. These algorithms input the training set composed by instances where each one is described by the set of attribute values and its assigned class. The output is a decision tree ensuring the classification of new instances

Decision tree learning algorithm has been successfully used in expert systems in capturing knowledge. The main task performed in these systems is using inductive methods to the given values of attributes of an unknown object to determine appropriate classification according to decision tree rules.

ID3 is a simple decision tree learning algorithm developed by Ross Quinlan (1983). The basic idea of ID3 algorithm is to construct the decision tree by employing a top-down, greedy search through the given sets to test each attribute at every tree node. In order to select the attribute that is most useful for classifying a given data set, a metric called information gain, which will be defined later, is used.

ID3 is used as a machine learning methods which induces a rule set that is a subset of all potential rules hidden in the original data set. Successful applications of ID3, C4 and other decision tree algorithms have provided knowledge bases for working expert systems whose task is to classify. They are widely used in a variety of fields notably in artificial intelligence applications. Their success is explained by their ability to handle complex problems by providing an understandable representation easier to interpret and also their adaptability to the inference task by producing logical rules of classification (Elouedi et al. 2000).

One approach to the induction task above would be to generate all possible decision trees that correctly classify the training set and to select the simplest of them. The number of such trees is finite but very large, so this approach would only be feasible for small induction tasks. ID3 was designed for the other end of the spectrum, where there are many attributes and the training set contains many objects, but where a reasonably good decision tree is required without much computation. It has generally been found to construct simple decision trees, but the approach it uses cannot guarantee that better trees have not been overlooked.

The basic structure of ID3 is iterative. A subset of the training set called the *window* is chosen at random and a decision tree formed from it; this tree correctly classifies all objects in the window. All other objects in the training set are then classified using the tree. If the tree gives the correct answer for all these objects then it is correct for the entire training set and the process terminates. If not, a selection of the incorrectly classified objects is added to the window and the process continues(Quinlan, J.R., 1985).

### 5.1. Data Set

This study assumes that data is correct and classifiable. A measurement by a specific variable is the assignment of a specific value to that variable, notionally by the real-world process. The value set belonging to a variable is a discrete set of names, usually describing qualitative properties. A value set must have at least two members. The prototypical case is a boolean variable with values {true, false}, but other value sets are possible: for example the variable sex has the value set {male, female}. If a variable refers to a continuous measurement, its value set frequently names the results of a series of relational tests on the measurement
The basis is a universe of objects that are described in terms of a collection of attributes. Each attribute measures some important feature of an object and will be limited here to taking a (usually small) set of discrete, mutually exclusive values(Quinlan, J.R., 1985).

### 5.2. ID3 Algorithm

Several algorithms have been developed for learning decision trees. In the artificial intelligence community, the most used is based on the TDIDT (Top-Down Induction of Decision Tree) approach. In that approach, the tree is constructed by employing a recursive divide and conquer strategy. Its steps can be defined as follows:

- By using an attribute selection measure, an attribute will be chosen in order to partition the training set in an ”optimal” manner.
- Based on a partitioning strategy, the current training set will be divided into training subsets by taking into account the values of the selected attribute.
- When the stopping criterion is satisfied, the training subset will be declared as a leaf.

In the literature many attribute selection measures are proposed in. Among the most used, we mention the information gain used within the ID3 algorithm. The information gain of an attribute A relative to a set of objects S measures the effectiveness of A in classifying the training data.

Algorithm is defined in Figure 5.1.

**ID3(Examples, Target-attribute, Attributes)**

/* Examples: The training examples; */
/* Target-attribute:The attribute whose value is to be predicted by the tree; */
/* Attributes: A list of other attributes that may be tested by the learned decision tree. */
/* Return a decision tree that correctly classifies the given Examples */
**Step 1:** Create a Root node for the tree
**Step 2:** If all *Examples* are positive, Return the single-node tree *Root*,with label = +
**Step 3:** If all E*Examples* are negative, Return the single-node tree *Root*,with label = -
**Step 4:** If *Attributes* is empty, Return the single-node tree *Root*, with label = most common value of *Target-attribute* in *Examples*
**Step 5:** Otherwise Begin

- A ← the attribute from *Attributes* that best (i.e., highest information gain) classifies *Examples*;
- The decision attribute for *Root* ← A;
- For each possible value, $v_i$ , of A,
  - Add a new tree branch below *Root*, corresponding to the test A=$v_i$;
  - Let $Examples_{(v_i)}$ be the subset of *Examples* that have value $v_i$ for A;
  - If $Examples_{(v_i)}$ is empty
    - Then below this new branch add a leaf node with label = most common value of *Target-attribute* in *Examples*
    - Else below this new branch add the subtree ID3($Examples_{(v_i)}$, *Target-attribute*, *Attributes*- A ))

End

Return *Root*

Figure 5.1 ID3 Algorithm
(Yüret D., 2003)

### 5.2.1. Entropy

Entropy, characterizes the (im)purity of an arbitrary collection of examples. That is, it measures the homogeneity of examples. Entropy equation for two classes positive and negative is below;

$$Entropy(S) \equiv - p_p \log_2 p_p - p_n \log_2 p_n \qquad \text{(Equation 5.1)}$$

Where

S is a sample of training examples

$p_p$ is the proportion of positive examples in S

$p_n$ is the proportion of negative examples in S

In summary; entropy is expected number of bits needed to encode class (p or n) of randomly drawn member of S.

If all instances in S belong to the same class, then E(S) equals 0.

If S contains the same number of instances for each class, then E(S) equals 1.

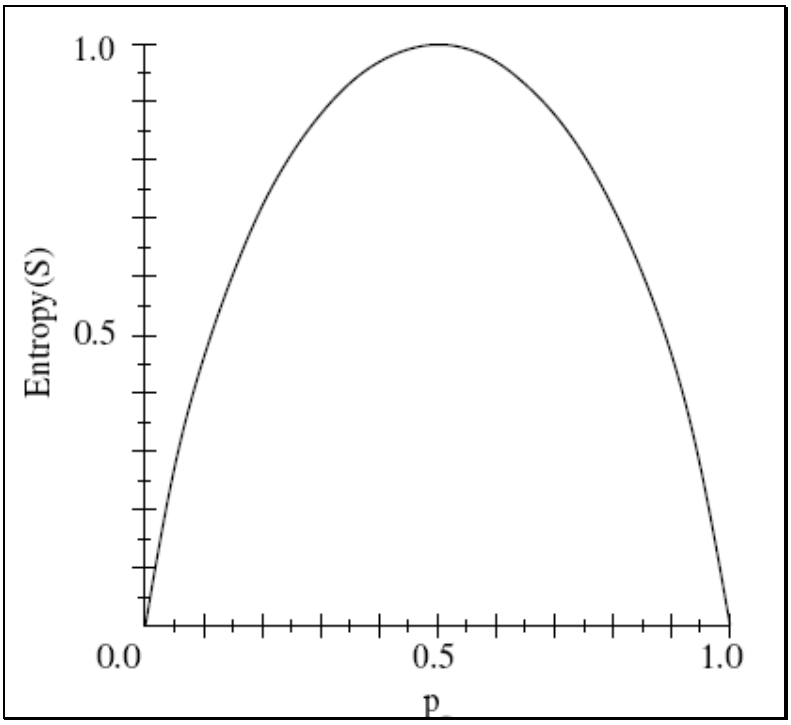



Figure 5.2 Entropy Graph

### 5.2.2. Information Gain

Information gain is the answer of the "How do we choose the best attribute?" question in decision tree algorithm.

In order to measure the worth of an attribute a statistical property is defined, information gain, which measures how well a given attribute separates the training examples according to their target classification.

Information Gain equation is given;

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v) \qquad \text{(Equation 5.2)}$$

The information gain Gain(S,A) is the expected reduction in entropy caused by knowing the value of the attribute A.

### 5.2.3. Example Calculations with Weather Data Set

The weather problem is a example data set which we will use to understand how a decision tree is built. It comes from Quinlan's paper which discusses the ID3 algorithm. It is reproduced with slight modifications by Witten I.H., Frank E. (1999), and concerns the conditions under which some hypothetical outdoor game may be played. The data is shown in Table 5.1.

Table 5.1 Weather Data set

| **Outlook** | **Temperature** | **Humidity** | **Windy** | **Play(Class)** |
|---|---|---|---|---|
| sunny | hot | high | FALSE | N |
| sunny | hot | high | TRUE | N |
| overcast | hot | high | FALSE | P |
| rain | mild | high | FALSE | P |
| rain | cool | normal | FALSE | P |
| rain | cool | normal | TRUE | N |
| overcast | cool | normal | TRUE | P |
| sunny | mild | high | FALSE | N |
| sunny | cool | normal | FALSE | P |
| rain | mild | normal | FALSE | P |
| sunny | mild | normal | TRUE | P |
| Overcast | mild | high | TRUE | P |
| Overcast | hot | normal | FALSE | P |
| Rain | mild | high | TRUE | N |

In this dataset, there are five categorical attributes outlook, temperature, humidity, windy, and play. We are interested in building a system which will enable us to decide whether or not to play the game on the basis of the weather conditions, i.e. we wish to predict the value of play using outlook, temperature, humidity, and windy. We can think of the attribute we wish to predict, i.e. play, as the output attribute, and the other attributes as input attributes.

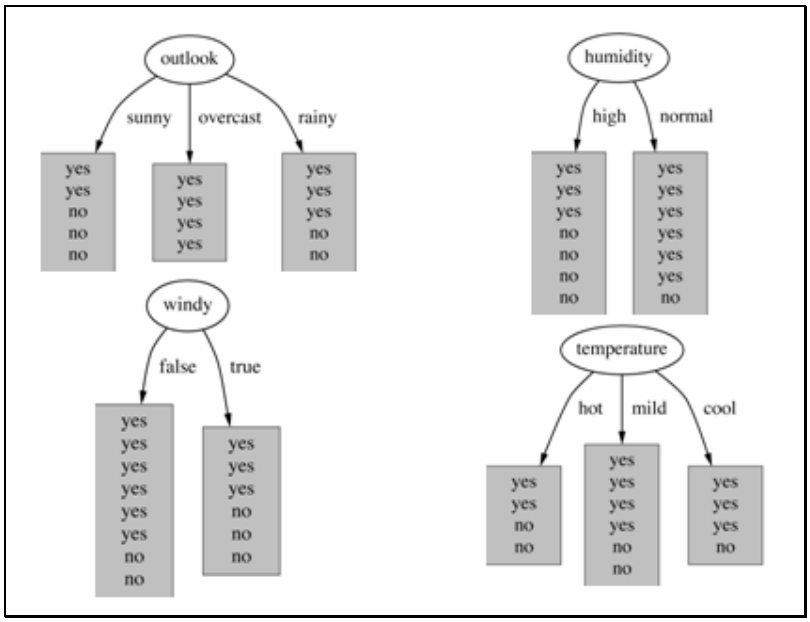



Figure 5.3 Which attribute to select?

Calculation for the entropy of the humidity attribute is as follows

*H(D) = -(9/14) log (9/14) - (5/14) log (5/14) = 0.94*

*H(D, Humidity = High) = -(3/7) log (3/7) - (4/7) log (4/7) = 0.985*

*H(D, Humidity = Normal) = -(6/7) log (6/7) - (1/7) log (1/7) = 0.592*

Calculation for the information gain of the humidity attribute;

*Gain(D, Humidity) = 0.94 - (7/14) * 0.985 + (7/14) * 0.592 = 0.151*

Similarly, for wind attrbute;

*Gain (D, Wind) = 0.94 - (8/14) * 0.811 + (6/14) * 1.0 = 0.048*

Information gains for all attributes;

*Gain(D, Humidity) = 0.151 bits*

*Gain(D, Wind) = 0.048 bits*

*Gain(D, Temperature) = 0.029 bits*

*Gain(D, Outlook) = 0.246 bits*

Clearly, outlook is the highest gain, so this should be the root node. According to the algorithm, the procedure should continue recursively until the end. After that, the result tree can be obtained as seen in Figure 5.4.

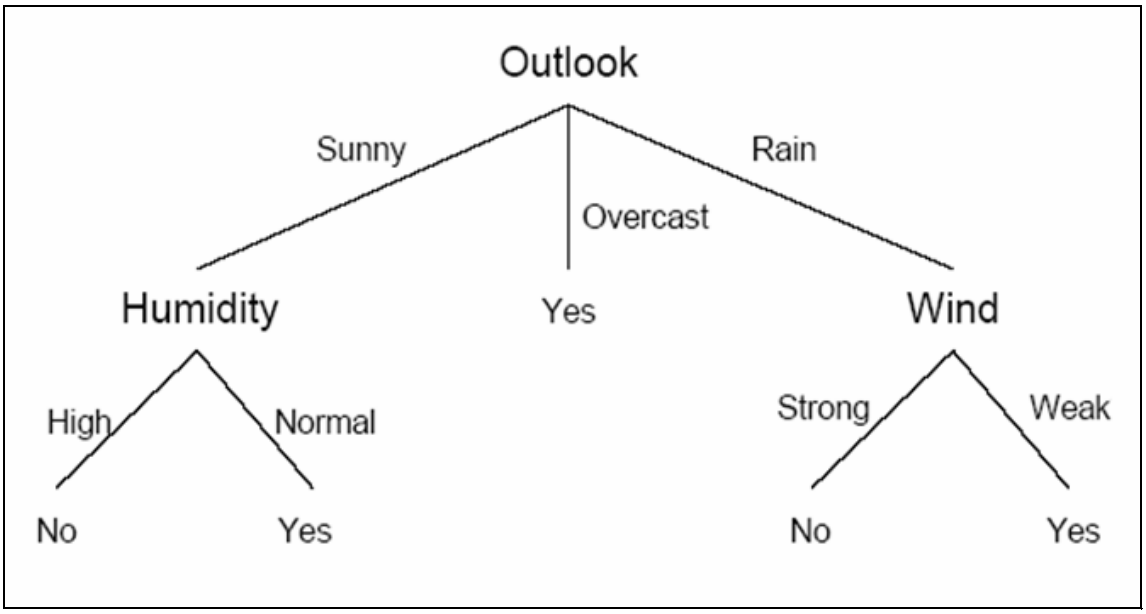
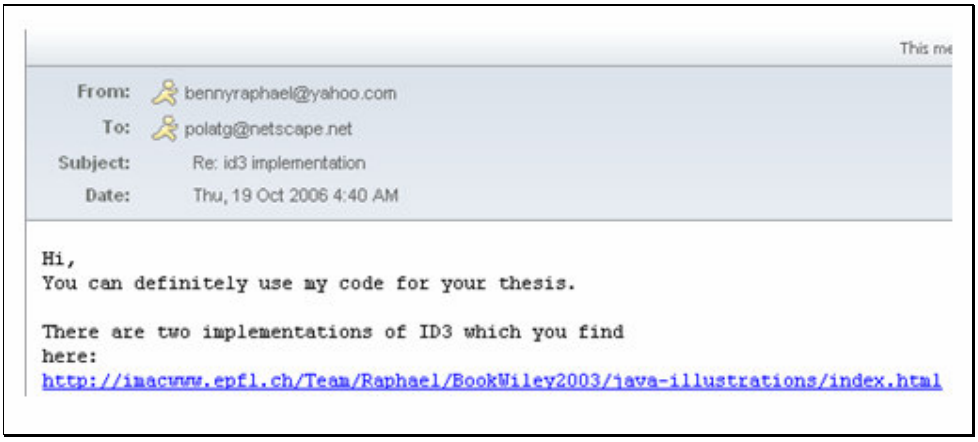



Figure 5.4 Final decision tree for weather data set

### 5.3. ID3 Java Implementation

In this thesis, java technologies are used for all implementations. On the other hand, ID3 java implementation is not developed by the author. It is originally developed by Dr.Benny Raphael. With his permission, some modifications are done for the framework adaptation.



This me

From: bennyraphael@yahoo.com
To: polatg@netscape.net
Subject: Re: id3 implementation
Date: Thu, 19 Oct 2006 4:40 AM

Hi,
You can definitely use my code for your thesis.

There are two implementations of ID3 which you find
here:
http://imacwww.epfl.ch/Team/Raphael/BookWiley2003/java-illustrations/index.html

Figure 5.5 Permission from Dr.Benny Raphael

Important methods of the Dr. Benny Raphael's implementation and our modifications are below. Furthermore, its javadoc parts in the Appendix section.

### 5.3.1. Original Methods

**readData** : Function to read the data file. The first line of the data file should contain the names of all attributes. The number of attributes is inferred from the number of words in this line. The last word is taken as the name of the output attribute. Each subsequent line contains the values of attributes for a data point. If any line starts with // it is taken as a comment and ignored. Blank lines are also ignored.

**calculateEntropy** : Calculates the entropy of the set of data points. The entropy is calculated using the values of the output attribute which is the last element in the array attributes.

**decomposeNode** : This function decomposes the specified node according to the ID3 algorithm. Recursively divides all children nodes until it is not possible to divide any further.

**printTree** : This function prints the decision tree in the form of if/then/else structure. The action part of the rule is of the form outputAttribute = "symbolicValue" or outputAttribute = { "Value1", "Value2", .. } The second form is printed if the node cannot be decomposed any further into an homogenous set.

### 5.3.2. New Methods

**listRules** : This function prints the rules as a sentence.

**createRules4File** : This function exports the rules to a .drl file which is used for JBossRule Engine.

# 6. RULE-BASED SYSTEMS

Rule-based systems are a very simple model that can be used to solve many decision problems. Instead of representing knowledge in a relatively declarative, static way, rule-based system represent knowledge in terms of a bunch of rules. A rule-based system consists of a bunch of IF-THEN rules, a bunch of facts, and some interpreter controlling the application of the rules, given the facts.

Rule-Based systems maintain a small database of facts about the world, so that they can perform reasoning; if a fact about the world matches a condition of a rule, that condition is judged to be fulfilled (Kingston J., 1987).

In Summary, a rule-based system can be defined as a system that uses rules to derive conclusions from premises.

## 6.1. Requirements of a Rule-Based System

- A set of facts to represent the initial working memory. This should be anything relevant to the beginning state of the system.
- A set of rules. This should encompass any and all actions that should be taken within the scope of a problem, but nothing irrelevant. The number of rules in the system can affect its performance.
- A condition that determines that a solution has been found or that none exists. This is necessary to terminate some rule-based systems that find themselves in infinite loops otherwise.

A rule-based system works by applying the rules that are applicable to the current state of the system. At the beginning, the “working memory” consists of the description of the initial state of the system. It then finds all the rules that are applicable to this state.

In this study, facts are objects (java beans) which are asserted into the working memory. Facts are any java objects which the rules can access. Detailed explanation about framework is given on the section 10 and interaction between object-oriented functionality and rule-based knowledge is described in Figure 6.1.

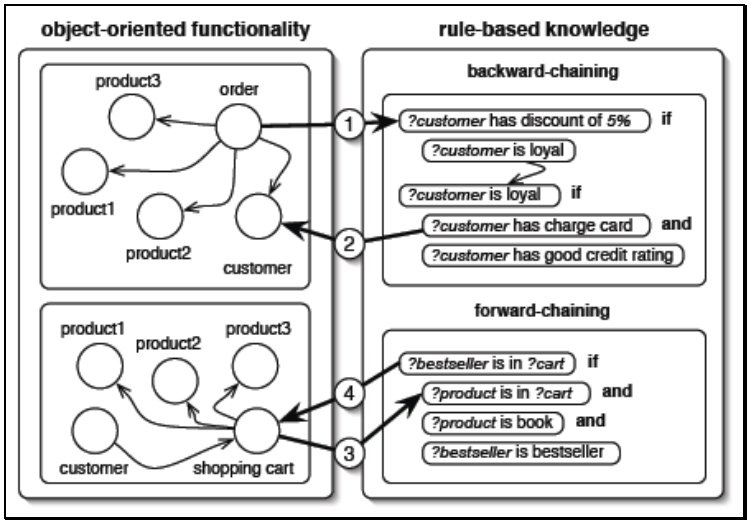



Figure 6.1 Interaction between OO and rule-based system
(Maja D., 2004)

## 6.2. Architecture of a Rule-Based System

A typical rule-based system contains below items. It can be also seen in the Figure 6.2.

- An inference engine
- A rule base
- A working memory

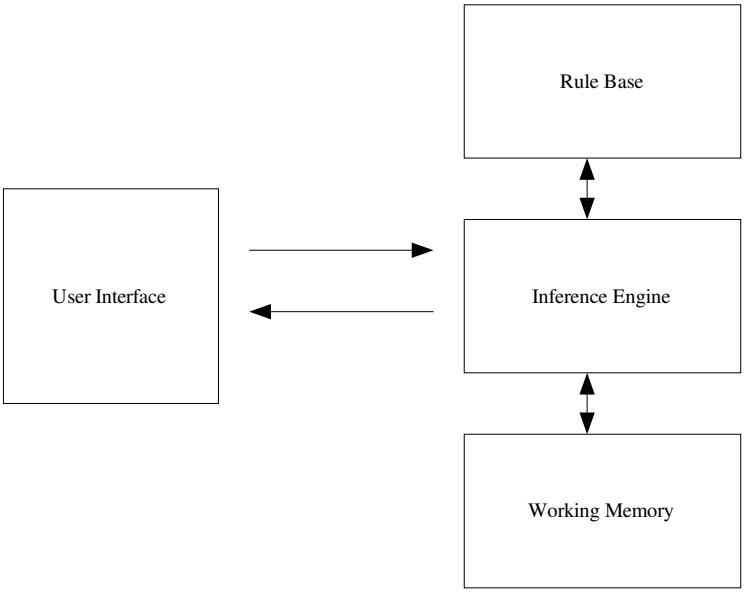



Figure 6.2 Architecture of a rule-based system

### 6.2.1. Inference Mechanism

The way knowledge systems model human reasoning is called inference. The inference engine is a component of a rule engine that fires the rules.

Many rule-based expert systems are developed using expert system shells. A shell provides facilities for writing rules easily, often in a format which resembles English syntax, and also provides a strategy for solving problems in general - that is, it has built-in algoithms for deciding which rule is to be used when. This strategy is known as the shell's inference mechanism. A shell can be thought of as a rule-based expert system without any knowledge, or a framework around which an expert system can be developed.

An inference mechanism consists of algorithms and the rules in a rule base. There are two methods for executing rules in rule-based systems, forward chaining and backward chaining.

#### 6.2.1.1. Forward Chaining Systems

Forward chaining searches the inference rules until it finds one where the “If” clause is known to be true. When found it can conclude, or infer, the “Then” clause, resulting in the addition of new information to its dataset.

Forward-chaining systems are data-driven. The facts in such systems are represented in a working memory that is continually updated. Furthermore, in these systems rules represent possible actions to take when specified conditions hold on items in the working memory, they are sometimes called condition-action rules. The conditions are usually patterns that must match items in the working memory, while the actions usually involve adding or deleting items from the working memory.

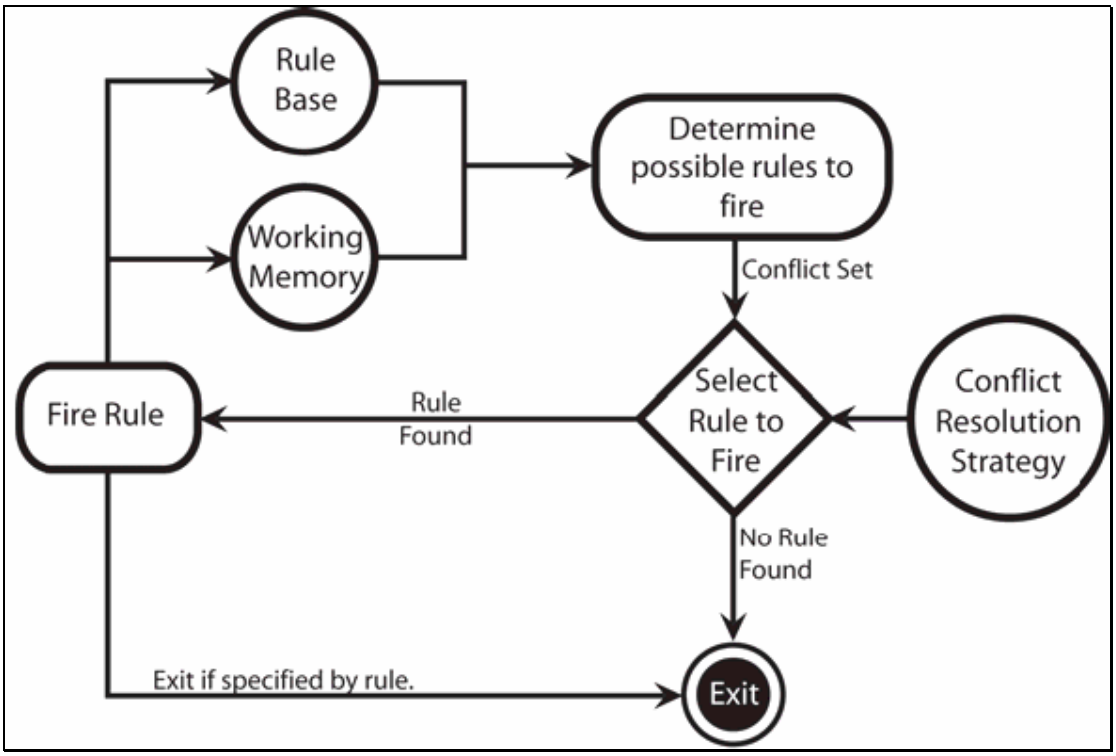



Figure 6.3 Forward chaining system
(Chan S. T. and Gröndahl F., 2005)

#### 6.2.1.2. Backward Chaining Systems

Backward chaining would search the inference rules until it finds one which has a "Then" clause that matches a desired goal. If the "If" clause of that inference rule is not known to be true, then it is added to the list of goals (in order for your goal to be confirmed you must also provide data that confirms this new rule).

Backward-chaining systems are goal-driven. These systems look for the action in the THEN clause of the rules that matches the specified goal. In other words, they look for the rules that can produce this goal. If a rule is found and fired, they take each of that rule's conditions as goals and continue until either the available data satisfies all of the goals or there are no more rules that match.

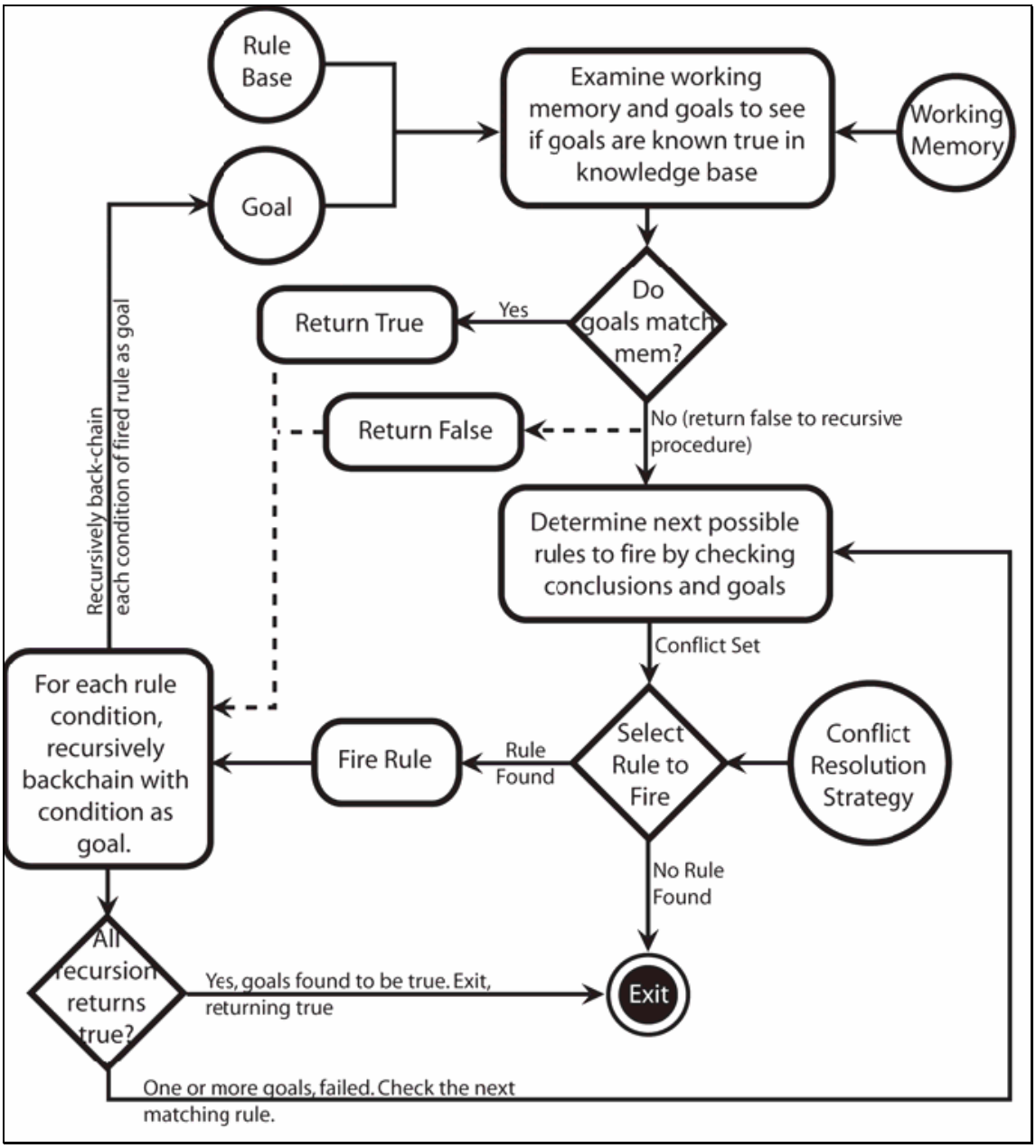



Figure 6.4 Backward chaining system
(Chan S. T. and Gröndahl F., 2005)

### 6.2.2. Rule Base

The rules need to be stored somewhere. The rule base contains all the rules the system knows. They may simply be stored as strings of text, but most often a rule compiler processes them

into some form that the inference engine can work with more efficiently (Chanda M. S.,2004).

The rule base contains specific knowledge about the problem area presented in rules. Rules, in the form "if - then" are elementary units of knowledge (Dobroslawa et al., 1995).

### 6.2.3. Working Memory

In a typical rule engine, the working memory, sometimes called the fact base, contains all the pieces of information the rule-base system is working with. The working memory can hold both the premises and conclusions (result objects) of the rules. Some implementations can hold only objects of a specific type, and others can include objects of any type, for example Java objects (Chanda M. S.,2004).

The working memory holds concrete data in the form of the object-attribute-value triplets. The data is used by the rule engine to match to the rules' conditions. Two possibilities arise:

1. If one of the rule conditions has no variables, then it is satisfied only if an identical expression is present in the working memory
2. If one of the rule conditions has at least one variable, i.e. if it is a pattern, then it is satisfied only if there exists data in working memory which matches it, taking into account the rule's other conditions that have been matched (D'Hondt M., 2004).

## 6.3. Rules

Rules are similer to the if-then statements of traditional programming languages. An order rule can look like this, in an English-like pseudocode:

IF
    A student is in the laboratory
AND
    He/She is hungry
THEN
    He/She should go to the canteen to eat

In the simplest design, a rule is an ordered pair of symbol strings, with a left-hand side and a right-hand side (LHS and RHS). A rule can also be viewed as a simple conditional statement, and the invocation of rules as a sequence of actions chained by modus ponens.

A rule consists of two parts: an antecedent and a consequent. The rule antecedent consists of one or more conditions that specify when and where to apply the rule. If the conditions of the rule are met, then the second part of a rule – the consequent – specifies the actions to take when the conditions of the rule are met.

Rules are generally used to represent knowledge about strategies for solving problems in a particular area (Kingston J., 1987).

Basically two different formalisms of expressing rules exist, production rules, used in production systems, and first-order predicate logic used in logic-based systems. Production systems consist of three parts, the production rules, the working memory and the rule engine (Rosenberg F. and Dustdar S., 2005).

Obviously, production system means rule-based system and production rules are one of the major part of the constructed rule-based system in this study.

## 6.4. Rule Engine

The term “Rule Engine” can be defined for any system that uses rules, in any form, that can be applied to data to produce outcomes; which includes simple systems like form validation and dynamic expression engines.

### 6.4.1. Advantages of the Rule Engine

- **Declarative Programming** : Rule engines allow you to say "What to do" not "How to do it". They key advantage of this point is that it can make it easy to express solutions to hard problems, and consequently have those solutions verified (rules are much easier to read then code). Rule systems are capable of solving very hard problems, yet providing a solution that is able to explain why a "decision" was made.

- **Logic and Data Separation** : Your data is in your domain objects, the logic is in the rules.
- **Speed and Scalability** : The Rete algorithm, Leaps algorithm, and its descendents, provide very efficient ways of matching rule patterns to your domain object data.
- **Centralization of Knowledge** : By using rules, you are creating a repository of knowledge which is executable.
- **Tool Integration** : Tools such as eclipse provide ways to edit and manage rules and get immediate feedback, validation and content assistance. Auditing and debugging tools are also available.
- **Explanation facility** : Rule systems effectively provide an "explanation facility" by being able to log the "decisions" made by the rule engine (and why the decisions were made). Understandable rules (readable by domain experts). By creating object models that model your problem domain, rules can look very close to natural language. They lend themselves to logic that is understandable to domain experts who may be non technical.

### 6.4.2. Why and When to Use a Rule Engine?

While rule engines can solve a lot of problems for us , it is worth considering if a rule engine is appropriate for the application. Some important points are:

- Application Complexity : For applications that shuffle data to and from a database , but not much more , it is probably best not to use a rules engine. However , where there is even a moderate amount of processing, it is worthwhile considering the use of rule engine. This is becuase most applications develop complexity over time and rule engine will let you cope easily with this.
- Application Lifetime : Using a rule engine pays off especially in the medium to long term. Prototypes can benefit from the combination of rule engine and agile methods to take the 'prototype' into production.
- Application updates : The only sure thing about your requirements is that they will change, either during or just after development. A rule engine helps to cope with this by specifying the business rule in one or more easy to configuration files.

### 6.4.3. Which Rule Engine to Use?

There are many business rule engines on the market, both open source and commercial. Here is a list of the most popular commercial business rule engines (Chanda M. S.,2004):

- JRules from ILOG
- Advisor from Brokat
- OPSJ from Charles Forgey
- QuickRules from Yasu Technologies
- CommonRules from IBM alphaworks
- exteNd Director from Novell
- ACQUIRE from acquired Intelligence

The list of the most popular open source rule engines is as follows:

- JBoss Rule Engine
- JESS (Java Expert System Shell) from Sandia National Labs
- Mandarax
- CLIPS from Gary Riley
- InfoSapient

In this study, JBoss Rule Engine is used as a rule engine. One of the major reason is that this rule engine is an open source rule engine. Why open source question will be explained next section.

### 6.4.4. RETE Algorithm

The RETE algorithm was invented by Dr. Charles Forgy and documented in his PHd thesis in 1978-79 (Forgy C., 1979),. A simplified version of the paper was published in 1982.

There are many methods for optimizing rule engines to execute rules more efficiently. Most rule engines use the Rete (Latin for `net') Algorithm for optimization. This algorithm is intended to improve the speed of forward-chained rule-based engines by limiting the effort

required to re-compute a conflict set after a rule is fired. In the Rete algorithm, executable rules are compiled into a network. Input data to the network consists of changes to the working memory. Objects are inserted, removed, and modified. The network processes these changes and produces a new set of rules to be fired. The network minimizes the number of evaluations by sharing tests between rules and propagating changes incrementally. Briefly, the rete algorithm eliminates the inefficiency in the simple pattern matcher by remembering past test results across iterations of the rule loop. Only new or deleted working memory elements are tested against the rules at each step. Furthermore, Rete orgonizes the pattern matcher so that these few facts are only tested against the subset of rules that may actually match. The main drawback of this algorithm is its high memory space requirement.

# 7. JBOSS RULE AS AN OPEN SOURCE RULE ENGINE

## 7.1. Open Source Perspective

Although there is considerable confusion about the strengths and weaknesses of open source software (OSS), it has become clear that OSS has an important role to play in the IT industry and business in general. OSS, for the most part, represents a software development process. It can be leveraged to provide considerable value and complement commercial software products. At the same time, commercial software products will continue to play a critical role for the foreseeable future (Heintzman D., 2003)..

The IT industry is going through major changes. New concepts in technology, such as Web services and grid computing, are opening the door to tremendous opportunities for taking e-business to the next level of profitability. The potential of these technologies to transform business is truly remarkable, and open standards and open source software will play increasingly critical roles in this new world.

To clear the open source concept some definition should be placed.

**Open source software :** Is the software whose source code is published and made available to the public, enabling anyone to copy, modify and redistribute the source code without paying royalties or fees. Open source code evolves through community cooperation. These communities are composed of individual programmers as well as very large companies. Some examples of open source initiatives are Linux, Eclipse, Apache, Mozilla, and various projects hosted on SourceForge.

**Free software :** Is the terms that are roughly equivalent to Open Source. The term "free" is meant to describe the fact that the process is open and accessible and anyone can contribute to it. "Free" is not meant to imply that there is no charge. "Free software" may be packaged with various features and services and distributed for a fee by a private company. The term "public domain" software is often erroneously used interchangeably with the term "free software" and "open source" software. In fact, "public domain" is a legal term that refers to software whose

copyright is not owned by anyone, either because it has expired, or because it was donated without restriction to the public. Unlike open source software, public domain software has no copyright restrictions at all. Any party may use or modify public domain software.

**Commercial software :** Is the software that is distributed under commercial license agreements, usually for a fee. The main difference between the commercial software license and the open source license is that the recipient does not normally receive the right to copy, modify, or redistribute the software without fees or royalty obligations. Many people use the term "proprietary software" synonymously with "commercial software." Because of the potential confusion with the term "proprietary" in the context of standards and interfaces, and because commercial software may very well implement open, non-proprietary interfaces, this article will use the term "commercial software" to refer to non-open source software (Heintzman D., 2003).

### 7.2. Why Open Source?

The most obvious boon of open source to software developers is the opportunity to base a design on existing software elements. The open source community gives us a rich base of reusable software, typically available at the cost of downloading the code from the Internet. So, in many cases we can select best code to reuse in our system without having to reinvent the wheel. The resulting products benefit in two ways. First, the reused open source code will typically be of higher quality than the custom-developed code's first incarnation. Second, the functionality the reused element offers will often be far more complete than what the bespoke development would afford (Spinellis D. and Szyperski C. , 2004),.

Moreover, reuse granularity is not restricted by the artificial product boundaries of components distributed in binary form (which marketing considerations often impose). When reusing open source, code adoption can happen at the level of a few lines of code, a method, a class, a library, a component, a tool, or a complete system. Furthermore, when software is available in source code form, we can more easily port to our target platform and adjust its interfaces to suit our needs.

Consequently, software reuse possibilities open up on three axes: what to reuse (promoted by the available software's breadth and price), how to reuse it (diverse granularity and interfacing options), and where to reuse it (inherent portability of source code over most binary packaged component technologies). Movement along all three axes increases the breadth of software reuse opportunities in any development effort.

In addition, source code's availability lets us perpetually improve, fix, and support the reused elements. This factor often mitigates the risk of orphaned components or incompatible evolution paths that are associated with the reuse of proprietary components. Also, by incorporating the source code of a reused element into the system being built, developers can achieve tight integration and a system that can be maintained as a whole.

### 7.3. When Open Source?

Before deciding to use open source, some the conditions must be considered. An open source software

- should meet the requirements
- should support by large community
- should be sure continuity
- should be examine for performance issue
- should be documented, published, and reviewed in source code form
- should be discussed, internalized, generalized, and paraphrased
- should used for solving real problems, often in conjunction with other programs

### 7.4. JBoss Rule Engine

At the start of the study, an investigation was made for rule engines. Our concept is , it should be open source java software and meet the conditions on section “When Open Source?”. At the same time rule engine should obey open standards JSR-94 (Java Specification Requests, which are formal documents that describe proposed specifications and technologies to be added to the Java platform). After this investigation two alternatives were found; JESS and Drools.

On the other hand JESS was not fully open source software, but for academic usage required permission was possible. However, JESS structure is very complicated and not so suitable for java implementation. So, Drools was best alternative for our work.

Initial implementation was made by Drools 2.1. After this time, Drools Rule Engine is acquired by JBoss. This trade was proof the power of the Drools, because JBoss one of the most important open source software constitution. JBoss products are using many production environments, and now its rule engine is Drools(after here ,both JBoss Rule Engine and Drools Rule Engine are used in the same meaning).

One drawback of this trade is; knowledge representation and implementation was slightly changed, and its version was Drools 3.x. So our initial works were reimplemented.

Drools is an "augmented implementation of Charles Forgy's Rete algorithm. Rete algorithm is a popular approach to Forward Chaining, tailored for the Java language". Drools has implementations for both Rete and Leaps. The Drools Rete implementation is called ReteOO signifying that Drools has an enhanced and optimized implementation of the Rete algorithm for Object Oriented systems.

In summary, open source business rule management systems might make more sense then their expensive commercial counterparts. JBoss Rules and Jess represent two of the better open source offerings out on the market. In this thesis, JBoss rule engine is chosen for the below reasons:

It has

- A very active community
- Easy to use
- Fast execution speed
- Gaining popularity among Java developers
- JSR 94-compliant (JSR 94 is the Java Rule Engine API)
- Free

After this summary about history of our rule engine works, JBoss rule engine architecture and its components should be explained.

## 7.5. Architecture Of JBoss Rule Engine

Drools is split into two main parts Authoring and Runtime.

### 7.5.1. Authoring

The authoring process involves the creation of drl or xml files for rules which are fed into a parser. The parser checks for correctly formed grammar and produces an intermediate structure, then passed to the Package Builder which produces Packages. Package Builder also undertakes any code generation and compilation that is necessary for the creation of the Package. A Package object is a self contained and deployable, in that it's serializable, object consisting of one or more rules (Proctor et al., 2006).

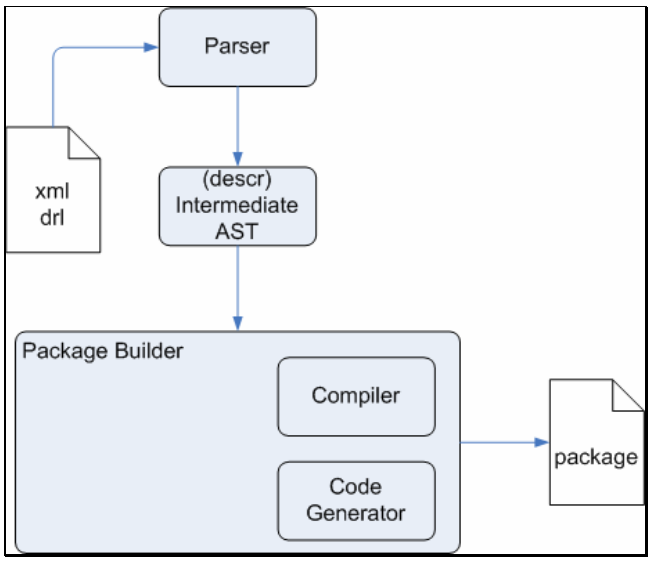



Figure 7.1 JBoss authoring
(Proctor et al., 2006)

### 7.5.2. Runtime

A RuleBase is a runtime component which consists of one or more Package's. Packages can be added and removed from the RuleBase at any time. A Rule Base can instantiate one or more Working Memories at any time; a weak reference is maintained, unless it's told otherwise. The Working Memory consists of a number of sub components including Working Memory Event Support, Truth Maintenance System, Agenda and Agenda Event Support. Object assertion may result in the creation of one or more Activations, the agenda is responsible for scheduling the execution of these Activations (Proctor et al., 2006).

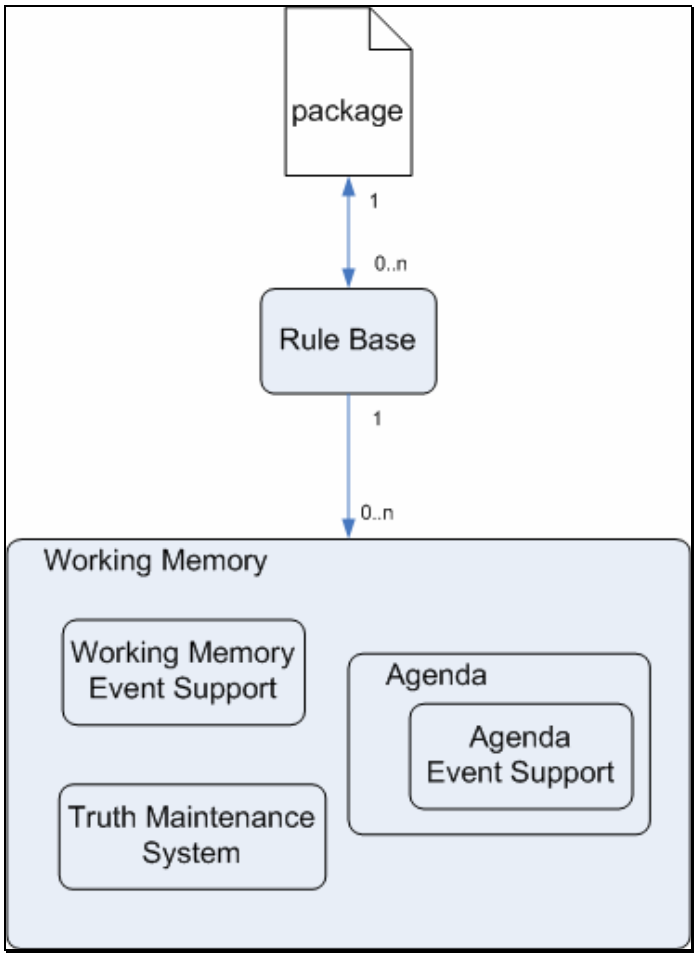



Figure 7.2 JBoss runtime
(Proctor et al., 2006)

### 7.5.3. Drools Rule Base

A Rule Base contains one more packages of rules, ready to be used (i.e. they have been validated/compiled etc). A Rule Base is serializable so it can be deployed to JNDI, or other such services. Typically, a rule base would be generated and cached on first use; to save on the continually re-generation of the Rule Base; which is expensive (Proctor et al., 2006).

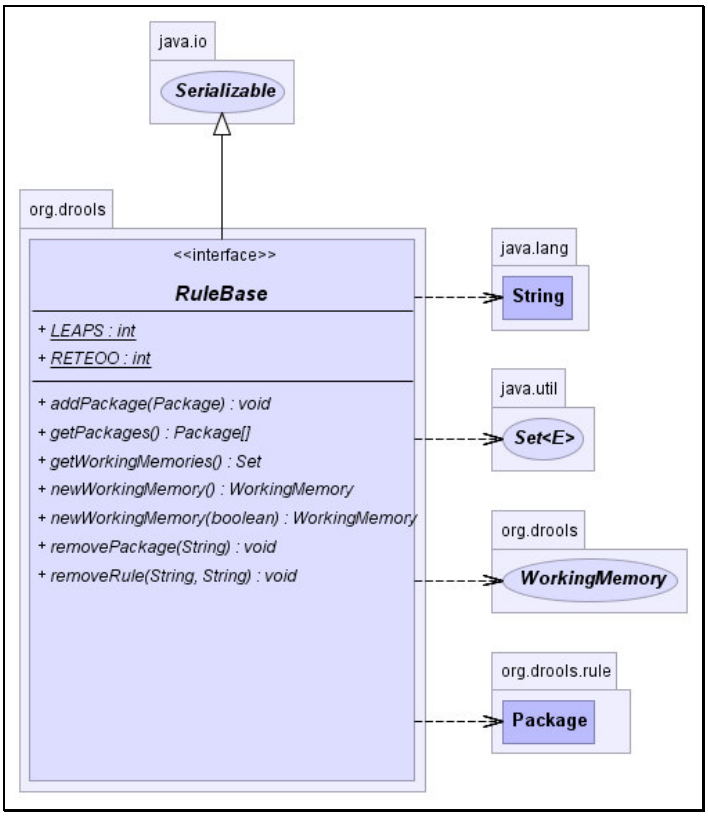



Figure 7.3 Drools rule base
(Proctor et al., 2006)

### 7.5.4. Drools Working Memory

The Working Memory is the main Class for using the Rule Engine at runtime. It holds references to all data that has been "asserted" into it (until retracted) and it is the place where the interaction with your application occurs. Working memories are stateful objects. They may be shortlived, or longlived (Proctor et al., 2006).

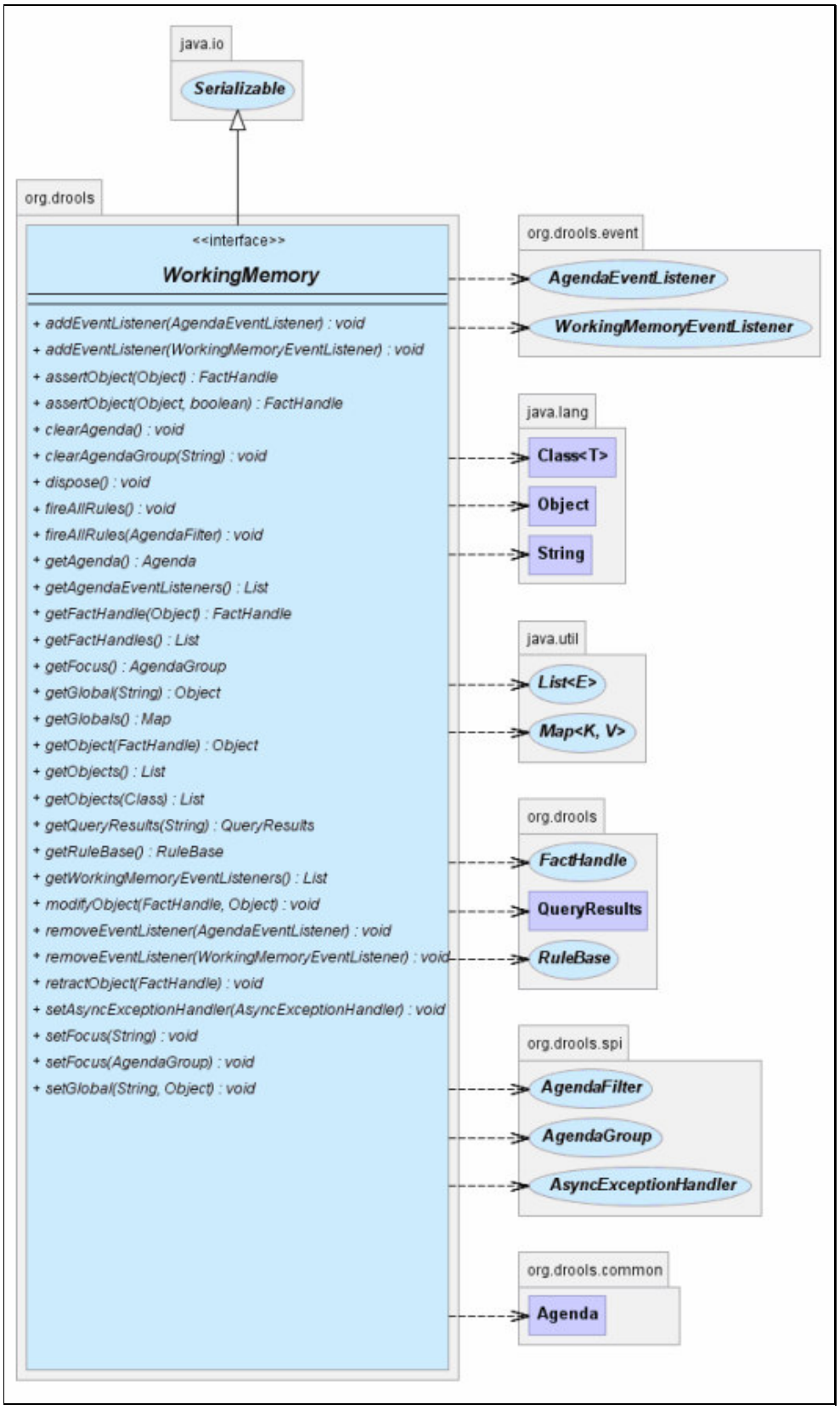



Figure 7.4 Drools Working Memory
(Proctor et al., 2006)

### 7.5.5. Knowledge Representation

#### 7.5.5.1. Rules

A Production Rule, or Rule, in Drools is a two part structure with a Left Hand Side (LHS) and a Right Hand Side (RHS). Additionally a rule may have the following attributes:

- salience
- agenda-group
- auto-focus
- activation-group
- no-loop
- duration

```
if ( <LHS> ) {
    <RHS>
}
```

```
rule "<name>"
    <attribute> <value>
    when
        <LHS>
    then
        <RHS>
end
```

Figure 7.5 Procedural IF and drools rule

The LHS of a Rule consists of Conditional Elements (CE) and Columns; to run the encoding of propositional and first order logic. The term Column is used to indicate Field Constraints on a Fact (Proctor et al., 2006).

#### 7.5.5.2. Facts

Facts are objects (beans) from your application that you assert into the working memory. Facts are any java objects which the rules can access. The rule engine does not "clone" facts at all, it is all references/pointers at the end of the day. Facts are applications data. Strings and other classes without getters and setters are not valid Facts and can't be used with Field Constraints which rely on the JavaBean standard of getters and setters to interact with the object (Proctor et al., 2006).

### 7.5.6. The Rule Language

Drools 3 has a "native" rule language that is non XML textual format. This format is very light in terms of punctuation, and supports natural and domain specific languages via "expanders" that allow the language to morph to your problem domain.

A rule file is typically a file with a .drl extension. In a drl file you can have multiple rules, functions etc. However, rules can be spread across multiple rule files. Spreading rules across files can help with managing large numbers of rules. A DRL file is simply a text file.

Domain specific languages are implemented as an enhancement over the native rule language. They use the "expander" mechanism. The expander mechanism is an extensible API, but by default it can work with .dsl files, which contain mappings from the domain or natural language to the rule language and your domain objects.

As an option, Drools also supports a "native" rule language as an alternative to DRL. This allows to capture and manage the rules as XML data. Just like the non-XML DRL format, the XML format is parsed into the internal "AST" representation - as fast as possible (using a SAX parser). There is no external transformation step required. All the features are available with XML that are available to DRL (Proctor et al., 2006).

## 7.6. Using Drools (Simple Example)

In this chapter, a simple example will explained for giving answer the question "how drools rule engine works". This is classical hello world example which is a simple java class. It's full name is gp.tez.jbossrule.HelloWorldExample.

This example simple get messages, print to system out, modify the message and reprint modified message to the system out. For this operation two rules are written to the HelloWorld.drl file (Figure 7.6).

```
package gp.tez.jbossrule

import gp.tez.jbossrule.Message;

rule "Hello World"
        when
                m : Message( status == Message.HELLO, message : message )
        then
                System.out.println( message );
                m.setMessage( "Goodbye cruel world" );
                m.setStatus( Message.GOODBYE );
                modify( m );
end

rule "GoodBye"
        no-loop true
        when
                m : Message( status == Message.GOODBYE, message : message )
        then
                System.out.println( message );
                m.setMessage( message );
end
```

Figure 7.6 HelloWorld.drl file

To run a drools application, some java libraries must be classpath. This list of jar files can be seen in Figure 7.7.

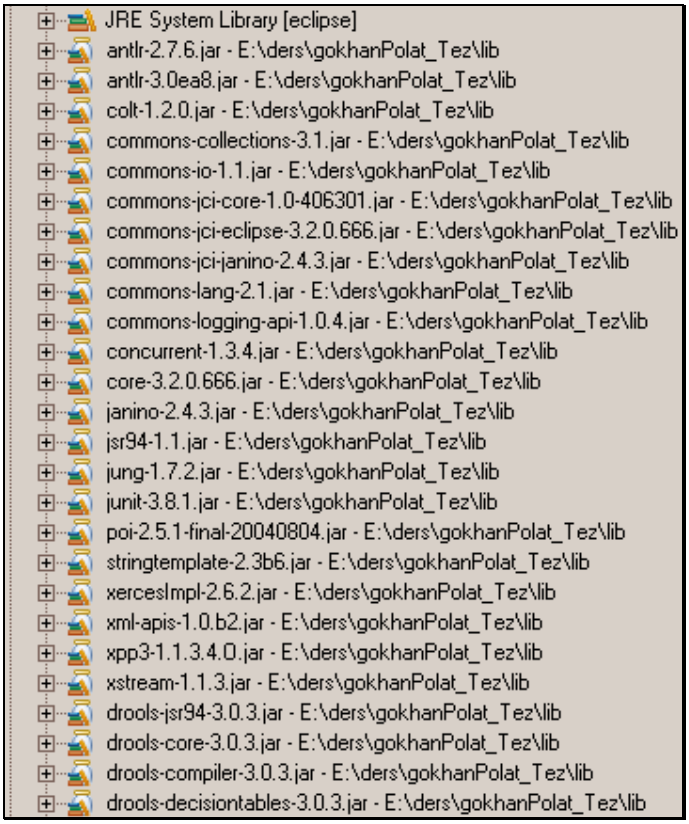



Figure 7.7 Required library for Drools

After adding libraries, required classes should be imported (Figure 7.8).

```
import java.io.InputStreamReader;
import java.io.Reader;

import org.drools.RuleBase;
import org.drools.RuleBaseFactory;
import org.drools.WorkingMemory;
import org.drools.compiler.PackageBuilder;
import org.drools.rule.Package;
```

Figure 7.8 Importing Packages

At this point a new rule base can be created for the our rule set in the HelloWorld.drl file.

```
private static RuleBase readRule() throws Exception {
    //read in the source
    final Reader source = new InputStreamReader(HelloWorldExample.class
            .getResourceAsStream(drlName));

    //optionally read in the DSL (if you are using it).
    //Reader dsl = new InputStreamReader(
    // DroolsTest.class.getResourceAsStream( "/mylang.dsl" ) );

    //Use package builder to build up a rule package.
    //An alternative lower level class called "DrlParser" can also be
    // used...

    final PackageBuilder builder = new PackageBuilder();

    //this wil parse and compile in one step
    //NOTE: There are 2 methods here, the one argument one is for normal
    // DRL.
    builder.addPackageFromDrl(source);

    //Use the following instead of above if you are using a DSL:
    //builder.addPackageFromDrl( source, dsl );

    //get the compiled package (which is serializable)
    final Package pkg = builder.getPackage();

    //add the package to a rulebase (deploy the rule package).
    final RuleBase ruleBase = RuleBaseFactory.newRuleBase();
    ruleBase.addPackage(pkg);
    return ruleBase;
}
```

Figure 7.9 Returning a rule base

A new working memory is created for the rule base, “Message” object (Figure 7.11) is asserted to this working memory and rules fired.

```
public static final void main(final String[] args) {
    try {

        drlName = System.getProperty("drlName");

        //load up the rulebase
        final RuleBase ruleBase = readRule();
        final WorkingMemory workingMemory = ruleBase.newWorkingMemory();

        //go !
        final Message message = new Message();
        message.setMessage( "Hello World" );
        message.setStatus( Message.HELLO );
        workingMemory.assertObject( message );
        workingMemory.fireAllRules();

    } catch ( final Throwable t ) {
        t.printStackTrace();
    }
}
```

Figure 7.10 Main part of the HelloWold example

```
package gp.tez.jbossrule;

public class Message {
    public static final int HELLO   = 0;
    public static final int GOODBYE = 1;

    private String          message;

    private int             status;

    public String getMessage() {
        return this.message;
    }

    public void setMessage(final String message) {
        this.message = message;
    }

    public int getStatus() {
        return this.status;
    }

    public void setStatus(final int status) {
        this.status = status;
    }
}
```

Figure 7.11 Message object

## 8. STANDARDIZATION OF RULE ENGINE

The specification defines a Java API for rule engines. The API prescribes a set of fundamental rule engine operations. The set of operations is based on the assumption that most clients need to be able to execute a basic multiple-step rule engine cycle that consists of parsing rules, adding objects to an engine, firing rules, and getting resultant objects from the engine.

This new API gives developers a standard way to access and execute rules at runtime. As implementations of this new spec ripen and are brought to the market, programming teams will be able to pull executive logic out of their applications.

JSR 94 defines a simple API to access a rule engine from a Java SE or Java EE client. It provides APIs to

- Register and unregister rules
- Parse rules
- Inspect rule metadata
- Execute rules
- Retrieve results
- Filter results

Note that JSR 94 does not standardize the following:

- The rule engine itself
- The execution flow for rules
- The language used to describe the rules
- The deployment mechanism for Java EE technology

In other words, it doesn't standardize the semantics of rule execution (Mahmoud Q. H.,2005).

The goals of the specification are to:

- Facilitate adding rule engine technology to Java applications.
- Increase communication and standardization between rule engine vendors.
- Encourage the creation of a market for third-party application and tool vendors through a standard rule engine API.
- Facilitate embedding rule engine technology in other JSRs to support declarative programming models.
- Promote independence of client code from J2SE environment.
- Make Java applications more portable from one rule engine vendor to another.
- Provide implementation patterns for rules-based applications for the J2SE platform.
- Support rule engine vendors by offering a harmonized API that meets the needs of their existing customers and is easily implemented.

## 8.1. Architecture Of JSR-94

The interfaces and classes defined by the specification are in the javax.rules and javax.rules.admin packages. The javax.rules package contains classes and interfaces that are aimed at "runtime clients" of the rule engine. The runtime client API exposes methods to acquire a rule session for a registered rule execution set and interact with the rule session. The administrator API exposes methods to load an execution set from these external resources: URI, InputStream, XML Element, binary abstract syntax tree, or Reader. The administrator API also provides methods to register and unregister rule execution sets. Only registered rule execution sets are accessible through the runtime client API (Toussaint A., 2003).

### 8.1.1. Runtime API

The runtime API for the specification is defined in the `javax.rules` package. The high-level capabilities of the runtime API are (Toussaint A., 2003):

- Acquire an instance of a rule engine vendors RuleServiceProvider interface through the RuleServiceProviderManager class.

- Acquire an instance of the RuleRuntime interface through the RuleServiceProvider class.
- Create a RuleSession through the RuleRuntime.
- Get a java.util.List of registered URIs.
- Interact with an acquired RuleSession.
- Retrieve metadata for a RuleSession through the RuleExecutionSetMetadata interface.
- Provide an ObjectFilter interface to filter the results of executing a RuleExecutionSet.
- Use Handle instances to access objects added to a StatefulRuleSession.

### 8.1.2. Rules Administrator API

The administrator API for the specification is defined in the javax.rules.admin package. The high-level capabilities of the administrator API are (Toussaint A., 2003):

- Acquire an instance of the RuleAdministrator interface through theRuleServiceProvider class.
- Create a RuleExecutionSet from external Serializable or non-Serializable resources, as listed below:
    - org.w3c.dom.Element . for reading from an XML sub-document.
    - java.io.InputStream . for reading from binary streams.
    - java.lang.Object . for reading from vendor specific abstract-syntax-trees.
    - java.io.Reader . for reading from character streams.
    - java.lang.String . for reading from a URI.
- Register a RuleExecutionSet object against a URI for use from the RuleRuntime. Registrations should be persistent and the rule engine vendor should clearly document the scope of a registration.
- Deregister a RuleExecutionSet object from a URI so it is no longer accessible from the RuleRuntime.
- Query the structural metadata of a RuleExecutionSet by retrieving a list of Rule objects from the RuleExecutionSet.
- Set and get application or vendor specific properties on RuleExecutionSets and Rules.

## 8.2. Using JSR-94 With Drools (Simple Example)

In this section, JSR-94 compliant HelloWorld example will be explained. To work with this simple example, two more parameters are required. First of them is "*rule service provider*" which is "**http://drools.org/**" in our example. And other one is "*provider class*" which is "**org.drools.jsr94.rules.RuleServiceProviderImpl**" for drools rule engine.

Main part of this implementation is shown in Figure 8.1. Three external attributes are given as a system property.

```
public static final void main(final String[] args) {
    try {

        //load up the rulebase
        HelloWorldJSR94 hw = new HelloWorldJSR94();
        hw.setDrlName(System.getProperty("drlName"));
        hw.setRULE_SERVICE_PROVIDER(System.getProperty("RuleProvider"));
        hw.setDROOLS_RULE_SERVICE_PROVIDER_CLASS(System
                .getProperty("ImpClass"));
        hw.fireRule();

    } catch (final Throwable t) {
        t.printStackTrace();
    }
}
```

Figure 8.1 Main part of JSR-94 compliant example

After this, fireRule method is called. It's content is in Figure 8.2.

By changing the parameters, any different JSR-94 compliant rule engine can be used. This can be summarized the importance of the JSR-94.

```
private void fireRule() throws Exception {
    System.out.println(getDROOLS_RULE_SERVICE_PROVIDER_CLASS());
    Class rspi = Class.forName(getDROOLS_RULE_SERVICE_PROVIDER_CLASS());
    RuleServiceProviderManager.registerRuleServiceProvider(
            getRULE_SERVICE_PROVIDER(), rspi);
    Class.forName(getDROOLS_RULE_SERVICE_PROVIDER_CLASS());

    System.out.println("register rule service provider.. ");
    RuleServiceProvider ruleServiceProvider = RuleServiceProviderManager
            .getRuleServiceProvider(getRULE_SERVICE_PROVIDER());
    System.out.println("get the RuleAdministrator.. ");
    //get the RuleAdministrator
    RuleAdministrator ruleAdministrator = ruleServiceProvider
            .getRuleAdministrator();
    System.out.println("InputStream.. ");
    InputStream rules = (InputStream) new FileInputStream(getDrlName());
    System.out.println("Loaded ruleSetProvider: ");
    LocalRuleExecutionSetProvider ruleSetProvider = ruleAdministrator
            .getLocalRuleExecutionSetProvider(null);
    System.out.println("Loaded RuleExecutionSet: ");
    RuleExecutionSet ruleExecutionSet = ruleSetProvider
            .createRuleExecutionSet(rules, null);
    System.out.println("registerRuleExecutionSet  for " + getDrlName());
    ruleAdministrator.registerRuleExecutionSet(getDrlName(),
            ruleExecutionSet, null);
    RuleServiceProvider serviceProvider = RuleServiceProviderManager
            .getRuleServiceProvider(getRULE_SERVICE_PROVIDER());
    //       create a stateless RuleSession
    RuleRuntime ruleRuntime = serviceProvider.getRuleRuntime();
    StatelessRuleSession srs = (StatelessRuleSession) ruleRuntime
            .createRuleSession(getDrlName(), null,
                    RuleRuntime.STATELESS_SESSION_TYPE);

    //       execute all the rules
    List inputList = new LinkedList();
    final Message message = new Message();
    message.setMessage("Hello World");
    message.setStatus(Message.HELLO);
    inputList.add(message);
    List resultList = srs.executeRules(inputList);
    //       release the session
    srs.release();

}
```

Figure 8.2  Hello world JSR-94 example

# 9. A FRAMEWORK FOR A LEARNING EXPERT SYSTEM

A framework is an extensible semi-finished piece of software that represents a generic solution to a set of applications in a specific domain. A framework constitutes an ever-volving representation of our knowledge of the domain in terms of variations and commonalties. A very important point is that the framework design should not start by trying to model its variability and flexibility at once. Instead, a fixed application should be designed from the framework domain and generalize it only when the fixed case is understood.

This thesis suggests a framework for a rule base learning expert system. It takes a data set, employs some learning algorithms, constructs a rule base, and presents a web based interface.

## 9.1. Requirements For The Framework

Each framework has some default assumptions. These assumptions can be defined as requirements of the framework. To employ framework, all requirements should be satisfied. In this framework, requirements are given:

- JDK 1.5.x should be installed
- Apache Tomcat 5.5.x should be installed
- All libraries must be on the classpath of the Tomcat
- C:/ESM/ should exist as a “project folder”
- A class should be created as a fact target class which acts as a bean. Contains attributes and their getter and setter methods.
- Attributes of the targetClass must be String.
- Target Class must be on "ClassPath"
- "targetClassName".properties file also must be located under the "project folder" Should containbelow property pairs ;
    - targetClass=..
    - DROOLS_RULE_SERVICE_PROVIDER_CLASS=..
    - RULE_SERVICE_PROVIDER=..
- Data Set file must be start with "targetClassName_"
- Attribute name’ first letter should be capital.

- Class attribute's name must start with "Class"
- Distinct attributes file must start with "targetClassName_" and end with "distinctValues"
- Distinct attributes file must contains target java class full name
- rule file (*.drl) must be under the "project folder"
- rule file (*.drl) must like "targetClassName".drl

## 9.2. Architecture of the Framework

In this framework java technologies are used. There are mainly two parts. First of them is rule base construction part, and other is web interface part. Folder structures and java classes are shown in Figure 9.1. Their javadocs and jsp files can be found in Appendix I and II.

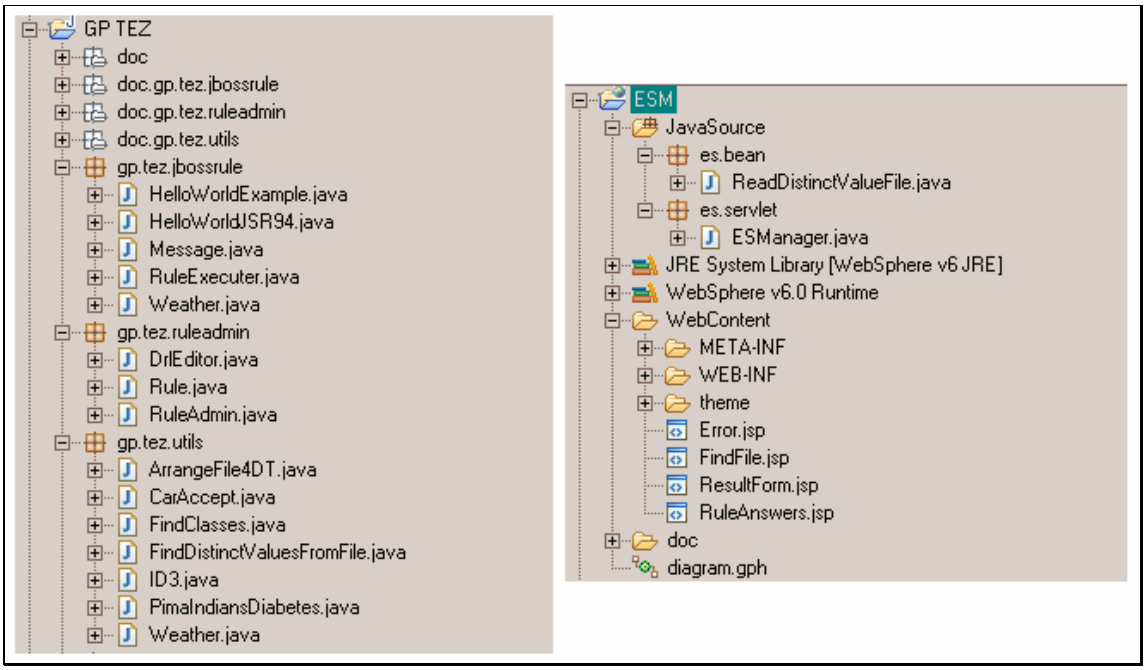



Figure 9.1 Java based projects for the framework

### 9.2.1. Constructing Rule Base

First operation is getting data set. This data set should meet mentioned decision tree requirements. For example, its attributes should not be continuous. For continuous attributes a

small implementation is developed which is name ArrangeFile4DT. But this is just an optional helper tool. Our main focus is decision tree compliant data sets.

Valid form of a data set can be found in Figure 9.2. First line of the data set should be names of the attributes. Last column represent class attribute. All attribute values should be separated with a comma.

```
Outlook Tempreature Humidity Windy Class
sunny,hot,high,false,N
sunny,hot,high,true,N
overcast,hot,high,false,P
rain,mild,high,false,P
rain,cool,normal,false,P
rain,cool,normal,true,N
overcast,cool,normal,true,P
sunny,mild,high,false,N
sunny,cool,normal,false,P
rain,mild,normal,false,P
sunny,mild,normal,true,P
overcast,mild,high,true,P
overcast,hot,normal,false,P
rain,mild,high,true,N
```

Figure 9.2 Content of the Weather_DataSet.txt

This valid data set can be processed by ID3 java implementation. This implementation presents three forms of rule sets. The first of them is if-then-else form Figure 9.3. This is the original output of the Dr.Benny Raphael ID3 java implementation.

```
if( Outlook == "sunny") {
        if( Humidity == "high") {
                Class = "N";
        } else  if( Humidity == "normal") {
                Class = "P";
        }
} else if( Outlook == "overcast") {
            Class = "P";
} else if( Outlook == "rain") {
        if( Windy == "false") {
                Class = "P";
        } else  if( Windy == "true") {
                Class = "N";
        }
}
```

Figure 9.3 If-then-else form of ID3 output

The other one is the rule list form which is one of the our extension of this implementation (Figure 9.4). This form is mainly used for middle step for controlling output.

```
1 Outlook equals "sunny" AND Humidity equals "high"  ==> "N"
2 Outlook equals "sunny" AND Humidity equals "normal"  ==> "P"
3 Outlook equals "overcast"  ==> "P"
4 Outlook equals "rain" AND Windy equals "false"  ==> "P"
5 Outlook equals "rain" AND Windy equals "true"  ==> "N"
```

Figure 9.4 Rule list form of ID3 output

The last one is the drl file format of the ID3 output (Figure 9.5). This is our focus for this study. This file will be used on the inference mechanism which is explained in the JBoss rule engine section. ID3 implementation constructs all rules about data set and writes into a drl file in convenient format.

```
package gp.tez.utils
import gp.tez.utils.*;

rule "1"
        when
obj : Weather (outlook == "sunny" , humidity == "high" )
        then
obj.setResult("N");
end

rule "2"
        when
obj : Weather (outlook == "sunny" , humidity == "normal" )
        then
obj.setResult("P");
end

rule "3"
        when
obj : Weather (outlook == "overcast" )
        then
obj.setResult("P");
end

rule "4"
        when
obj : Weather (outlook == "rain" , windy == "false" )
        then
obj.setResult("P");
end

rule "5"
        when
obj : Weather (outlook == "rain" , windy == "true" )
        then
obj.setResult("N");
end
```

Figure 9.5 drl output of ID3 output

Next part of the framework is obtaining distinct values of the data set. This process is the linkage of the rule base construction part and the web interface part. Its output is used for web based interface and constitutes web page for testing new cases. Two examples of the framework's distinct values file can be found in Figure 9.6.

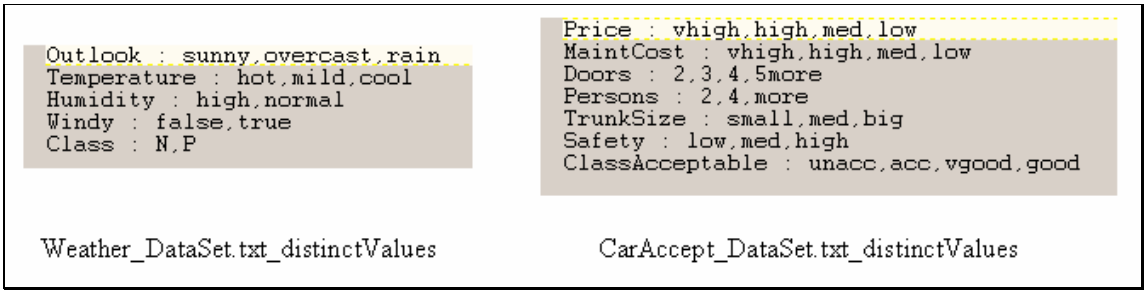



Figure 9.6 Distinct values files

### 9.2.2. Web Based Interface

After getting distinct values file, web interface file can be used. It's starting point is this file. It simply gets all attribute names and their possible values, and builds a page for selecting them for a new case.

After submitting form, it calls rule engine routines. Because of this, all libraries must be on classpath. Then resulting page presents to the user.

In this study Apache Tomcat 5.5 is used as open source servlet engine. But this part is optional. Because it is standard web application, it can be run any java based application server like WebSphere or Weblogic.

#### 9.2.2.1. RuleExecuter

One of the most important phase for inference mechanism is getting rule in a standardized way and sending it to the rule engine. By means of java reflection technology, this goal can be achieved in this method (Figure 9.7 and Figure 9.8).

```
/**
 *
 * fire rule by means of normal jbossrule procedure
 *
 */

public void fireRule() {

    try {

        //load up the rulebase
        ruleBase = readRule();
        workingMemory = ruleBase.newWorkingMemory();
        workingMemory.assertObject(getObject());
        workingMemory.fireAllRules();

    } catch (final Throwable t) {
        t.printStackTrace();
    }
}

public RuleBase readRule() {
    try {
        //read in the source
        URL url = new URL("file:/" + drlName);
        InputStreamReader reader = new InputStreamReader(url.openStream());
        final PackageBuilder builder = new PackageBuilder();
        //this will parse and compile in one step
        //NOTE: There are 2 methods here, the one argument one is for
        // normal DRL.
        builder.addPackageFromDrl(reader);
        //get the compiled package (which is serializable)
        final Package pkg = builder.getPackage();
        //add the package to a rulebase (deploy the rule package).
        final RuleBase ruleBase = RuleBaseFactory.newRuleBase();
        ruleBase.addPackage(pkg);
        return ruleBase;
    } catch (Throwable e) {
        e.printStackTrace();
        return null;
    }
}
```

Figure 9.7 RuleExecuter method I

```
public void fireRuleJSR94() {

    try {
        System.out.println(getDROOLS_RULE_SERVICE_PROVIDER_CLASS());
        Class rspi = Class.forName(getDROOLS_RULE_SERVICE_PROVIDER_CLASS());
        RuleServiceProviderManager.registerRuleServiceProvider(
                getRULE_SERVICE_PROVIDER(), rspi);
        Class.forName(getDROOLS_RULE_SERVICE_PROVIDER_CLASS());

        System.out.println("get rule service ");
        //       Register the driver
        System.out.println("register rule service.. ");
        RuleServiceProvider ruleServiceProvider = RuleServiceProviderManager
                .getRuleServiceProvider(getRULE_SERVICE_PROVIDER());
        //       load the rules and register them
        System.out.println("get the RuleAdministrator.. ");
        //get the RuleAdministrator
        RuleAdministrator ruleAdministrator = ruleServiceProvider
                .getRuleAdministrator();
        System.out.println("InputStream.. ");
        InputStream rules = (InputStream) new FileInputStream(getDrlName());
        System.out.println("Loaded ruleSetProvider: ");
        LocalRuleExecutionSetProvider ruleSetProvider = ruleAdministrator
                .getLocalRuleExecutionSetProvider(null);
        System.out.println("Loaded RuleExecutionSet: ");
        RuleExecutionSet ruleExecutionSet = ruleSetProvider
                .createRuleExecutionSet(rules, null);
        System.out.println("registerRuleExecutionSet  for " + getDrlName());
        ruleAdministrator.registerRuleExecutionSet(getDrlName(),
                ruleExecutionSet, null);
        RuleServiceProvider serviceProvider = RuleServiceProviderManager
                .getRuleServiceProvider(getRULE_SERVICE_PROVIDER());
        //       create a stateless RuleSession
        RuleRuntime ruleRuntime = serviceProvider.getRuleRuntime();
        StatelessRuleSession srs = (StatelessRuleSession) ruleRuntime
                .createRuleSession(getDrlName(), null,
                        RuleRuntime.STATELESS_SESSION_TYPE);
        //       execute all rules
        List inputList = new LinkedList();
        inputList.add(getObject());
        List resultList = srs.executeRules(inputList);
        //       release the session
        srs.release();
    } catch (Throwable e) {
        e.printStackTrace();
    }

}
```

Figure 9.8  RuleExecuter method II

### 9.3. Framework Example 1

First example is classic weather/Play tennis data set. Weather data set has 14 lines and their attributes are “Outlook”, “Temperature”, ”Humidity” and “Windy”. Attribute’s possible values can be viewed in Figure 9.6. In the first page of the web interface weather data set’s distinct values option should be chosen (Figure 9.9).

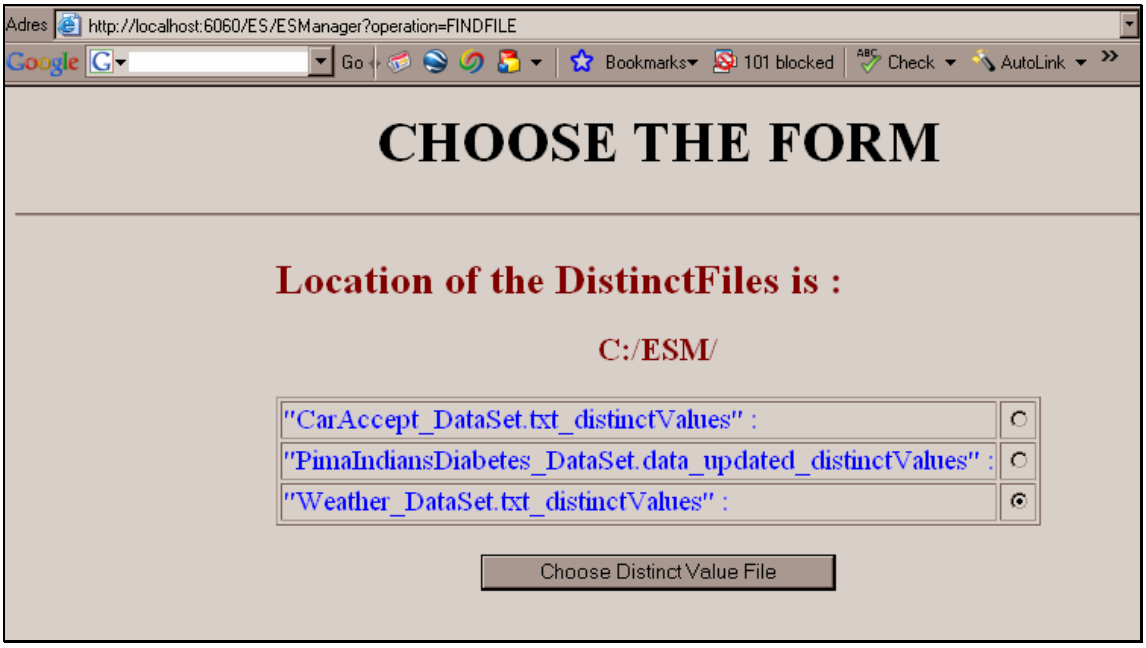



Figure 9.9 Example 1 choosing data set

After choosing distinct file, interface reads the attributes and their possible values, then constructs a question form (Figure 9.10). With help of this form, we can try new options for the attributes.

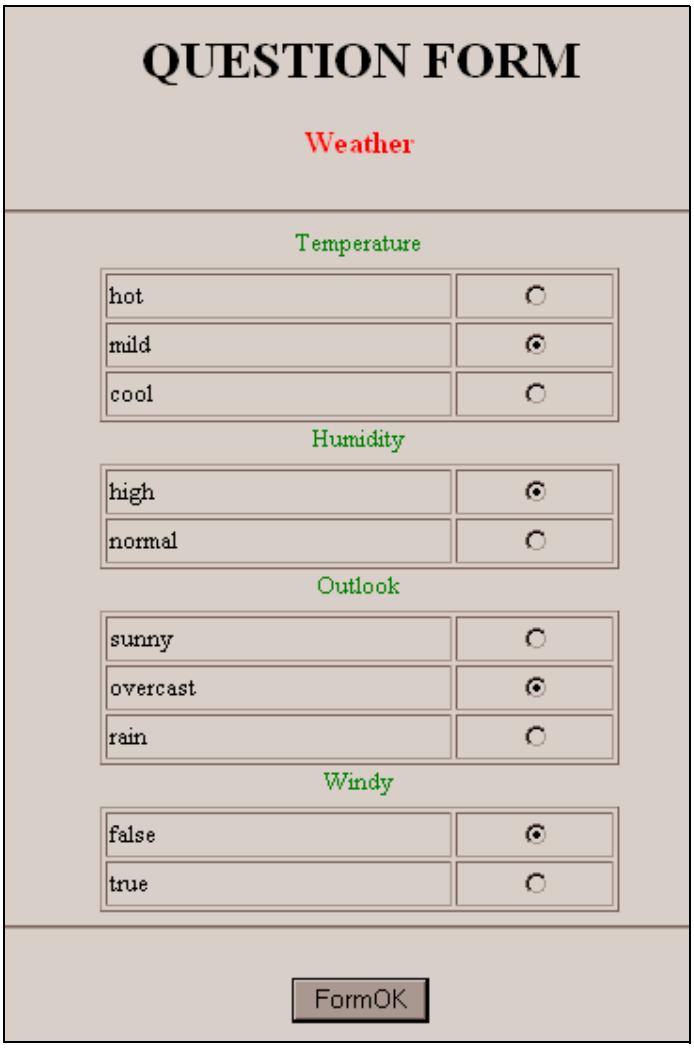



Figure 9.10 Example 1 testing new case

After setting all attributes and submitting form, interface starts inference mechanism for given attribute values. At the end of process, form result page is constructed. In this form selected attributes and result are displayed (Figure 9.11).

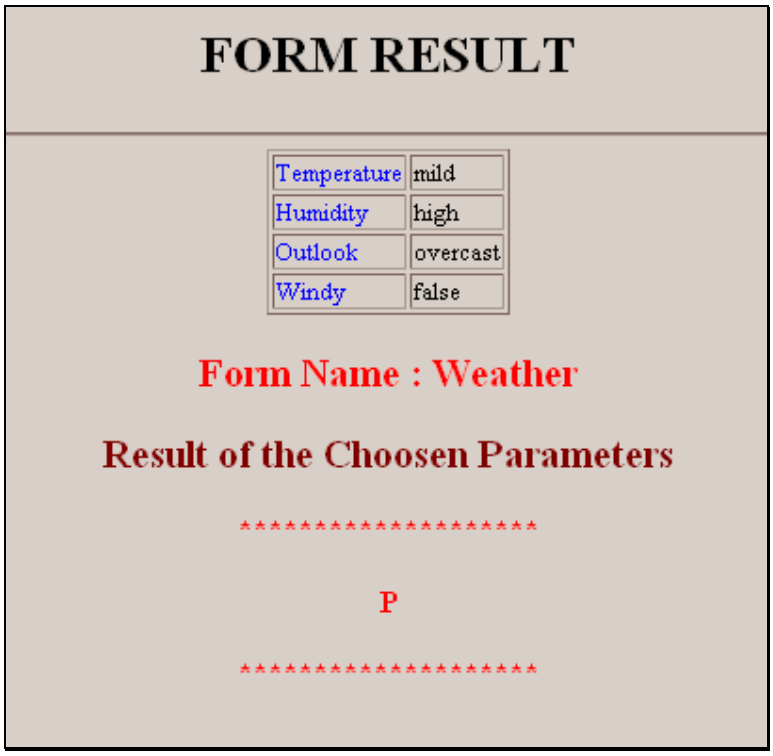



Figure 9.11 Example 1 result page

### 9.4. Framework Example 2

A framework should work with different inputs. To prove that, an other example is needed. For this reason CarAccept data set will be used. This data set is bigger than first data set. It has 1728 lines. Attributes of this data set are “Price”, “MaintCost”, “Doors”, “Persons”, “TrunkSize” and “Safety”. Their possible values are declared in Figure 10.5.

Other main difference of this example from first one is, its class values are “unacceptable” (unacc), “acceptable” (acc), “very good” (vgood) and “good” (good), totally four possible cases. At the first example, there was only two class variables; play or not play.

Screenshots of this example can be viewed in Figure 9.12, Figure 9.13 and Figure 9.14.

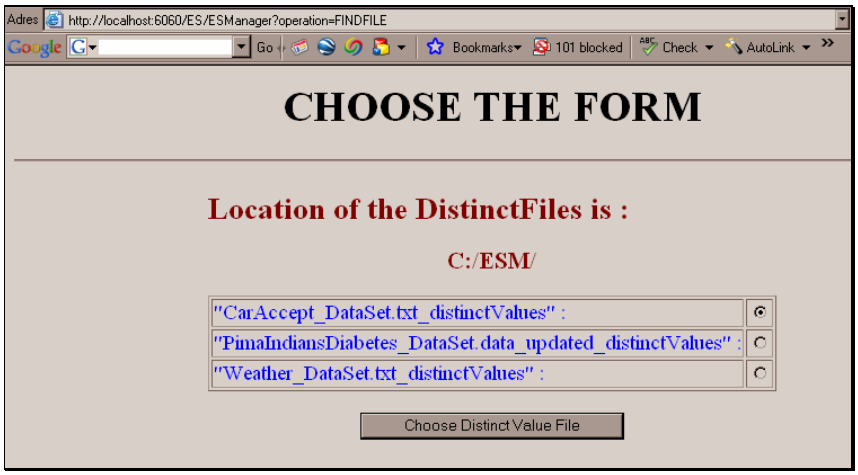



Figure 9.12 Example 2 choosing data set

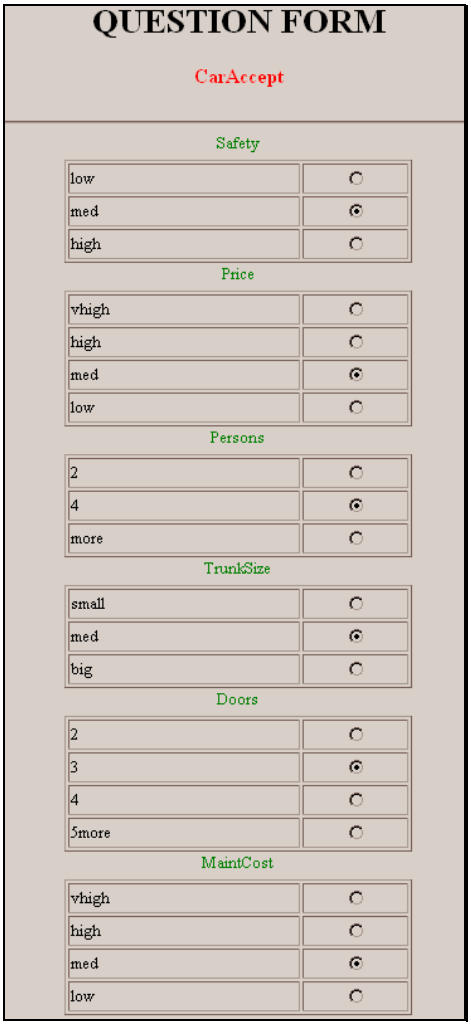



Figure 9.13 Example 2 testing new case

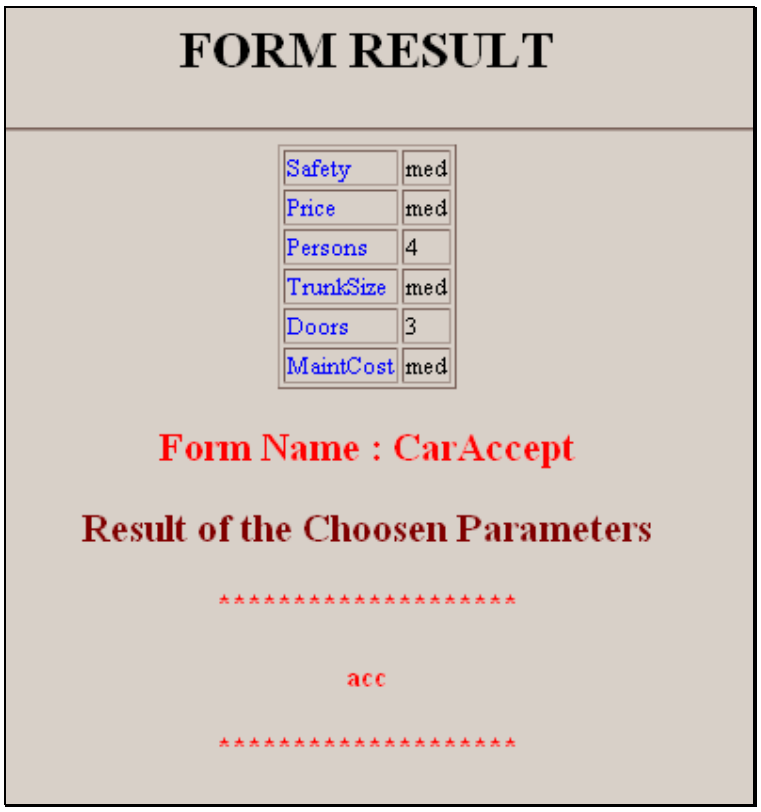



Figure 9.14 Example 2 result page

## 10. RULE DECLARATION FILE (DRL FILES) EDITOR

To complete the study as a framework one more thing is needed. In some circumstances rule file could be not sufficient or not fit the real life exactly. For this cases rule file should be edited by a professional user. This user should be aware of JBoss rule language. For this reason a tool is developed as swing based java application (Figure 10.1).

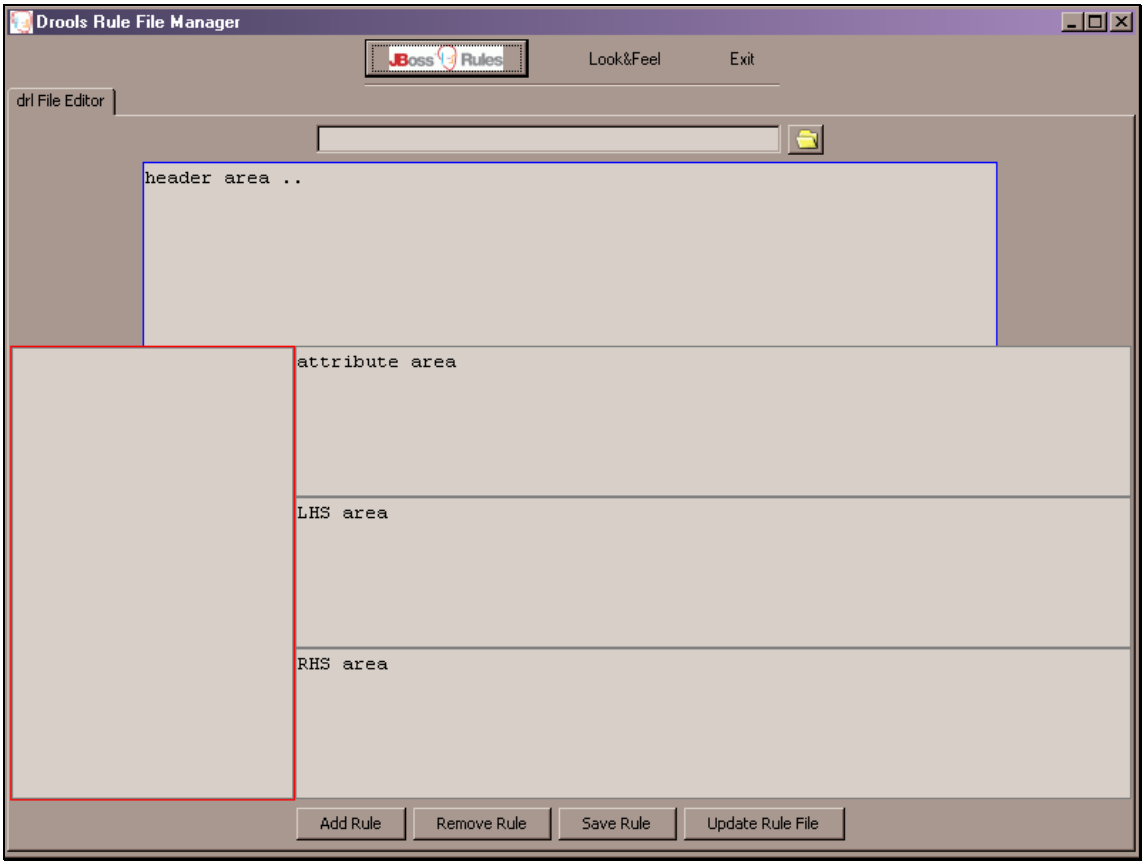



Figure 10.1 Rule file editor

At the initial screen, there are five boxes which can be defined as rule name container, rule file header area, rule attribute area, LHS area and RHS area.

To editing a rule file, target drl file should be selected from the file system. Whenever selecting target file, its content is read and boxes are filled by related information (Figure 10.2).

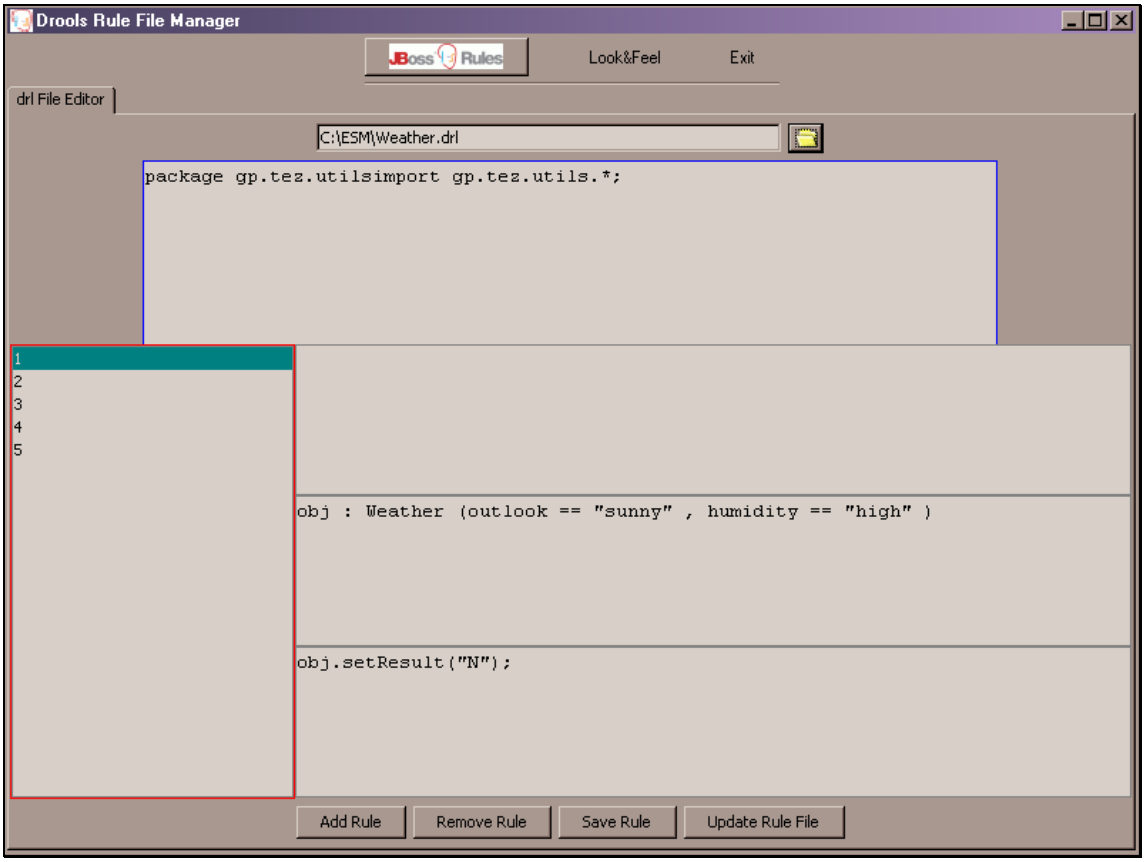



Figure 10.2 Rule editing

By means of selecting a rule in the rule container, rule information are changed automatically on the three boxes which is right hand side of the rule container. These three boxes are editable. User can change the information in these boxes. After changing rule should be saved. Saving is done on the memory. Whenever finish change operation rule file can be updated by “update rule file” button.

## 11. DISCUSSIONS

We often meet decision-making problems in our daily life or working environment. Sometimes it is very difficult for us to make good decision. In practice, we usually use our past experiences to make a decision. We can see these past experiences as a form of performing experiments to come to a correct decision. However, executing experiments costs time and money. Fortunately, the developments of computer technologies and automatic learning techniques can make this easier and more efficient. In the domain of machine learning where it always lets computers decide or come up with suggestions for the right decision, there exist many approaches of decision making techniques, such as decision trees, artificial neural networks and Bayesian learning. This thesis focuses on the decision tree approach to solve decision making problems.

There exist many methods to do decision analysis. Each method has its own advantages and disadvantages. In machine learning, decision tree learning is one of the most popular techniques for making classifications decisions in pattern recognition.

The approach of decision tree is used in many areas because it has many advantages. Compared with maximum likelihood and version spaces methods, decision tree is the quickest, especially under the condition that the concept space is large. Furthermore, it is easy to do the data preparation and to understand for non-technical people (Liang G., 2005).

Decision tree learning algorithm has been successfully used in expert systems in capturing knowledge. The main task performed in these systems is using inductive methods to the given values of attributes of an unknown object to determine appropriate classification according to decision tree rules.

Expert systems provide a strong rationale for continued funding of research on machine learning, but they also serve to sharpen our understanding of problems. Expert systems offer a focus for development of new machine learning methods and better understanding of old ones(Buchanan B.G., 1989).

There is a large class of expert systems whose purpose is essentially to classify cases, for example to diagnose disease from symptoms. Expert systems have become an important decision making tool in many organizations. Some of the benefits attributed to expert systems include increased quality, reduced decision making time, and reduced downtime. Examples of successful expert systems are reported in many areas such as ticket auditing, trouble shooting, risk analysis ,computer system design, and building construction.

An expert system has mainly two parts, rule base and inference engine. The inference engine is the mechanism by which the search for conclusions or reasoning is conducted using a search strategy of the knowledge built in the rule base. These search strategies could be either or both the forward and backward reasoning. The inference engine is located between the rule-base and the user interface where it accepts inputs from the user and tries to draw a conclusion or answer with reasoning a users' question.

These two main parts can be handled by a rule engine. Rule engine features are described as below on the JSR-94 specification document.

- Promote declarative programming by externalizing business or application logic.
- Include a documented file-format or tools to author rules and rule execution sets external to the application.
- Act upon input objects to produce output objects. Input objects are often referred to as facts and are a representation of the state of the application domain. Output objects are often referred to as conclusions or inferences and are grounded by the application into the application domain.
- The rule engine may execute actions directly, which affect the application domain, input objects, the execution cycle, rules, or the rule engine.
- The rule engine may merely create output objects, delegating the interpretation and execution of the output objects to the caller.

In this thesis “Drools Rule Engine” is used. Drools (The JBoss Rules Engine) is a JSR-94 complaint rules engine , and is fully open source under an “Apache-Style” License. Not only does it express rules in a familiar Java and XML syntax , it has a strong user and developer

community. Other advantage of this rule engine is; RETE based highly efficient object oriented algotihm is used for the inference engine.

All implementations are done by means of the java technologies and open source tools for this framework. Result of this approach, this framework can be used all environment without any license restriction.

## 12. CONCLUSION

In this thesis a “Rule Based Expert System Framework” is presented. Adopting a rule-based approach for the framework has the following advantages:

- Rules that represent policies are easily communicated and understood.
- Rules retain a higher level of independence than conventional programming languages.
- Rules separate knowledge from its implementation logic.
- Rules can be changed without changing source code; thus, there is no need to recompile the application's code.

This study is an example of fully open source java projects. Two main parts can be expressed for this framework, rule base construction part and web interface part. In rule base construction part, a rule set is derived from a data set by means of ID3 decision tree algorithm. For web based interface a web application, which can be run any java based servlet engine, is developed. This web application act as an expert system which is used JBoss Rule Engine as an inference mechanism. In addition these parts some utility applications, like drl file editor, are prepared.

For the future plan, this study can be applied for the more complex environments. An implementation on the portal system, an implementation on the workflow process engine (eg. BPEL engine) or an implementation on the enterprise service bus virtualization can be set an example of the complex environment implementations. These types of environments have lots of rules and need good rule engine implementations.

## REFERENCES


Buchanan B.G., (1989), "Can Machine Learning Offer Anything to Expert Systems?", Professor of Computer Science, Medicine, and Philosophy, University of Pittsburgh, Pittsburgh, *PA 15260*

Buchanan B.G and Shortliffe E.H., (1984), *Rule-Based Expert Systems,* Addison-Wesley Publishing Company

Chan S. T. and Gröndahl F., (2005), An Object-Oriented Rule-Based System, Master Thesis, Uppsala University Department of Information Science Computing Science Division

Chanda M. S., (2004), "Integration of a Rule Engine Component With a Portal Platform", Master Thesis, Technical Unıversity Hamburg-Harburg Germany

D'Hondt M., (2004), "Hybrid Aspects for Integrating Rule-Based Knowledge and Object-Oriented Functionality", Thesis , Vrije Universiteit Brussel Faculteit Wetenschappen Vakgroep Informatica System and Software Engineering Lab

Dobroslawa M., Grzymala-Busse and Jerzy W. Grzymala-Busse, (1995), "On The Usefulness Of Machine Learning Approach To Knowledge Acquisition", Department of Electrical Engineering and Computer Science, University of Kansas, Lawrence, KS 66045,USA

Dumais S. 1,Platt J 1.,Heckerman D 1.,Sahami M. 2,(1998),"Inductive Learning Algorithms and Representations for Text Categorization", 1 Microsoft Research One Microsoft Way Redmond, WA 98052, 2 Computer Science Department Stanford University Stanford, CA 94305-9010

Heintzman D., (2003), An introduction to open computing, open standards, and open source Staff, IBM

Elouedi Z., Mellouli K., and Smets P., (2000), "Classification with Belief Decision Trees", Institut Superieur de Gestion de Tunis, 41 Avenue de la liberte, cite Bouchoucha, 2000 Le Bardo, Tunis, Tunisia, IRIDIA, Universite Libre de Bruxelles, 50 av., F. Roosvelt, CP194/6, 1050 Brussels, Belgium

Forgy C., (1979), On the efficient implementation of production systems, Ph.D. Thesis Carnegie Mellon University

Kamber M., Winstone L., Gong W., Cheng S., Han J, (1997), "Generalization and Decision Tree Induction: Efficient Classification in Data Mining", Database Systems Research Laboratory School of Computing Science Simon Fraser University, B.C., Canada V5A 1S6

Kolahi S., (2006), Example on Decision Table: Jacket Weather Department of Computer Science Concordia University

Kingston J., (1987), "Rule-Based Expert Systems And Beyond: An Overview", Artificial Intelligence Applications Institute University of Edinburgh

Liang G., (2005), "A comparative study of three Decision Tree algorithms: ID3, Fuzzy ID3 and Probabilistic Fuzzy ID3", Bachelor Thesis Informatics & Economics Erasmus University Rotterdam Rotterdam, Netherlands

Mahmoud Q. H., (2005), "Getting Started With the Java Rule Engine API (JSR 94): Toward Rule-Based Applications", Java Sun Article

Maja D., (2004), Hybrid Aspects for Integrating Rule-Based Knowledge and Object-Oriented Functionality, Vrije Universiteit Brussel Faculteit Wetenschappen Vakgroep Informatica System and Software Engineering Lab

Pham D. T. and Afify A. A., (2004),"Machine-learning techniques and their applications in manufacturing", Manufacturing Engineering Centre, Cardiff University, Cardiff, UK

Polumetla A., (2006), "Machine Learning Methods For The Detection Of RWIS Sensor Malfunctions", A Thesis Submitted To The Faculty Of The Graduate School Of The Unıversity Of Minnesota

Pop D. and Negru V., (2003), "An Extensible Environment for Expert System Development" Department of Computer Science, University of the West from Timi_oara 4 V. Pârvan Street, RO-1900 Timi_oara, Romania

Rosenberg F. and Dustdar S., (2005), "Business Rules Integration in BPEL – A Service-Oriented Approach", Technical University of Vienna Information Systems Institute Distributed Systems Group

Proctor M., Neale M., Lin P., Frandsen M., (2006), Drools Documentation , *http://labs.jboss.com/file-access/default/members/jbossrules/freezone/docs/3.0.5/html/index.html* Red Hat, Inc

Spinellis D. 1 and Szyperski C. 2, (2004), How Is Open Source Affecting Software Development? *IEEE Software*, 21(1):28–33, January/February 2004, 1 Athens University of Economics and Business, 2 Microsoft Research

Subramanian G.H. 1, Nosek J. 2, Raghunathan S.P. 3, Kanitkar S.S. 4, (1989), "A Comparison of the decision table and tree", 1 Schoolof Business Administration, 2 Division of Computer and Information Sciences, Temple University, 3 Graduate School of Management,University Heights, 4 Comstar Computer Corp.

Toussaint A., (2003), Java Rule Engin e API Specification "Specification" Ver s ion : 1 .0, Specification and Maintenance Lead

Quinlan, J.R., (1985), “Induction of Decision Trees”, Centre for Advanced Computing Sciences, New South Wales Institute of Technology, Sydney 2007,Australia

Utgoff P.E., (1995), Decision Trees, Department of Computer and Information Science, University of Massachusetts, Amherst, MA 01003

Witten I.H., Frank E., (1999), *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations,* The Morgan Kaufmann Series in Data Management Systems

Yüret D., (2003), Machine Learning Lecture Notes, Assistant Professor of Computer Engineering Koc University

## APPENDIX I. JAVADOCS OF THE JAVA CLASSES

**gp.tez.jbossrule**

# Class RuleExecuter

```
java.lang.Object
  |
  +--gp.tez.jbossrule.RuleExecuter
```

public class **RuleExecuter**

extends java.lang.Object

This class executes the rule. drl file name is required. if JSR94 is used additionally RULE_SERVICE_PROVIDER and DROOLS_RULE_SERVICE_PROVIDER_CLASS attributes required.

**Version:**

1.1

**Author:**

gokhan polat

| Constructor Summary | |
|---|---|
| **RuleExecuter**() | |

| Method Summary | |
|---|---|
| void | **fireRule**()<br>fire rule by means of normal jbossrule procedure |
| void | **fireRuleJSR94**()<br>fire rule by means of JSR94 standard procedure |
| java.lang.String | **getDrlName**() |
| java.lang.String | **getDROOLS_RULE_SERVICE_PROVIDER_CLASS**() |
| java.lang.Object | **getObject**() |
| java.lang.String | **getRULE_SERVICE_PROVIDER**() |

| | |
|---|---|
| org.drools.RuleBase | **readRule**() |
| void | **setDrlName**(java.lang.String drlName) |
| void | **setDROOLS_RULE_SERVICE_PROVIDER_CLASS**(java.lang.String drools_rule_service_provider_class) |
| void | **setObject**(java.lang.Object object) |
| void | **setRULE_SERVICE_PROVIDER**(java.lang.String rule_service_provider) |

| Methods inherited from class java.lang.Object |
|---|
| equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

# Constructor Detail

### RuleExecuter

public **RuleExecuter**()

# Method Detail

### fireRule

public void **fireRule**()

fire rule by means of normal jbossrule procedure

### readRule

public org.drools.RuleBase **readRule**()

### fireRuleJSR94

public void **fireRuleJSR94**()

fire rule by means of JSR94 standard procedure

### getDROOLS_RULE_SERVICE_PROVIDER_CLASS

public java.lang.String **getDROOLS_RULE_SERVICE_PROVIDER_CLASS**()

### setDROOLS_RULE_SERVICE_PROVIDER_CLASS

public void **setDROOLS_RULE_SERVICE_PROVIDER_CLASS**(java.lang.String drools_rule_service_provider_class)

### getRULE_SERVICE_PROVIDER

```
public java.lang.String getRULE_SERVICE_PROVIDER()
```

### setRULE_SERVICE_PROVIDER

```
public void setRULE_SERVICE_PROVIDER(java.lang.String
rule_service_provider)
```

### getObject

```
public java.lang.Object getObject()
```

### setObject

```
public void setObject(java.lang.Object object)
```

### getDrlName

```
public java.lang.String getDrlName()
```

### setDrlName

```
public void setDrlName(java.lang.String drlName)
```

**gp.tez.utils**

# Class ID3

```
java.lang.Object
  |
  +--gp.tez.utils.ID3
```

public class **ID3**

extends java.lang.Object

A simple implementation of the ID3 algorithm This is a modified version to make my code closer to the standard ID3 algorithm

**Version:**

Dec. 13 2004, updated Sep 2006

**Author:**

Dr. Benny Raphael , updated by Gokhan Polat with the permission from author

| Constructor Summary | |
|---|---|
| ID3(java.lang.String infileName) | |
| ID3(java.lang.String infileName, java.lang.String resultClass) | |

| Method Summary | |
|---|---|
| boolean | **alreadyUsedToDecompose**(gp.tez.utils.ID3.TreeNode node, int attribute)<br>This function checks if the specified attribute is used to decompose the data set in any of the parents of the specfied node in the decomposition tree. |
| double | **calculateEntropy**(java.util.Vector data)<br>Calculates the entropy of the set of data points. |
| void | **createRules4File**(gp.tez.utils.ID3.TreeNode node, java.lang.String tab, java.io.BufferedWriter out, java.lang.String shortClassName) |
| void | **decomposeNode**(gp.tez.utils.ID3.TreeNode node)<br>This function decomposes the specified node according to the ID3 algorithm. |
| int[] | **getAllValues**(java.util.Vector data, int attribute)<br>Returns all the values of the specified attribute in the data set |
| java.util.Vector | **getSubset**(java.util.Vector data, int attribute, int value) |
| int | **getSymbolValue**(int attribute, java.lang.String symbol)<br>This function returns an integer corresponding to the symbolic value of the attribute. |
| void | **listRules**(gp.tez.utils.ID3.TreeNode node, java.lang.String tab) |
| static void | **main**(java.lang.String[] args) |
| void | **printTree**(gp.tez.utils.ID3.TreeNode node, java.lang.String tab) |
| int | **readData**(java.lang.String filename)<br>Function to read the data file. |
| void | **runCreateRuleFile**()<br>This function prints the rules to a .drl file.. |
| void | **runListRules**()<br>This function prints the rules as a sentence.. |
| void | **runPrintTree**()<br>This function prints the decision tree in the form of if/then/else structure. |

| Methods inherited from class java.lang.Object |
| --- |
| equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

# Constructor Detail

### ID3

```
public ID3(java.lang.String infileName,
           java.lang.String resultClass)
```

### ID3

```
public ID3(java.lang.String infileName)
```

# Method Detail

### getSymbolValue

```
public int getSymbolValue(int attribute,
                          java.lang.String symbol)
```

This function returns an integer corresponding to the symbolic value of the attribute. If the symbol does not exist in the domain, the symbol is added to the domain of the attribute

### getAllValues

```
public int[] getAllValues(java.util.Vector data,
                          int attribute)
```

Returns all the values of the specified attribute in the data set

### getSubset

```
public java.util.Vector getSubset(java.util.Vector data,
                                  int attribute,
                                  int value)
```

### calculateEntropy

```
public double calculateEntropy(java.util.Vector data)
```

Calculates the entropy of the set of data points. The entropy is calculated using the values of the output attribute which is the last element in the array attribtues

### alreadyUsedToDecompose

```
public boolean alreadyUsedToDecompose(gp.tez.utils.ID3.TreeNode node,
                                      int attribute)
```

This function checks if the specified attribute is used to decompose the data set in any of the parents of the specfied node in the decomposition tree. Recursively checks the specified node as well as all parents

### decomposeNode

```
public void decomposeNode(gp.tez.utils.ID3.TreeNode node)
```

This function decomposes the specified node according to the ID3 algorithm. Recursively divides all children nodes until it is not possible to divide any further I have changed this code from my earlier version. I believe that the code in my earlier version prevents useless decomposition and results in a better decision tree! This is a more faithful implementation of the standard ID3 algorithm

### readData

```
public int readData(java.lang.String filename)
             throws java.lang.Exception
```

Function to read the data file. The first line of the data file should contain the names of all attributes. The number of attributes is inferred from the number of words in this line. The last word is taken as the name of the output attribute. Each subsequent line contains the values of attributes for a data point. If any line starts with // it is taken as a comment and ignored. Blank lines are also ignored.

### runPrintTree

```
public void runPrintTree()
```

This function prints the decision tree in the form of if/then/else structure. The action part of the rule is of the form outputAttribute = "symbolicValue" or outputAttribute = { "Value1", "Value2", .. } The second form is printed if the node cannot be decomposed any further into an homogenous set

### printTree

```
public void printTree(gp.tez.utils.ID3.TreeNode node,
                      java.lang.String tab)
```

### runListRules

```
public void runListRules()
```

This function prints the rules as a sentence..

### listRules

```
public void listRules(gp.tez.utils.ID3.TreeNode node,
                      java.lang.String tab)
```

### runCreateRuleFile

```
public void runCreateRuleFile()
```

This function prints the rules to a .drl file..

### createRules4File

```
public void createRules4File(gp.tez.utils.ID3.TreeNode node,
                             java.lang.String tab,
                             java.io.BufferedWriter out,
                             java.lang.String shortClassName)
```

### main

```
public static void main(java.lang.String[] args)
                 throws java.lang.Exception
```

**gp.tez.utils**

# Class FindDistinctValuesFromFile

```
java.lang.Object
  |
  +--gp.tez.utils.FindDistinctValuesFromFile
```

public class **FindDistinctValuesFromFile**

extends java.lang.Object

for Expert System Web Interface , *_distinctValues file needed. this file is generated from data set file by means of this program

**Version:**

1.0

**Author:**

gokhan polat

| Constructor Summary | |
|---|---|
| **FindDistinctValuesFromFile**() | |
| **FindDistinctValuesFromFile**(java.lang.String fileName_p) | |

| Method Summary | |
|---|---|
| static void | **main**(java.lang.String[] args) |
| boolean | **makeNewFile**(java.lang.String filename)<br>make _distictValues file |
| int | **readData**(java.lang.String filename)<br>reading data from source data set file |

| Methods inherited from class java.lang.Object |
|---|
| equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

# Constructor Detail

### FindDistinctValuesFromFile

```
public FindDistinctValuesFromFile()
```

### FindDistinctValuesFromFile

```
public FindDistinctValuesFromFile(java.lang.String fileName_p)
```

## Method Detail

### readData

```
public int readData(java.lang.String filename)
```

reading data from source data set file

**Parameters:**

`filename` -

**Returns:**

int

### makeNewFile

```
public boolean makeNewFile(java.lang.String filename)
```

make _distictValues file

**Parameters:**

`filename` -

**Returns:**

boolean

### main

```
public static void main(java.lang.String[] args)
```

**gp.tez.utils**

# Class ArrangeFile4DT

```
java.lang.Object
  |
  +--gp.tez.utils.ArrangeFile4DT
```

public class **ArrangeFile4DT**

extends java.lang.Object

Arranges dataset file to the acceptable format for decision tree algorithm

**Version:**

1.0

**Author:**

gokhan polat

| Constructor Summary | |
|---|---|
| **ArrangeFile4DT**() | |
| **ArrangeFile4DT**(java.lang.String fileName_p, java.lang.String divideConstant_p) | |

| Method Summary | |
|---|---|
| static void | **main**(java.lang.String[] args) |
| boolean | **makeNewFile**(java.lang.String filename)<br>new file for decision tree alg. |
| int | **readData**(java.lang.String filename)<br>reading data from source data set file |

| Methods inherited from class java.lang.Object |
|---|
| equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

## Constructor Detail

### ArrangeFile4DT

```
public ArrangeFile4DT()
```

### ArrangeFile4DT

```
public ArrangeFile4DT(java.lang.String fileName_p,
                      java.lang.String divideConstant_p)
```

# Method Detail

### readData

```
public int readData(java.lang.String filename)
```

reading data from source data set file

### makeNewFile

```
public boolean makeNewFile(java.lang.String filename)
```

new file for decision tree alg.

**Parameters:**

`filename` -

**Returns:**

boolean

### main

```
public static void main(java.lang.String[] args)
```

**gp.tez.utils**

# Class Weather

```
java.lang.Object
  |
  +--gp.tez.utils.Weather
```

public class **Weather**

extends java.lang.Object

Weather bean. has only getter and setter methods.

**Version:**

1.0

**Author:**

gokhan polat

| Constructor Summary | |
|---|---|
| **Weather**() | |

| Method Summary | |
|---|---|
| java.lang.String | **getHumidity**() |
| java.lang.String | **getOutlook**() |
| java.lang.String | **getResult**() |
| java.lang.String | **getTempreature**() |
| java.lang.String | **getWindy**() |
| void | **setHumidity**(java.lang.String humidity) |
| void | **setOutlook**(java.lang.String outlook) |
| void | **setResult**(java.lang.String result) |

| | |
|---|---|
| void | **setTempreature**(java.lang.String tempreature) |
| void | **setWindy**(java.lang.String windy) |

| Methods inherited from class java.lang.Object |
|---|
| equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

# Constructor Detail

## Weather

public **Weather**()

# Method Detail

## getHumidity

public java.lang.String **getHumidity**()

## setHumidity

public void **setHumidity**(java.lang.String humidity)

## getOutlook

public java.lang.String **getOutlook**()

## setOutlook

public void **setOutlook**(java.lang.String outlook)

## getResult

public java.lang.String **getResult**()

## setResult

public void **setResult**(java.lang.String result)

## getTemperature

public java.lang.String **getTemperature**()

## setTemperature

public void **setTemperature**(java.lang.String tempreature)

## getWindy

public java.lang.String **getWindy**()

## setWindy

public void **setWindy**(java.lang.String windy)

**gp.tez.utils**

# Class CarAccept

```
java.lang.Object
  |
  +--gp.tez.utils.CarAccept
```

public class **CarAccept**

extends java.lang.Object

CarAccept bean. has only getter and setter methods.

**Version:**

1.0

**Author:**

gokhan polat

| Constructor Summary | |
|---|---|
| **CarAccept**() | |

| Method Summary | |
|---|---|
| java.lang.String | **getDoors**() |
| java.lang.String | **getMaintCost**() |
| java.lang.String | **getPersons**() |
| java.lang.String | **getPrice**() |
| java.lang.String | **getResult**() |
| java.lang.String | **getSafety**() |
| java.lang.String | **getTrunkSize**() |
| void | **setDoors**(java.lang.String doors) |

| | |
|---|---|
| void | **setMaintCost**(java.lang.String maintCost) |
| void | **setPersons**(java.lang.String persons) |
| void | **setPrice**(java.lang.String price) |
| void | **setResult**(java.lang.String result) |
| void | **setSafety**(java.lang.String safety) |
| void | **setTrunkSize**(java.lang.String trunkSize) |

| Methods inherited from class java.lang.Object |
|---|
| equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

# Constructor Detail

### CarAccept

public **CarAccept**()

# Method Detail

### getDoors

public java.lang.String **getDoors**()

### setDoors

public void **setDoors**(java.lang.String doors)

### getMaintCost

public java.lang.String **getMaintCost**()

### setMaintCost

public void **setMaintCost**(java.lang.String maintCost)

### getPersons

public java.lang.String **getPersons**()

### setPersons

public void **setPersons**(java.lang.String persons)

### getPrice

```
public java.lang.String getPrice()
```

### setPrice

```
public void setPrice(java.lang.String price)
```

### getResult

```
public java.lang.String getResult()
```

### setResult

```
public void setResult(java.lang.String result)
```

### getSafety

```
public java.lang.String getSafety()
```

### setSafety

```
public void setSafety(java.lang.String safety)
```

### getTrunkSize

```
public java.lang.String getTrunkSize()
```

### setTrunkSize

```
public void setTrunkSize(java.lang.String trunkSize)
```

**gp.tez.ruleadmin**

# Class RuleAdmin

```
java.lang.Object
  |
  +--gp.tez.ruleadmin.RuleAdmin
```

public class **RuleAdmin**

extends java.lang.Object

This is utility program. by means of this , drl files can be edited easily.

**Version:**

1.3

**Author:**

gokhan polat

| Field Summary | |
|---|---|
| java.lang.String | **RAdmin_HOME** |

| Constructor Summary | |
|---|---|
| **RuleAdmin**() | |

| Method Summary | |
|---|---|
| boolean | **initConsole**() |
| static void | **log**(java.lang.String msg) |
| static void | **main**(java.lang.String[] args) |
| void | **setMenu**()<br>builds menu ******* |
| void | **settingRoot**() |

| Methods inherited from class java.lang.Object |
|---|
| equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

# Field Detail

### RAdmin_HOME

```
public java.lang.String RAdmin_HOME
```

# Constructor Detail

### RuleAdmin

```
public RuleAdmin()
```

# Method Detail

### initConsole

```
public boolean initConsole()
```

### settingRoot

```
public void settingRoot()
```

### setMenu

```
public void setMenu()
```

buils menu *******

### main

```
public static void main(java.lang.String[] args)
```

### log

```
public static void log(java.lang.String msg)
```

**gp.tez.ruleadmin**

# Class DrlEditor

```
java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Container
              |
              +--javax.swing.JComponent
                    |
                    +--javax.swing.JPanel
                          |
                          +--gp.tez.ruleadmin.DrlEditor
```

**All Implemented Interfaces:**

javax.accessibility.Accessible, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable

public class **DrlEditor**

extends javax.swing.JPanel

this class generates JPanel for drl editing

**Version:**

1.3

**Author:**

gokhan polat

**See Also:**

Serialized Form

| Inner classes inherited from class javax.swing.JComponent |
|---|
| javax.swing.JComponent.AccessibleJComponent |

| Fields inherited from class javax.swing.JComponent |
|---|
| TOOL_TIP_TEXT_KEY, UNDEFINED_CONDITION, WHEN_ANCESTOR_OF_FOCUSED_COMPONENT, WHEN_FOCUSED, WHEN_IN_FOCUSED_WINDOW |

| Fields inherited from class java.awt.Component |
|---|
| BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT |

| Fields inherited from interface java.awt.image.ImageObserver |
|---|
| ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH |

| Constructor Summary | |
|---|---|
| **DrlEditor**(java.lang.String RAdmin_Home) | |

| Method Summary | |
|---|---|
| void | **addRule**()<br>adding new rule to the rule list |
| void | **analyzeRuleFromString**(java.lang.String str, Rule rule)<br>analize the rules inside the drl file |
| void | **fileChoose**()<br>choosing the rule file for editing |
| boolean | **init**() |
| static void | **log**(java.lang.String msg)<br>generic sysout |
| boolean | **readDrlFile**() |
| void | **showRuleDetail**(java.lang.String ruleName)<br>shows rule detail |
| void | **updateRule**()<br>updates rule attributes |
| void | **updateRuleFile**()<br>after updating rule new file generated |

| Methods inherited from class javax.swing.JPanel |
|---|
| getAccessibleContext, getUIClassID, updateUI |
| **Methods inherited from class javax.swing.JComponent** |
| addAncestorListener, addNotify, addPropertyChangeListener, addPropertyChangeListener, addVetoableChangeListener, computeVisibleRect, contains, createToolTip, disable, enable, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, getActionForKeyStroke, getActionMap, getAlignmentX, getAlignmentY, getAutoscrolls, getBorder, getBounds, getClientProperty, getConditionForKeyStroke, getDebugGraphicsOptions, getGraphics, getHeight, getInputMap, getInputMap, getInputVerifier, getInsets, getInsets, getListeners, getLocation, getMaximumSize, getMinimumSize, getNextFocusableComponent, getPreferredSize, getRegisteredKeyStrokes, getRootPane, getSize, getToolTipLocation, getToolTipText, getToolTipText, |

| getTopLevelAncestor, getVerifyInputWhenFocusTarget, getVisibleRect, getWidth, getX, getY, grabFocus, hasFocus, hide, isDoubleBuffered, isFocusCycleRoot, isFocusTraversable, isLightweightComponent, isManagingFocus, isMaximumSizeSet, isMinimumSizeSet, isOpaque, isOptimizedDrawingEnabled, isPaintingTile, isPreferredSizeSet, isRequestFocusEnabled, isValidateRoot, paint, paintImmediately, paintImmediately, print, printAll, putClientProperty, registerKeyboardAction, registerKeyboardAction, removeAncestorListener, removeNotify, removePropertyChangeListener, removePropertyChangeListener, removeVetoableChangeListener, repaint, repaint, requestDefaultFocus, requestFocus, resetKeyboardActions, reshape, revalidate, scrollRectToVisible, setActionMap, setAlignmentX, setAlignmentY, setAutoscrolls, setBackground, setBorder, setDebugGraphicsOptions, setDoubleBuffered, setEnabled, setFont, setForeground, setInputMap, setInputVerifier, setMaximumSize, setMinimumSize, setNextFocusableComponent, setOpaque, setPreferredSize, setRequestFocusEnabled, setToolTipText, setVerifyInputWhenFocusTarget, setVisible, unregisterKeyboardAction, update |
|---|
| **Methods inherited from class java.awt.Container** |
| add, add, add, add, add, addContainerListener, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getLayout, insets, invalidate, isAncestorOf, layout, list, list, locate, minimumSize, paintComponents, preferredSize, printComponents, remove, remove, removeAll, removeContainerListener, setLayout, validate |
| **Methods inherited from class java.awt.Component** |
| action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, bounds, checkImage, checkImage, contains, createImage, createImage, dispatchEvent, enable, enableInputMethods, getBackground, getBounds, getColorModel, getComponentOrientation, getCursor, getDropTarget, getFont, getFontMetrics, getForeground, getGraphicsConfiguration, getInputContext, getInputMethodRequests, getLocale, getLocation, getLocationOnScreen, getName, getParent, getPeer, getSize, getToolkit, getTreeLock, gotFocus, handleEvent, imageUpdate, inside, isDisplayable, isEnabled, isLightweight, isShowing, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, postEvent, prepareImage, prepareImage, remove, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, repaint, repaint, repaint, resize, resize, setBounds, setBounds, setComponentOrientation, setCursor, setDropTarget, setLocale, setLocation, setLocation, setName, setSize, setSize, show, show, size, toString, transferFocus |

| **Methods inherited from class java.lang.Object** |
|---|
| equals, getClass, hashCode, notify, notifyAll, wait, wait, wait |

# Constructor Detail

### DrlEditor

```
public DrlEditor(java.lang.String RAdmin_Home)
```

## Method Detail

### init

```
public boolean init()
```

### fileChoose

```
public void fileChoose()
```

choosing the rule file for editing

### log

```
public static void log(java.lang.String msg)
```

generic sysout

### addRule

```
public void addRule()
```

adding new rule to the rule list

### updateRule

```
public void updateRule()
```

updates rule attributes

### updateRuleFile

```
public void updateRuleFile()
```

after updating rule new file generated

### readDrlFile

```
public boolean readDrlFile()
```

### analyzeRuleFromString

```
public void analyzeRuleFromString(java.lang.String str,
                                  Rule rule)
```

analize the rules inside the drl file

**Parameters:**

`str` -

`rule` -

### showRuleDetail

```
public void showRuleDetail(java.lang.String ruleName)
```

shows rule detail

**Parameters:**

`ruleName` -

**gp.tez.ruleadmin**

# Class Rule

```
java.lang.Object
  |
  +--gp.tez.ruleadmin.Rule
```

public class **Rule**

extends java.lang.Object

Rule bean. has only getter and setter methods.

**Version:**

1.0

**Author:**

gokhan polat

| Constructor Summary | |
|---|---|
| **Rule**() | |

| Method Summary | |
|---|---|
| java.lang.String | **getAttribute**() |
| java.lang.String | **getLHS**() |
| java.lang.String | **getName**() |
| java.lang.String | **getRHS**() |
| void | **setAttribute**(java.lang.String attribute) |
| void | **setLHS**(java.lang.String lhs) |
| void | **setName**(java.lang.String name) |
| void | **setRHS**(java.lang.String rhs) |

| Methods inherited from class java.lang.Object |
|---|
| equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

# Constructor Detail

### Rule

```
public Rule()
```

# Method Detail

### getAttribute

```
public java.lang.String getAttribute()
```

### setAttribute

```
public void setAttribute(java.lang.String attribute)
```

### getLHS

```
public java.lang.String getLHS()
```

### setLHS

```
public void setLHS(java.lang.String lhs)
```

### getName

```
public java.lang.String getName()
```

### setName

```
public void setName(java.lang.String name)
```

### getRHS

```
public java.lang.String getRHS()
```

### setRHS

```
public void setRHS(java.lang.String rhs)
```

**es.servlet**

# Class ESManager

```
java.lang.Object
  |
  +--javax.servlet.GenericServlet
        |
        +--javax.servlet.http.HttpServlet
              |
              +--es.servlet.ESManager
```

**All Implemented Interfaces:**

java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

public class **ESManager**

extends javax.servlet.http.HttpServlet

**Version:**

1.0

**See Also:**

Serialized Form

| Constructor Summary | |
|---|---|
| **ESManager**() | |

| Method Summary | |
|---|---|
| void | **destroy**() |
| void | **doGet**(javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse resp) |
| void | **doPost**(javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse resp) |
| void | **init**() |
| void | **performTask**(javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse resp) |

| | |
|---|---|
| void | **readProp**() |

| Methods inherited from class javax.servlet.http.HttpServlet |
|---|
| service |

| Methods inherited from class javax.servlet.GenericServlet |
|---|
| getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, log, log |

| Methods inherited from class java.lang.Object |
|---|
| equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

# Constructor Detail

## ESManager

```
public ESManager()
```

# Method Detail

## destroy

```
public void destroy()
```

**Overrides:**

destroy in class javax.servlet.GenericServlet

**See Also:**

()

## doGet

```
public void doGet(javax.servlet.http.HttpServletRequest req,
                  javax.servlet.http.HttpServletResponse resp)
           throws javax.servlet.ServletException,
                  java.io.IOException
```

**See Also:**

(javax.servlet.http.HttpServletRequest,
javax.servlet.http.HttpServletResponse)

### doPost

```
public void doPost(javax.servlet.http.HttpServletRequest req,
                   javax.servlet.http.HttpServletResponse resp)
            throws javax.servlet.ServletException,
                   java.io.IOException
```

**See Also:**

```
(javax.servlet.http.HttpServletRequest,
javax.servlet.http.HttpServletResponse)
```

### init

```
public void init()
          throws javax.servlet.ServletException
```

**Overrides:**

`init` in class `javax.servlet.GenericServlet`

**See Also:**

```
()
```

### performTask

```
public void performTask(javax.servlet.http.HttpServletRequest req,
                        javax.servlet.http.HttpServletResponse resp)
```

### readProp

```
public void readProp()
```

**es.bean**

# Class ReadDistinctValueFile

```
java.lang.Object
  |
  +--es.bean.ReadDistinctValueFile
```

public class **ReadDistinctValueFile**

extends java.lang.Object

**Author:**

gokhan polat

| Constructor Summary | |
|---|---|
| `ReadDistinctValueFile()` | |

| Method Summary | |
|---|---|
| boolean | `findOptions()` |

| | |
|---|---|
| java.util.Hashtable | **getAttributes**() |
| java.lang.String | **getFileName**() |
| java.lang.String | **getFormText**() |
| void | **setAttributes**(java.util.Hashtable attributes) |
| void | **setFileName**(java.lang.String fileName) |
| void | **setFormText**(java.lang.String formText) |

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

# Constructor Detail

## ReadDistinctValueFile

public **ReadDistinctValueFile**()

# Method Detail

## getFormText

public java.lang.String **getFormText**()

## setFormText

public void **setFormText**(java.lang.String formText)

## getAttributes

public java.util.Hashtable **getAttributes**()

## setAttributes

public void **setAttributes**(java.util.Hashtable attributes)

## getFileName

public java.lang.String **getFileName**()

## setFileName

public void **setFileName**(java.lang.String fileName)

## findOptions

public boolean **findOptions**()

## APPENDIX II. JSP FILES

**********

### FindFile.jsp

**********

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<%@ page
language="java"
contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"
%>
<%!
	public void jspInit() {

	}
	public void jspDestroy() {

	}
%>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<META name="GENERATOR" content="IBM WebSphere Studio">
</HEAD>
<BODY>
<h1 align="center">CHOOSE THE FORM</h1>
<HR>
<FORM action="/ES/ESManager" method="post" >
<TABLE border="1" align="center">
	<TBODY >
<%
	String ESMDir = "C:/ESM/";
%>
		<h2 align="center" style="color: maroon">Location of the
DistinctFiles is : </h2>
		<h3 align="center" style="color: maroon"><%=ESMDir%></h3>
<%
	java.io.File b = new java.io.File(ESMDir);
	java.io.File[] fileName = b.listFiles();
	int i=0;
	String cls="";
	String ara="";
	java.util.Enumeration jarEntries;
	while (i < fileName.length) {
		if(fileName[i].toString().endsWith("distinctValues") ){
%>
		<TR>
			<TD style="color:
blue"><%="\""+fileName[i].getName()+"\""  %> :</TD>
			<TD><input type='radio' name='fileOpt' value=
<%=fileName[i].getName()  %> /></TD>
		</TR>
<%
		}

	++i;
	}
 %>
```

```
      </TBODY>
</TABLE>
 <p align="center"><INPUT type="submit" name="SubmitButton" value="Choose
Distinct Value File"></p>
<INPUT type="hidden" name="operation" value="READFORM">
 </FORM>
</BODY>
</HTML>
```

**********

**RuleAnswers.jsp**

**********

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<%@ page
language="java"
contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"
%>
<%!
       public void jspInit() {

       }
       public void jspDestroy() {

       }
%>
<%
      java.util.Hashtable rAttribute = (java.util.Hashtable)
request.getAttribute("rAttribute");
      String formText = (String) request.getAttribute("formText");
%>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<META name="GENERATOR" content="IBM WebSphere Studio">
<META http-equiv="Content-Style-Type" content="text/css">
</HEAD>
<BODY>
<h1 align="center">QUESTION FORM</h1>
<h3 align="center" style='color: red'><%=formText  %></h3>
<FORM action="/ES/ESManager" method="post" >
<HR>
<%
      java.util.Enumeration e = rAttribute.keys();
      int i = 1;
      while (e.hasMoreElements()) {
            String element = (String) e.nextElement();
            java.util.Vector tmpV = (java.util.Vector)
rAttribute.get(element);
            String ruleText = element;
%>
<TABLE border="1" align="center">
      <TBODY >
<caption style='color: green'><%=ruleText  %></caption>
<%
            for (int j = 0; j < tmpV.size(); j++) {
```

```
                    String nOpt = (String) tmpV.elementAt(j);
%>
                <TR>
                    <TD width="180"><%=nOpt %></TD>
                    <TD width="80" align="center"><input type="radio"
name=Param_<%= ruleText %> value="<%= nOpt %>" /></TD>
                </TR>
<%
                }
                i++;
        }

%>
        </TBODY>
</TABLE>
<HR>
<p align="center"><INPUT type="submit" name="SubmitButton"
value="FormOK"></p>
<INPUT type="hidden" name="operation" value="RESULT">
<INPUT type="hidden" name="formText" value="<%= formText %>">
</FORM>
</BODY>
</HTML>
```

**********

**ResultForm.jsp**

**********

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<%@ page
language="java"
contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"
%>
<%!
        public void jspInit() {

        }
        public void jspDestroy() {

        }
%>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<META name="GENERATOR" content="IBM WebSphere Studio">
</HEAD>
<BODY>
<h1 align="center">FORM RESULT</h1>
<HR>
<%
        java.util.Hashtable ruleMap = (java.util.Hashtable)
request.getAttribute("ruleMap");
        String ruleResult = (String) request.getAttribute("ruleResult");
        String formText = (String) request.getAttribute("formText");
```

```
%>
<TABLE border="1" align="center">
      <TBODY >

<%
      java.util.Enumeration e = ruleMap.keys();

      while (e.hasMoreElements()) {
            String element = (String) e.nextElement();
%>
            <TR>
                  <TD style="color: blue"><%= element%></TD>
                  <TD><%= ruleMap.get(element).toString() %></TD>
            </TR>

            <%
      }
%>
      </TBODY>
</TABLE>

<h2 align="center" style="color: red"> Form Name : <%= formText %></h2>
<h2 align="center" style="color: maroon">Result of the Choosen Parameters
</h2>
<h3 align="center" style="color: red;font-weight:
bold">********************</h3>
<h3 align="center" style="color: red;font-weight:
bold"><%=ruleResult%></h3>
<h3 align="center" style="color: red;font-weight:
bold">********************</h3>
</BODY>
</HTML>
```

**CURRICULUM VITAE**

**GÖKHAN POLAT**

## PERSONAL INFORMATION

| | | |
|---|---|---|
| **Date of Birth** | **:** | September 10,1972 |
| **City, Country of Birth** | **:** | Tefenni/Burdur, Turkey |
| **Citizen** | **:** | Turkish |
| **Marital Status** | **:** | Married and has a daughter |
| **Military Obligation** | **:** | Done |

## EDUCATION

| | |
|---|---|
| 2003-cont. | Doğuş University Master Program for Computer Engineering |
| 1997-not finished | Akdeniz University MBA Program |
| 1989-1995 | M.E.T.U. Computer Engineering |

## EXPERIENCE

***July 2000 – today  Finansbank IT Center / İstanbul***

- DBA related jobs
- Storage Administration
- Project Member of the Core Banking Project
- Application Server Administration
- Java and J2EE architecture experience
- Implementation of Deployment System

***July 1996 – 2000  IT Center of Hospital of Akdeniz University / Antalya***

- System Analyst
- Programmer
- DBA