一個發現變異物件的知識擷取方法

研究生: 劉力豪 指導教授: 曾憲雄博士

國立交通大學資訊科學系

摘要

隨著知識庫系統的演進，也伴隨著產生許多的變異物件，而傳統的知識擷取表格，無法有效的發現變異物件。且沒有相關的研究指出，知識庫能夠隨著推論結果的演化。在本論文中，我們提出一個發現變異物件的知識擷取方法，找出隱藏在實際世界中的變異物件。由紀錄那些經常被推論且有著較低信賴指數的隱藏規則，並藉由新增物件，及兩個額外的運作: 改變屬性的型態及新增加屬性。這些運作可以幫助專家來找尋變異物件及提高隱含規則的信賴指數(CF)。為了實現我們的想法，我們亦提出一個 EMCUD-DRAMA-VODKA 的架構。在這個架構底下包含了知識產生，知識利用，及知識發掘三個階段。首先我們運用 EMCUD 來產生原本和隱含的規則。在知識利用階段，我們運用 DRAMA 所產生實際的運用上的使用紀錄，最後在知識發掘階段，我們運用 VODKA 來發現變異物件及屬性。我們也完成包含三個階段的系統之實作。並且我們使用蠕蟲來做為我們的例子。

關鍵字：信賴指數，知識表格，知識庫系統，不確定性，知識擷取，變異物件。

A Variant Object Discovering Knowledge Acquisition

Student: Li-Hao, Liu Advisor: Dr. Shian-Shyong Tseng

Department of Computer and Information Science

National Chiao Tung University

Hsinchu, Taiwan, 300, Republic of China

Abstract

The traditional knowledge acquisition process might be useless for solving the problems of variant object discovery. It might never be mentioned for the knowledge base to evolve with inference results. So the purpose of this thesis is to present a Variant Object Discovering Knowledge Acquisition (VODKA) to add a new object from the inference results, including two operations: changing attribute data type and adding a new attribute. These operations assist experts in discovering the variant objects and elevating the Certainty Factor (CF) of embedded rules. We also propose the framework of EMCUD-DRAMA-VODKA which has three phases: Knowledge Generation, Knowledge Utilization, and Knowledge Discovery. To begin with, we use Embedded Meaning Capturing and Uncertainty Deciding (EMCUD) to generate the original and embedded rules. Moreover, we apply these rules in the real world and accumulate the logs of DRAMA in the Knowledge Utilization phase. Finally, the VODKA is proposed to discover the variants or attributes in the Knowledge Discovery phase. In addition, a worm taxonomy classification example is given. The implementation of these phases is also constructed for describing how to develop a complete knowledge-based application.

Keywords: Certainty Factor, Knowledge Acquisition, Repertory Grid, Worm Taxonomy, Knowledge Based System, Embedded Meaning.

Acknowledgement

This work can be attributed to many people. The first one I want to thank is my advisor, Prof. Shian-Shyong Tseng (曾憲雄), who provided a motivating, enthusiastic, and critical atmosphere during the many discussions we had. His insightful advice helped me to not get lost during the development of this thesis. I am deeply indebted to my supervisor.

I also thank the members of my proposal committee: Prof. Chi-Sung Laih (賴溪松), Prof. Gwo-Jen Hwang (黃國禎), and Prof.Wen-Nung Tsai (蔡文能). They have read draft versions of the thesis, provided a source of technical criticism and aided in clarifying my presentation of ideas.

Moreover, discussions with the upperclassman Shun-Chieh Lin (林順傑) provided constructive comments during the whole time of my thesis development. Regarding the implementation issues of the whole system, I received a lot of valuable comments from Dr. Yao-TsungLin (林耀聰). Special thanks to Jui-Feng Weng (翁瑞鋒) for his assistance on my presentation. We worked many hours together on this work.

Many more persons participated in various ways to ensure my research succeeded and I am thankful to them all.

Li-Hao, Liu

July, 10, 2004

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Examples

Chapter 1. Introduction

With the technology revolution, mankind's ability to generate new data has rapidly increased. But our capability to analyze knowledge from the mass data could not keep up with the knowledge exploitation. As we know, knowledge system is an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution, such as disease diagnosis [3][14], investment prediction [2], or science [1]. Inference is the way that computer reasons and knowledge base stores the domain expertise in the computer recognizable format. To acquire the expertise, there are generally three kinds of approaches: interviewing experts by experienced knowledge engineers, machine learning, and knowledge acquisition systems. Knowledge engineers explicitly code the knowledge by interviewing the domain experts; Machine learning is the way to automatically generate some rules from some given training cases; knowledge acquisition systems are computer systems or tools that help experts or knowledge engineers to construct the knowledge system.

Michael Polanyi [13](1966) classified human knowledge into two categories: tacit knowledge, and explicit knowledge. Tacit knowledge is highly personal and hard to formalize. Explicit knowledge is the codified knowledge that can be transformed into formal, systematic language. Almost all of knowledge acquisition tools are suitable for eliciting explicit knowledge. One of the most popular knowledge acquisition techniques is the repertory gird [7], which is used to assist experts in identifying objects in a domain and how to distinguish between those objects. It helps knowledge engineers systematically elicit the domain expertise, especially on uncertainty or incomplete information.

The current knowledge acquisition techniques aim to achieve attribute reductions and then generate the most efficient rule set [12]. Existing repertory grid techniques and their improvements are limited in their potential to evolve with applications and to create new attributes.

A computer worm which is a self-propagating program via a network is an acutely evolving crisis [15].Since the recent worms are incredible sophisticated, they are difficult for experts to predict [8]. The taxonomy of the various worms continues to develop simply because the new tactics, payload, and attackers [15]. These above algorithms seem helpless for solving the problems of worm and its variant taxonomy. Here, the purpose of this thesis is to present a new knowledge acquisition methodology to learn new variant objects.

In addition, the contributions of this thesis are listed as follows.

1. Learning mechanism for variant objects

   Although several repertory grid techniques have been developed, the learning and feedback mechanism based on the real world application would be never mentioned. We propose a Variant Object Discovering Knowledge Acquisition (VODKA) method to add a new object from the applications by the following two operations: changing attribute data type and adding a new attribute. This is beneficial to discover the variant objects and elevate the certainty.

2. Worm variant discovering

   The first step in understanding the treat posed by computer worm is to understand the classes. The classification of worm is mainly based the domain experts [15] and might be in lack of a systematical method. Until now the repertory grid technology has not been applied to the worm classification domain. We clarify the attributes of existing worms. A series of computer worms and their variants are identified and classified. So we can discover the unknown worm of similar property.

3. Design and implementation of VODKA

   This thesis describes the theoretical foundations of a system which discovers variant objects using VODKA and a practical approach of its implementation. This approach provides an effective and efficient method for VODKA that is supported by the experiment.

Chapter 2. Related Work

Since the knowledge is rapidly growing, many knowledge acquisition methods have been proposed to systematically acquire the expertise from domain experts. In this chapter, we present the surveys of knowledge acquisition and three kinds of knowledge acquisition approaches. Further we are concerned about the knowledge acquisition tools approach.

2.1 Knowledge Acquisition

Michael Polaner (1966) [13] classified human knowledge into two categories: tacit and explicit knowledge. Tacit knowledge is highly personal and hard to formalize, making it difficult to communicate or share with others. Subjective insights, intuitions and hunches fall into this category of knowledge, i.e. knowledge that does not manifest as rules. Explicit knowledge is codified knowledge that can be translated in formal, systematic language. It can be best described as canonical knowledge, i.e. knowledge formalized within database, business rules, protocols, and so on.

Furthermore, to obtain the knowledge of a special domain, knowledge acquisition is proposed to transfer the expertise of domain experts into knowledge bases. Generally speaking, there are three kinds of approaches for knowledge acquisition.

1. Interviewing experts by experienced knowledge engineers.

After interviewing the domain expert, knowledge engineers explicitly code the knowledge. There are two major drawbacks about this approach. Firstly, it might be time consuming that the domain expert and knowledge engineers understand the thoughts of each other. In addition, it might be impossible to capture the embedded meaning.

2. Machine learning.

Given the training cases, machine learning approach will automatically learn out some rules. But it still has two disadvantages. Firstly, there might be few or no available training cases in many application domains. Moreover, it is hard to understand the relation among cases.

3. Knowledge acquisition systems.

The knowledge acquisition system is interviewing with the help of knowledge acquisition tools. It helps knowledge engineers work better in interviewing with experts. Besides, deeper knowledge can be elicited using this approach such as repertory grid technique which gets domain experts to rank objects against concepts.

The knowledge acquisition system solves the problem of communication between domain experts and knowledge engineers without the required training cases. This will be thoroughly described in the following section.

2.2 Knowledge Acquisition System

Here we are concerned about some current research issues on the Knowledge acquisition systems approach. In the past few years, many knowledge acquisition systems were developed to rapidly build prototypes and to improve the quality of the elicited explicit knowledge. The traditional approach of knowledge acquisition from domain experts is via interviewing. However, deeper knowledge can be elicited using knowledge acquisition system. Repertory grid which is based on Kelly's Personal construct theory [7] can be used as knowledge acquisition tools in the development of expert systems. The theory reports how people make sense of the world. The technique assists in identifying different objects in a domain and how to distinguish between these objects. A single repertory grid is represented as a matrix whose columns have elements labels and whose rows have construct labels. In a sense, a grid is represented for a class of objects, or individuals. In addition, the value assigned to an element-construct pair needs not be Boolean. Grid values have numeric ratings, probabilities, etc, where each value reflects the degree. The possibility of using different kinds of values enriches the repertory grid technique in eliciting and representation about a domain. For example, Hwang [5] extended the repertory grid technique to the fuzzy table. In the fuzzy table, constructs were fuzzy attributes that can be rated by means of fuzzy linguistic terms from a finite set. Jose J [6] developed a technique using fuzzy repertory grid for acquiring the finite set of attributes or variables which the expert uses in a classification problem characterizing and discriminating a set of elements. Dan Pan [12] proposed a self-optimization approach based on difference comparison tables for knowledge acquisition.

Up until now, several models have been proposed for handling uncertainties in expert systems. One of the most fruitful models of uncertain reasoning is the EMYCIN Certainty Factor (CF) model. Furthermore, CF model which was first used in the medical expert system EMYCIN [14] decides the degree of belief of a rule. To extract rules with embedded meaning, Embedded Meaning Capturing and Uncertainty Deciding (EMCUD) [4] knowledge acquisition based on Personal Construct Theory was proposed. Each embedded rule extracted by EMCUD has a CF, which is based upon a general repertory-grid-analysis method.

These approaches aimed at obtaining the set of attributes and values which the expert can be used to “measure” the object type. They provided an effective method for attribute reduction and rule complexity simplification. Although these are suitable for eliciting knowledge, none of them touches the problem learning for new variant objects from the application. So we propose a new approach to observe the inference results and generate new rules accordingly.

2.3 EMCUD

EMCUD is proposed to elicit the embedded meanings of knowledge from the existing repertory grids. Additionally, by interacting with experts, it will guide experts to decide the certainty degree of each rule with embedded meaning. To capture the embedded meanings of the resulting grids, the Attribute-Ordering Table (AOT), which is used to record the relative importance of each attribute to each object, is employed. There are three kinds of values in each AOT entry, a pair of attribute and object, "X", "D" or an integer; "X" means that there is no relationship between the attribute and the object; "D" means that the attribute dominates the object; An integer is represented for the relative important degree of the attribute to the object. The larger the integer is, the more important the attribute is to the object.

Using AOT, the original rules generate some rules with embedded meaning, and the CF of each rule, which is between 0 and 1, could be determined to indicate the degree of supporting the inference result. The higher CF is, the more reliable the result is. The algorithm is listed in Algorithm 2.1.

Algorithm 2.1: EMCUD algorithm [4]

```
Input: The hierarchical grids.
Output: The guiding rules with embedded meaning

Step1: Build the corresponding AOT with each grid of the hierarchical multiple
       grids.
Step2: Generate the possible rules with embedded meaning.
Step3: Select the accepted rules with embedded meaning through the interaction
       with experts.
Step4: Generate the CF of the rules with embedded meaning.
Step5: Stop.
```

A Certainty Sequence (CS) value is used to represent the decreasing degree of certainty for an embedded rule, which is generated by negating some predicates of its original rule. CS of the rules with embedded meaning is calculated by using the following formula.

$$CS(R_i) = SUM(AOT[attribute_k, object_i]),$$

where $attribute_k$ belongs to the attribute set of R_i, and $object_i$ is the object of R_i.

Assume the rules with embedded meaning listed below are generated from the rule, R_a: *if* $att_1 = A$ *and* $att_2 = B$ *and* $att_3 = C$ *then* $goal=o_1$, where the corresponding AOT entries are [2(att_1, obj_1), 1(att_2, obj_1), D (att_3, obj_1)].

R1: if Not ($att_1 = A$) *AND* $att_2 = B$ *AND* $att_3 = C$ *then* $goal=O_1$

R2: if $att_1 = A$ *AND Not* ($att_2 = B$) *AND* $att_3 = C$ *then* $goal=O_1$

R3: if Not ($att_1 = A$) *AND Not* ($att_2 = B$) *AND* $att_3 = C$ *then* $goal=O_1$

So the CS (R1) is AOT [att_1, obj_1] + AOT [att_2, obj_1] = 2 + 1 = 3.

To decide the CF of the embedded rules, we have to firstly decide the upper bound of CF, *UB* (R_a) and the lower bound of CF, *LB* (R_a), where R_a is the original rule for these embedded rules. *UB* (R_a) is the original rule's CF, since there is no embedded rule whose CF is greater than its original rule. *LB* (R_a) is determined by comparing the meaning-embedded rule with maximum CS value and the original rule. Four comparison levels suggested by the experts are "confirm", "strongly support", "support" and "may support" which are mapped to 1.0, 0.8, 0.6 and 0.4, respectively. CF of the embedded rule, R_i, is then generated by the following formula.

$$CF(R_i) = UB(R_a) - (CS(R_i) / MAX(CS_i) * (UB(R_a) - LB(R_a)))$$

Where *MAX* (CS_i) is the maximum CS value of the embedded rules generated from the original R_a.

For the example R_a and its generated embedded rules, R1, R2, and R3, the *MAX(CS_i)* for R1, R2, and R3 is CS_3.

To get the lower bound *UB(R_a), EMCUD will ask the following question:*

EMCUD: How confident do you think R1?

Expert: Confirm.

To get the lower bound *LB (R_a), EMCUD will ask the following question:*

Q_2: If the R1 is "Confirm", which degree supports *R3*?

Expert: May Support.

Since in the mapping function "Confirm" is mapped to 1.0 and "May Support" is mapped to 0.6, CF (R1), CF (R2), and CF (R3) are as follows.

CF(R1) = 1.0 – (2/3)*(1.0 – 0.4) = 0.6

CF(R2) = 1.0 – (1/3)*(1.0 – 0.4) = 0.8

CF (R3) = 1.0 – (3/3)*(1.0 – 0.4) = 0.4

Chapter 3. Variant Object Discovering Knowledge Acquisition (VODKA) Method

As mentioned above, most of the knowledge acquisition systems employ the repertory grid based approaches to deal with elicitation of explicit knowledge. These knowledge acquisition tools usually elicit knowledge as the representation of "IF … THEN..." format due to the simplicity of knowledge representation. Since the goals of the problems are usually clear, some systematic ways have been proposed to help knowledge engineers for eliciting expertise from domain experts.

3.1 Problem of EMCUD

EMCUD uses the attribute ordering table (AOT), which indicates the relative significance of attributes, to find the certain factor of the generated original and embedded rules. Although EMCUD successfully solves the problems of conventional repertory grid: knowledge representation, and embedded meaning, it might ignore the following problems:

1. Due to the knowledge exploitation, there are many variant objects emerging in a short time. It is infeasible to elicit the knowledge of such a new variant object which is derived from original objects using EMCUD.
2. EMCUD would generate the embedded rules with lower CF due to the low confidence of domain expert. It is usually hard to explain these rules; for example, if a rule whose CF is 0.5, it means experts have half confidence to assure the action. On the other hand, experts have half confidence to reject the action.

Since EMCUD is limited in its potential to evolve by application and to discover new variant objects, a methodology is proposed to discover the knowledge of new variant objects according to the frequency of negated attributes occurred in embedded rules. Hence, we can incrementally level up the CF of embedded rules.

The idea is to observe the application and provide the critical information to experts who are unsure the embedded rules with lower CF. This will incrementally enhance the explanation power of original knowledge base.

Example 3.1: An example of fruit classification.

We will state the problems in details by the following example and propose a new methodology to deal with them: To represent that "IF (Shape is Round) AND (Skin is Red) AND (Inside is White) Then Apple", we can use the Acquisition and AOT table of fruit classification which is shown in Table 3.1 and Table 3.2 respectively.

The Acquisition and AOT table of Fruit Classification

Table 3.1:.An example of fruits classification acquisition table

	Apple	Orange	Grapes
Shape	Round	Round	Chain
Skin	Red	Orange	Purple
Inside	White	Orange	Limpid

Table 3.2: An example of fruits classification AOT

	Apple	Orange	Grapes
Shape	D	D	D
Skin	1	D	2
Inside	2	1	1

In EMCUD, there are four kinds of data types Boolean, Single Value, Set of Values, and Range of Values. From the above grid, we can generate the following rules. For example, some of the rules generated from above grids are shown in Table 3.3.

Table 3.3: An example of fruits' original rules, embedded rules and their CF

Rule Classification	**Rule Type**	**Condition**	**Action**	**CF**
Apple	**Original**	**Shape=** Round,& **Skin =** Red& **Inside =**White	Apple	**0.8**
Apple	**Embedded**	**Shape=**Round,& **Skin =Not** Red& **Inside =Not** White	Apple	**0.6**
Orange	**Original**	**Shape=**Round,& **Skin =** Orange& **Inside =**Orange	Orange	**1.0**
Grapes	**Original**	**Shape=** Chain,& **Skin =**Purple& **Inside =**Limpid	Grapes	**0.8**
Grapes	**Embedded**	**Shape=** Chain,& **Skin =Not** Purple& **Inside =**Limpid	Grapes	**0.7**

However, based on EMCUD or any other traditional repertory grid approach, it might be hard to find the variant object and its rules, such as green apple and white grape in this example. In addition, the problem of rules with lower CF, for example, the embedded rule of Apple, still exists. Hence we propose a methodology to discover the variant objects such as green apple or white grape. Moreover, our approach assists experts in elevation of the embedded rules' CF.

The novelty of our approach draws from the way of learning from history. Via the user interface, the inference engine would return the inference results decided by the rule classes in the knowledge base. We can monitor the user's application log to record the inferred rules and assigned values. From the statistical data, we can find the embedded rules which have higher inferred frequency and lower CF which may be treated as variant objects. Furthermore, the assigned value firing these embedded rules may be the possible attribute value of variant objects; for example, there is originally red apple's knowledge in knowledge base. However, we can observe the inference history where "green" is often assigned to the attribute "skin". According to our LV algorithm, we would recommend the expert to add a new variant object, such as green apple and then generate the original and embedded rules of this new variant object. The idea is shown in Figure 3.1.

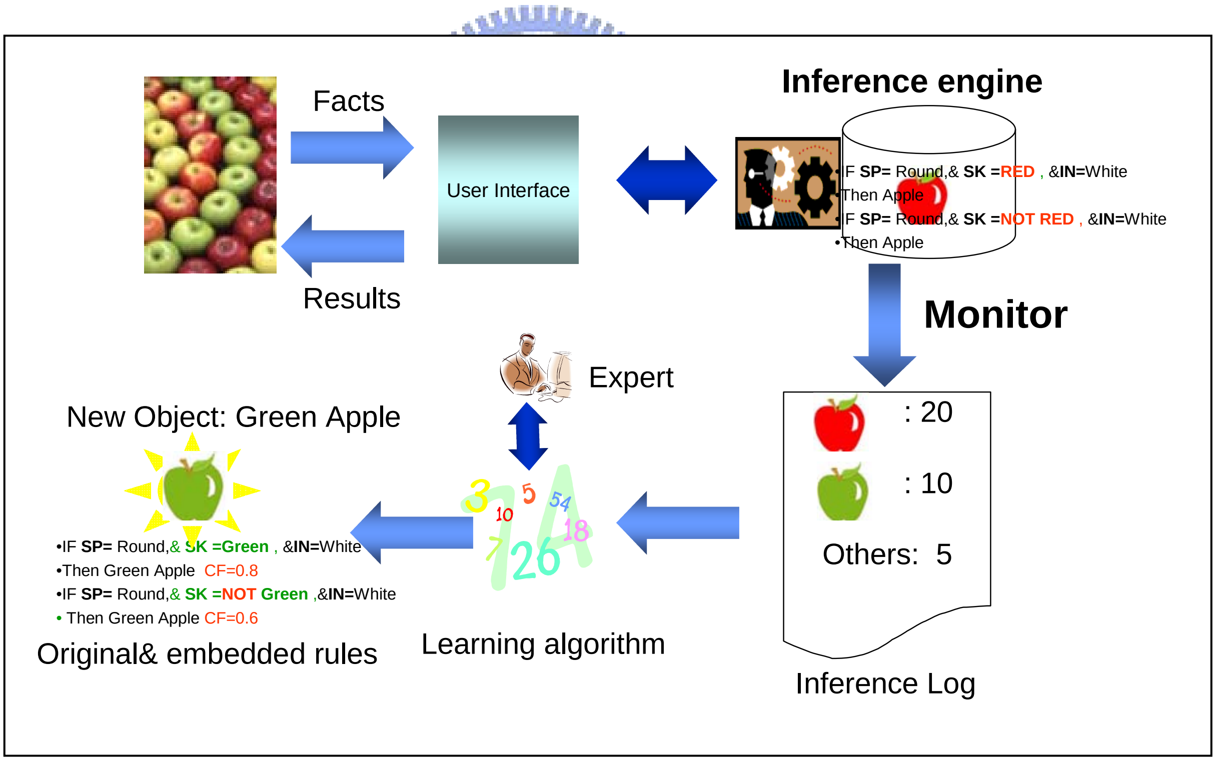


Figure 3.1: Concept of variant objects discovery

3.2 Variant Object Discovering Knowledge Acquisition (VODKA)

Because the embedded rules with lower CF means that the domain experts are highly unsure, there might be some variants if these are frequently inferred. To discover the variant objects, we propose a Variants Learning algorithm, including two operations: changing data type and adding new attribute.

3.2.1 Rule Format

We are concerned about the abnormal high percentage of inference, especially the embedded rules with lower CF. And the assigned values are the important information for experts to add a new object. So we define the rule format as follows.

IF Condition Then Action, CF

For example,

If **Shape=** Round, & **Skin =** Red& **Inside =**White Then Apple, CF=0.8

Besides, we also define the record format to store the inference history. The record format is a vector

$(A_1,\quad A_2, \ldots, A_m, R_i)$

Where:

$A_1,\quad A_2, \ldots, A_m$, are the assigned attribute values, and

R_i is the rule number.

For example, if a rule R0 is inferred with the fact **Shape=** Round,& **Skin =** Red& **Inside =**White, we will create a new vector like this

(Round, Red, White, R0)

Then we append it to the buffer. Finally, we write the buffer to the file.

3.2.2 Rule Fired Counting Matrix

To adjust condition adaptively, we define the Meta rule to trigger appropriate algorithms. In other words, we only need to adjust the Meta rule instead of the program. An $m \times n$ matrix A_{mxn} is defined to count the number of rule inference as follows:

$$\begin{bmatrix} C_{1,1} & C_{1,2}\cdots & C_{1,n} \\ C_{2,1} & C_{2,2}\cdots & C_{2,n} \\ . & . & . \\ C_{m,1} & C_{m,2}\cdots & C_{m,n} \end{bmatrix}$$

Where a is the number of attributes, b is the number of objects, the size of the matrix is $m \times n$, and $m = b$,$n = 2^a$. In addition, $C_{i,j}$ which is the element in row i and column j represents the counter to record the fired rule. Each row represents the original and embedded rules' counter of one object. $C_{1,1}\ C_{2,1}\ldots C_{m,1}$ represent the original rules' counter, and the others represent the counter of embedded rules. $C_{i,j}$'s value will increase by 1 when its corresponding rule is inferred. To find the rule easily, the integer j-1 can be represented as a binary number where we define the "1" in this format corresponding to the negated attribute of the embedded rules. An example is given as follows:

Given two attributes A, B, and the Action O_1 then $a = 2$,

$C_{1,1}$: O_1 original rule(If A, & B Then O_1)since 1-1=0 =$(00)_2$

$C_{1,2}$: O_1's embedded rule's (If A, & Not B Then O_1)since 2-1=1 =$(01)_2$

$C_{1,3}$: O_1's embedded rule's (If Not A, & B Then O_1)since 3-1=2 =$(10)_2$

$C_{1,4}$:O_1's embedded rule's (If Not A, & Not B Then O_1)since 4-1=3 =$(11)_2$

To implement the above representation method, we propose a Mapping algorithm where the relation of the rules and matrix is defined. The algorithm is listed in Algorithm 3.1

Algorithm 3.1: Mapping algorithm

```
Input:    a: number of attributes, b: number of objects.
Output: Rule Fired Counting Matrix.
Step 1: Create the $m \times n$ matrix $A_{mxn}$ where $m = b, n = 2^a$.
Step 2: For each element in each row i, each column j
        If j=1
            $C_{i,1}$ corresponds to the original rule of $Oi$.
        Else
            $C_{i,j}$ corresponds to embedded rules of $Oi$ according to the binary
              representation of j-1 where 1 represents the negated attribute.
Step 3: Return the matrix.
```

Example 3.2: An example of mapping function

Given two objects: O_1, O_2, and two attributes A,B, then a=2, b=2.

And there are four rules can be generated by EMCUD.

Rule Type	**Condition**	**Action**
Original R0	A&B	O_1
Embedded R1	A &Not B	O_1
Embedded R2	Not A &B	O_1
Embedded R3	Not A &Not B	O_1
Original R0	A&B	O_2
Embedded R1	A & Not B	O_2
Embedded R2	Not A & B	O_2
Embedded R3	Not A & Not B	O_2

So we can create an $m \times n$ matrix A_{mxn}, where m=2, n= $2^2 = 4$

$$\begin{bmatrix} C_{1,1} & C_{1,2} & C_{1,3} & C_{1,4} \\ C_{2,1} & C_{2,2} & C_{2,3} & C_{2,4} \end{bmatrix}$$

$C_{1,1}$: O_1 's original rule(If A, & B Then O_1)since 1-1=0 =(00)2

$C_{1,2}$: O_1 's embedded rule's (If A, & Not B Then O_1)since 2-1=1 =(01)2

$C_{1,3}$: O_1 's embedded rule's (If Not A, & B Then O_1)since 3-1=2 =(10)2

$C_{1,4}$: O_1 's embedded rule's (If Not A, & Not B Then O_1)since 4-1=3 =(11)2

$C_{2,1}, \ldots, C_{2,4}$ are similar to these cases.

To count the inferred rule correctly, we also present the record algorithm for uncertainty fired rules. To begin with, it reads the assigned value and fills the defined vector. And then it appends this new vector to the buffer. Finally, it writes the whole buffer to the file. The algorithm is listed in Algorithms 3.2. Where $R_{i,j}$ represents the rule corresponds to the $C_{i,j}$, CF ($R_{i,j}$) represents the CF of $R_{i,j}$

Algorithm 3.2: Recording algorithm for uncertainty fired rules

```
Input: Inference history, rule $R_{i,j}$.
Output: Uncertainty fired rules record

Step 1: Read the assigned value of $R_{i,j}$.
Step 2: Create a vector $V_{i,j}$.
    Step 2.1: Record the assigned value.
    Step 2.2: Record the inferred rule number.
    Step 2.3: Append $V_{i,j}$ to the buffer.
Step 3: Write the buffer to the record file.
Step 4: Return the record.
```

3.2.3 Variants Learning Algorithm

VODKA will take the action of adding a new object if some rule with lower CF is often inferred. After adding a new object, two operations must be considered: Changing the Attribute Data Type and Adding a new attribute. The cost of these two operations might be evaluated carefully since the big difference of effort between them. Changing attribute data type needs only modify the setting of the attribute, but adding new attribute needs to modify the original repertory grid and AOT which is very time consuming and costs much effort. Because the cost of transforming data type is less than adding new attributes, it had better choose CADT than ANA. In our approach, VODKA will firstly ask the expert if there exists another superior attribute data type .If the expert answers “Yes”, VODKA will guide the expert to find the proper attribute date type. In contrast, VODKA will acquire a new attribute and modify the acquisition table and AOT. Finally, VODKA will generate the new variant object. Algorithm 3.3 is shown as follows:

Algorithm 3.3: Variants learning algorithm (VL)

```
Input: Record, Rule $R_{i,j}$ .
Output: The new variant object.

Step 1: Count the assigned value of $R_{i,j}$ .
Step 2: Run ANO algorithm to acquire a variant of Object $Oi$ .
Step 3: Ask the expert if there is another superior attribute data type.
    If the answer is ”Yes”
        Run CADT algorithm to change the attribute data type.
    Else
        Run ANA algorithm to add a new attribute.
Step 4: Return the new variant object.
```

In conclusion, we generate the following Meta rules:

```
CF threshold Hc%., Rule inference threshold Hr%,

M1: If Cij> Hc AND CF($R_{i,j}$)< Hr
        Run Learning Algorithm for Variants ($R_{i,j}$) to acquire an object
M2: If CF($R_{i,j}$)< Hr  Run Record Algorithm($R_{i,j}$) to record the assigned value.
```

The drawback of EMCUD is that it is hard to explain the rule with lower CF. We would like to level up the CF of inferred rules. So we use the sum of rules' CF as the criteria to evaluate the outcome of a new object. So the goal is to maximize sum of each inferred rule's CF. Additionally we have to assess the number of rules and define the penalty of extra rules.

A general expression of object function is listed as follows:

Certainty Factor Sum (CFS) : $\sum_i CF(Ri)$

Object Function(OF) : $Max \frac{(CFS_{new} - CFS_{old})}{(CFS_{old})} X \frac{N_{old}}{N_{new}}$

Where

$R_{i,j}$: rule i∈ object O_i

CF($R_{i,j}$): the CF of rule $R_{i,j}$

N : the number of original and embedded rules of all objects O_i

3.3 The Method of Adding a New Object

To add a new variant object, VODKA firstly asks the expert for the new variant object's name. And then VODKA would create a new column in the acquisition table. Since the variant object is different from the original object only by the ambiguous attribute value, the variant object's acquisition table values are similar to these of original object except the ambiguous attribute. In addition, the AOT values of the variant object are the same as these of the original object. This will save much time and effort for experts to fill in the acquisition table and AOT value of the new object. After the acquisition and AOT tables are completed, VODKA would ask experts how confident they feel about the original rule of the variant object. And VODKA would ask experts if they accept the embedded rule of variant objects. Because all but one attribute's value between the variant object and original object are the same, the embedded rules of the variant object would highly conflict with those of the original object. VODKA would take action to adjust these rules according to the CF. Under the same condition, VODKA would prefer firing the rule with higher CF. If the CF of the original object rule's embedded rule is higher than that of the variant object's embedded rule, VODKA would append the negating ambiguous attribute to the predicates of the variant object's original rule. This would assure the condition that fires the variant object's original rule will not simultaneously fire the original object's embedded rule. Moreover, if the CF of original object's embedded rule is lower than that of variant objects', the embedded rule of original object will be discarded. This would assure the condition of variant object's embedded rule will fire the variant object's embedded rule, not the original object's embedded rule. The Adding a New Object (ANO) Algorithm is shown as follows:

Algorithm 3.4: Adding a New Object Algorithm (ANO)

```
Input: Inference Log.
Output: The new object.

Step 1: Ask for the name of the new variant object.
Step 2: Generate the original and possible embedded rules of the new variant object.
        Step 2.1: Create the new object's acquisition table which equals that of original
                  object.
        Step 2.2: Change the ambiguous attribute value of the variant object to the
                  assigned value.
        Step 2.3: Create the new object's AOT which equals that of original object.
Step 3: Ask for the confidence of the variant object's original rule.
Step 4: Ask the expert if the possible embedded rule of the variant object could be
        accepted?
Step 5: If the old embedded rule's CF is higher
            Append the negating ambiguous attribute to the old embedded rule.
        Else
            Discard the old embedded rule
Step 6: Output the original and embedded rules of new object.
```

Example 3.3: An example of adding a new object.

An example of adding a new object for computer worm is given as follows. The acquisition table and AOT which classify three kinds of worms: Nimda, CodeRed, and Blaster are shown in Table 3.4 and Table 3.5, respectively: The data type of " Large scaling Email" ,"DCOM RPC backdoor", and "System Reboot" are Boolean. And the data type of "IIS backdoor " and " Email attached filename" are Single Value (String). In addition, the data type of "Number of Threads" is Range of Value. Besides, "L" represents the attribute "Large scaling Email", "I" represents the attribute ":IIS backdoor", "N" represents the attribute ": Number of Threads", "D" represents the attribute ":DCOM RPC backdoor", and "S" represents the attribute " System reboot,E: Email attached filename"

Table 3.4: An example of worm classification acquisition table

	Nimda	**CodeRed**	**Blaster**
L	True	X	X
I	Web Traversal	Buffer Overflow in Index Service	X
N	X	>300	X
D	X	X	True
S	X	True	True
E	Sample.exe	X	X

Symbol definition: X: do not care

Table 3.5: An example of worm classification AOT

	Nimda	CodeRed	Blaster
L	1	X	X
I	3	3	X
N	X	1	X
D	X	X	1
S	X	2	2
E	2	X	X

Here the object Nimda is concerned. We can use EMCUD to generate the original and embedded rules as Table 3.6.

Table 3.6: The original and embedded rules of Nimda

Rule Type	**Condition**	**Action**	**CF**
Original (R0)	**L** = True,& **I** = True& **E** =Sample.exe	**Nimda**	**0.8**
Embedded(R1)	**L** = True,& **I** =True&**E**=Not Sample.exe	**Nimda**	**0.4**
Embedded(R2)	**L** = True,& **I** = **Not** True& **E** = sample.exe	**Nimda**	**0.6**

Assuming that we have the inference history about the attribute “Email Attached filename”, there are 15 assigned values shown in Figure 3.2.

```
Sample.exe,purlx.exe,purlx.exe,readme.txt,purlx.exe,Sample.exe,readme.txt

Sample.exe,purlx.exe,purlx.exe,readme.txt,Sample.exe,purlx.exe,purlx.exe,readme.txt
```

Figure 3.2: An example of Nimda inferene log

Table 3.7: The statistic of assigned values about Nimda

Assigned Value	**Count**
Sample.exe	4
purlx.exe	7
readme.txt	4

Table 3.8: The statistic of inferred rules about Nimda

Rule name	**CF**	**Count**
R0	0.8	4
R1	0.4	11
R2	0.6	0

The given threshold of the embedded rules' inference frequency is 50% and the given threshold of the embedded rule's CF is 0.7.This means the frequency of the attribute "Email Attached filename "is greater than threshold. And "purlx.exe" is certificated. VODKA takes the following steps:

VODKA: Is an object with the attribute "attached filename =purlx.exe "possible a variant of Nimda?

Expert: Yes.

VODKA: What is its name?

Expert: NimdaB

VODKA: **Add a New Object NimdaB**

VODKA will add a new object named NimdaB. All attribute values of NimdaB in Acquisition Table equal these of Nimda except the attribute “Email attached filename” which equals “purlx.exe”. In addition, all attribute values of NimdaB in AOT equal these of Nimda. They are shown in Table 3.9 and Table 3.10 respectively.

Table 3.9: The Computer Worm Classification Acquisition Table after adding NimdaB

	Nimda	NimdaB	**CodeRed**	**Blaster**
L	True	True	X	X
I	Web Traversal	Web Traversal	Buffer Overflow in Index Service	X
N	X	X	>300	X
D	X	X	X	True
S	X	X	True	True
E	Sample.exe	purlx.exe	X	X

Table 3.10: The Computer Worm Classification AOT after adding NimdaB

	Nimda	NimdaB	**CodeRed**	**Blaster**
L	1	1	X	X
I	3	3	2	X
N	X	X	1	X
D	X	X	X	1
S	X	X	3	2
E	2	2	X	X

Moreover, VODKA would decide the accepted rule by the following steps.

VODKA: Is the embedded rule of NimdaB accepted?

IF **L = True,& I = True& E =Not** purlx.exe Then NimdaB

Expert : Yes

VODKA: Add this embedded rule to the rule class NimdaB

Decide the CF.

VODKA: How confident do you feel about the original rule of NimdaB?

1. Confirm 2. Strongly support 3.Support 4. May support

Expert: 2

VODKA: If the original rule is "Strongly support", how about the embedded rule?

Expert: 3

VODKA: calculate the CF of original and embedded rule.

The original rule result of NimdaB and is accepted embedded rule

Table 3.11: The original and embedded rules of NimdaB.

Rule Type	**Condition**	**Action**	**CF**
Original(R0)	**L** = True,& **I** = True& **E** =purlx.exe	**NimdaB**	**0.8**
Embedded (R1)	**L** = True,& **I** =True&**E**=Not purlx.exe	**NimdaB**	**0.6**

The final result of NimdaB's original and embedded rules is shown in Table3.11.

In addition, there exists an embedded rule conflicting problem between Nimda and NimdaB as follows.

NimdaB R1: IF L = True,& I =True &E=Not purlx.exe THEN NimdaB CF 0.6

Nimda R2: IF L = True,& I =True &E=Not Sample.exe THEN Nimda CF 0.4

To solve the rule conflicting problem, VODKA would take the following steps.

VODKA: Since the new embedded rule's CF is greater than the old embedded rule's CF, is the NimdaB's embedded rule is fired under the condition "Not purlx.exe & Not Sample.exe" f?

Expert: Yes

VODKA: Delete the embedded rule Nimda R2

Then we have to evaluate the improvement of our approach. The object function calculation of the original result is listed as follows. Nimda R0 is inferred 4 times and its CF is 0.8. Nimda R1 is inferred 11 times and its CF is 0.4.

$$CFS_{old} = (4*0.8+11*0.4) = 7.6:$$

The new result: is summarized in Table 3.12.

Table 3.12: The statistic of inferred rules about Nimda and NimdaB

Rule name	**CF**	**Count**
Nimda R0	0.8	4
Nimda R2	0.6	0
NimdaB R0	0.4	7
NimdaB R1	0.6	4

The object function calculation of the new result is

$$CFS_{new} = (4*0.8+0*0.4+7*0.8+4*0.6) = 11.2$$

Then

$$OF = \frac{(CFS_{new} - CFS_{old})}{(CFS_{old})} X \frac{N_{old}}{N_{new}} = \frac{11.2-7.6}{7.6} X \frac{3}{4} \approx 35\%$$

We can conclude that the improvement is approximately 35**%.** This indeed improves the drawback of EMCUD.

3.4 The Method of Changing Attribute Data Type

Because of the real world application, there will be some attributes are unable to describe the object precisely. Changing attribute data type is feasible to improve this problem. Firstly, VODKA asks experts a superior attribute data type .Then VODKA will acquire the new acquisition table value. In addition, if the attribute significance order is modified after attribute data type is changed, VODKA will generate the original and embedded rules again. Finally, VODKA will output the whole new original and embedded rules. The Changing Attribute Data Type Algorithm (CADT) is shown in Algorithm 3.5:

Algorithm 3.5: Changing Attribute Data Type Algorithm (CADT)

```
Input: Inference log
Output: A changed attribute data type.
Step 1: Ask expert the new data type.
Step 2: For each object's acquisition table,
            Acquire the new attribute value.
Step 3: For each object's AOT,
            Ask experts if the order of attributes needs change.
            If the expert says "Yes", go to Step 4.
            Else, go to Step 5.
Step 4: Regenerate the embedded rules using EMCUD.
Step 5: Output the original and embedded rules.
```

Example 3.4: An example of changing the attribute data type

An example of Changing Data Type for computer worm is given as follows.

Here the object CodeRed is concerned . We can use EMCUD to generate the original and embedded rules as shown in Table 3.13.

Table 3.13: The original and embedded rules of CodeRed

Rule Type	**Condition**	**Action**	**CF**
Original (R0)	**I** = Buffer Overflow,& **N**= >300& **S** = True	**CodeRed**	**0.8**
Embedded(R1)	**I** = Buffer Overflow,,& **N** = **Not** >300&**S**= True	**CodeRed**	**0.4**
Embedded(R2)	**I** = Buffer Overflow,& **N** = >300& **S** = **Not** True	**CodeRed**	**0.6**
Embedded(R3)	**I** = **Not** Buffer Overflow,& **N** = >300& **S** = **Not** True	**CodeRed**	**0.7**

Assuming that we have the inference history about the attribute "Number of Threads", there are 15 assigned values shown in Figure 3.3.

```
300, 100, 305, 300,297,301,298,100,289,302,299,100,299,301,300
```

Figure 3.3: An example of CodeRed inferene log

Table 3.14: The statistic of assigned values about CodeRed

Assigned Value	**Count**
100	3
299	2
289	1
297	1
298	1
>300	7

Table 3.15: The statistic of inferred rule about CodeRed

Rule name	**CF**	**Count**
CodeRed R0	0.8	7
CodeRed R1	0.4	8
CodeRed R2	0.6	0
CodeRed R3	0.7	0

The given threshold of the embedded rules' inference frequency is 50% and the given threshold of the embedded rule's CF is 0.7.This means the frequency of attribute "Number of threads "is greater than threshold. And "100" is certificated. So VODKA will take the following steps:

VODKA: Is an object with the attribute "Number of Threads =100 "possible a variant of CodeRed?

Expert: Yes.

VODKA: What is its name?

Expert: CodeRedB.

VODKA: **Add a New Object CodeRedB.**

VODKA will add a new object named CodeRedB. All attribute values of CodeRedB in the acquisition table equal these of CodeRed except the attribute "Number of Threads" which equals "100". In addition. All attribute values of CodeRedB in AOT equal these of CodeRed. They are shown in Table 3.17 and Table 3.18 respectively.

Table 3.17: An example of worm classification acquisition table after adding CodeRedB

	Nimda	**CodeRed**	**CodeRedB**	**Blaster**
L	True	X	X	X
I	Web Traversal	Buffer Overflow in Index Service	Buffer Overflow in Index Service	X
N	X	>300	100	X
D	X	X	X	True
S	X	True	True	True
E	Sample.exe	X	X	X

Table 3.18: An example of worm classification AOT after adding CodeRedB

	Nimda	**CodeRed**	**CodeRedB**	**Blaster**
L	1	X	X	X
I	3	2	2	X
N	X	1	1	X
D	X	X	X	1
S	X	3	3	2
E	2	X	X	X

Since the embedded rule is still inferred over the threshold and the set fact frequency is not over the threshold, VODKA will take the following steps

VODKA: Is there any other superior data type for the attribute "Number of Thread s"?

Expert: Range

VODKA: Change to "Range"

The acquisition table and AOT after changing data type are shown in Table 3.19 and Table 3.20 respectively.

Table 3.19: An example of worm classification acquisition table after changing data type

	Nimda	**CodeRed**	**CodeRedB**	**Blaster**
L	True	X	X	X
I	Web Traversal	Buffer Overflow in Index Service	Buffer Overflow in Index Service	X
N	X	{90,110}	{290,310}	X
D	X	X	X	True
S	X	True	True	True
E	Sample.exe	X	X	X

Table 3.20: An example of worm classification AOT after changing data type

	Nimda	**CodeRed**	**CodeRedB**	**Blaster**
L	1	X	X	X
I	3	2	2	X
N	X	1	1	X
D	X	X	X	1
S	X	3	3	2
E	2	X	X	X

Moreover, VODKA would decide the accepted rule by the following steps.

VODKA: Is the embedded rule of Code B accepted?

IF **I** = Buffer Overflow,& **N= Not** {290,310}& **S** = True Then CodeRedB

Expert : Yes

VODKA: Add this embedded rule to the rule class of CodeRedB.

Decide the CF.

VODKA: How confident do you feel about the original rule?

1. Confirm 2. Strongly support 3. Support 4. May support

Expert: 2.

VODKA: If the original rule is "Strongly support", how about the embedded rule?

Expert: 3.

VODKA: Calculate the CF of original and embedded rule

The result is shown in Table 3.21.

Table 3.21: The original rule result of CodeRedB and is accepted embedded rule

Rule Type	**Condition**	**Action**	**CF**
Original(R0)	**I** = Buffer Overflow,& **N**= {290,310}& **S** = True	**CodeRedB**	**0.8**
Embedded(R1)	**I** = Buffer Overflow,& **N= Not** {290,310}& **S** = True	**CodeRedB**	**0.6**

In addition, there exists an embedded rule conflicting problem between CodeRed and CodeRedB as follows.

CodeRedB R1: IF I = Buffer Overflow,& N =Not{290,310} &S=True THEN CodeRedB CF 0.6

CodeRed R2: IF I = Buffer Overflow,& N = Not {90,110} &S=True THEN CodeRed CF 0.4

To solve the rule conflicting problem, VODKA would take the following steps:

VODKA: Since the new embedded rule'CF is greater than the old embedded rule'CF, is the CodeRedB's embedded rule fired under the conditions "Not{290,310}& Not{90,110}"?

Expert: Yes.

VODKA: Delete the embedded rule CodeRed R2.

Then we have to evaluate the improvement of our approach. The object function calculation of the original result is listed as follows. CodeRed R0 is inferred 7 times and its CF is 0.8. CodeRed R1 is inferred 8 times and its CF is 0.4.

The original result:

$$CFS_{old} = (7 * 0.8 + 8 * 0.4) = 8.8$$

The new result

{90,110}: 9 times.

{290,310}: 6 times

289: 1 time

The new result is summarized in Table 3.22.

Table 3.22: The statistic of inferred rules about CodeRed and CodeRedB

Rule name	**CF**	**Count**
CodeRed R0	0.8	9
CodeRed R2	0.6	0
CodeRed R3	0.7	0
CodeRedB R0	0.8	6
CodeRedB R1	0.6	1

The object function calculation of the new result is

$$CFS_{new} = (11 * 0.8 + 0 * 0.4 + 3 * 0.8 + 1 * 0.6) = 11.8$$

Then

$$OF = \frac{(CFS_{new} - CFS_{old})}{(CFS_{old})} X \frac{N_{old}}{N_{new}} = \frac{11.2 - 8.8}{8.8} X \frac{4}{5} \approx 22\%$$

We can conclude that the improvement is approximately 22**%.** This indeed improves the drawback of EMCUD.

3.5 The Method of Adding a New Attribute

Sometimes it is hard to discriminate between the original and variant objects after changing data type. When the condition occurs, VODKA would take the following steps to add a new attribute. Firstly, VODKA will acquire the name and data type of the new attribute. Then VODKA let experts fill in the acquisition table value of each object. To save the effort of experts, if the new attribute value of AOT is “do not care”, the original rule and embedded rules remain unchanged. In contrast, if that is not “do not care”, VODKA will acquire the importance order of each attribute. This means the experts have to regenerate the embedded rules and their CF by EMCUD. The Adding a New Attribute Algorithm (ANA) is shown in Algorithm 3.6 and an example of adding a new attribute for computer worm is shown in Example 3.5.

Algorithm 3.6: Adding a New Attribute Algorithm (ANA)

```
Input: Inference log.
Output: The new attribute.

Step 1: Ask for new attribute to discriminate the old embedded rule.
Step 2: Let experts fill in new attribute value of all objects.
Step 3: For each object
    Step 3.1 If the new attribute is “do not care”, go to Step 5.
    Step 3.2 Ask for the importance order of attributes.
Step 4: Regenerate the embedded rules using EMCUD.
Step 5: Output the original and embedded rules.
```

Example 3.5: An example of adding a new attribute.

The object Blaster is concerned. We can use EMCUD to generate the original and embedded rules as Table 3.23.

Table 3.23: The original and embedded rules of Blaster

Rule Type	**Condition**	**Action**	**CF**
Original (R0)	**D** = True,& **S** = True	**Blaster**	**0.8**
Embedded(R1)	**D** = True,& **S** = False	**Blaster**	**0.6**

Assuming that we have the inference history about the attribute "System Reboot", there are 15 assigned values shown in Figure 3.4.

```
True, False, True, False, True, False, True, False, True, False, False, False, False, True, True
```

Figure 3.4: An example of Blaster inferene log

Table 3.24: The statistic of assigned values about Blaster

Assigned Value	**Count**
True	7
False	8

Table 3.25: The statistic of inferred rules about Blaster

Rule name	**CF**	**Count**
R0	0.8	7
R1	0.6	8

The given threshold of the embedded rules' inference frequency is 50% and the given threshold of the embedded rule's CF is 0.7.This means the frequency of the attribute "System Reboot "is greater than threshold. And "false" is the highly certificated. So VODKA will take the following steps:

VODKA: Is an object with the attribute "System Reboot =False "possible a variant of Blaster?

Expert: Yes.

VODKA: What is its name?

Expert: BlasterB

VODKA: **Add a New Object BlasterB**

VODKA will add a new object named BlasterB. All attribute values of BlasterB in the acquisition table equal these of Blaster except the attribute "System Reboot" which equals "False". In addition, all attribute values of BlasterB in AOT equal these of Blaster. They are shown in Table 3.26 and Table 3.27 respectively.

Table 3.26: An example of worm classification acquisition table after adding BlasterB

	Nimda	**CodeRed**	**Blaster**	**BlasterB**
L	True	X	X	X
I	Web Traversal	Buffer Overflow in Index Service	X	X
N	X	>600	X	X
D	X	X	True	True
S	X	True	True	False
E	Sample.exe	X	X	X

Table 3.27: An example of worm classification AOT after adding BlasterB

	Nimda	**CodeRed**	**Blaster**	**BlasterB**
L	1	X	X	X
I	3	2	X	X
N	X	1	X	X
D	X	X	1	1
S	X	3	2	2
E	2	X	X	X

Moreover, VODKA would decide the accepted rule by the following steps.

VODKA: Is the following embedded rule of BlasterB accepted?

IF **D** = True & **S** = False Then BlasterB

Expert : Yes

VODKA: Add it to the rule class of BlasterB

Decide the CF.

VODKA: How confident do you feel about the original rule of BlasterB?

1. Confirm 2. Strongly support 3. Support 4. May support

Expert: 2

VODKA: If the original rule is "Strongly support", how about the embedded rule?

Expert: 3

VODKA: Calculate the CF of original and embedded rules.

The original rule result of BlasterB and the accepted embedded rule are shown in Table 3.28.

Table 3.28: The original rule result of BlasterB and the accepted embedded rule

Rule Type	**Condition**	**Action**	**CF**
Original(R0)	D= True& S = False	**BlasterB**	**0.8**
Embedded(R1)	**I =** Buffer Overflow,& **N= Not** {290,310}& **S =** True	**BlasterB**	**0.6**

However, we find the rule conflicting problem such as the Blaster's original rule equals BlasterB's embedded rule shown in Table 3.29 and Table 3.30 respectively.

Table 3.29: The original rule result of BlasterB and is accepted embedded rule

Original (R0)	**D =** True,& **S =** True	**Blaster**	**0.8**
Embedded(R1)	**D =** True,& **S =** False	**Blaster**	**0.6**

Table 3.30: The original rule result of Blaster and is accepted embedded rule

Original(R0)	D= True& S = False	**BlasterB**	**0.8**
Embedded(R1)	D= True& S = True	**BlasterB**	**0.7**

Apparently, the attributes are not sufficient. So VODKA would take action to add a new attribute:

VODKA: Are there any other attributes except "System reboote"which discriminates between **Blaster and BlasterB**?

Expert: Activity port

VODKA: **Add a New Attribute "**Activity port"

VODKA will let experts fill in the acquisition table and AOT as shown in Table 3.31, Table 3.32, respectively

Table 3.31: The acquisition table after adding a new attribute "Activity Port"

	Nimda	**CodeRed**	**Blaster**	**BlasterB**
L	True	X	X	X
I	Web Traversal	Buffer Overflow in Index Service	X	X
N	X	100	X	X
D	X	X	True	True
S	X	True	True	False
E	Sample.exe	X	X	X
A	X	X	TCP 135	UDP 69

Table 3.32: The AOT after adding a new attribute "Activity Port"

	Nimda	**CodeRed**	**Blaster**	**BlasterB**
L	1	X	X	X
I	3	2	X	X
N	X	1	X	X
D	X	X	2	2
S	X	3	3	3
E	2	X	X	X
A	X	X	1	1

Based on Table 3.31 and Table 3.32, the new original rule and embedded rules of Blaster and BlasterB are generated and listed in Table 3.33 and Table 3.34 respectively.

Table 3.33: The original and embedded rule of Blaster

Original (R0)	**D** = True,& **S** = True& **A** = TCP 135	**Blaster**	**0.8**
Embedded(R1)	**D** = True,& **S** = False& **A** = TCP 135	**Blaster**	**0.6**
Embedded(R2)	**D** = Flase,& **S** = False& **A** = TCP 135	**Blaster**	**0.4**

Table 3.34: The original and embedded rule of BlasterB

Original(R0)	D= True& S = False& **A** = UDP 69	**BlasterB**	**0.8**
Embedded(R1)	D= True& S = True& **A** = UDP 69	**BlasterB**	**0.7**
Embedded(R2)	D= False& S = True& **A** = UDP 69	**BlasterB**	**0.5**

Obviously, we can discriminate between these two objects well.

3.6 The Discussion of Operations

As mentioned above, VODKA provides three kinds of operation: ANO, CADT, and ANA .Adding a new object is the basis of the other operations. Since we would like to cost less and maximize the objection function, we have to analyze the cost of each operation. ANO will add a new column to the original acquisition table, and the attribute ordering is not changed. Experts have only to decide one original and one embedded rule and their CF. Moreover, if the attribute importance order is not changed after CADT, experts will not be asked to regenerate them which will be done automatically by VODKA. However, if the attribute importance order is changed, VODKA will call EMCUD again to regenerate these rules. Finally, because it would change the order of attributes after ANA, this operation will cost most effort because the experts have to repeat the deciding process. To sum up, VODKA will prefer choosing ANO to CADT and ANA.

Chapter 4. System Implementation

We propose the architecture to implement our VOKDA method, including the combination of DRAMMA and EMCUD.

4.1 System Architecture

As shown in Figure 4.1. The domain expertise is elicited by EMCUD. We transform these objects including their original and embedded rules into rule classes and define the relation between these rule classes. Furthermore, the meta rules are generated to assist in triggering the correct rule class. On the other hand, users can input the facts via the interface. And the inference engine will generate the related log as the input of VOKDA to generate the new variant object. This object will be put into the knowledge base to incrementally enhance our elicited knowledge base.

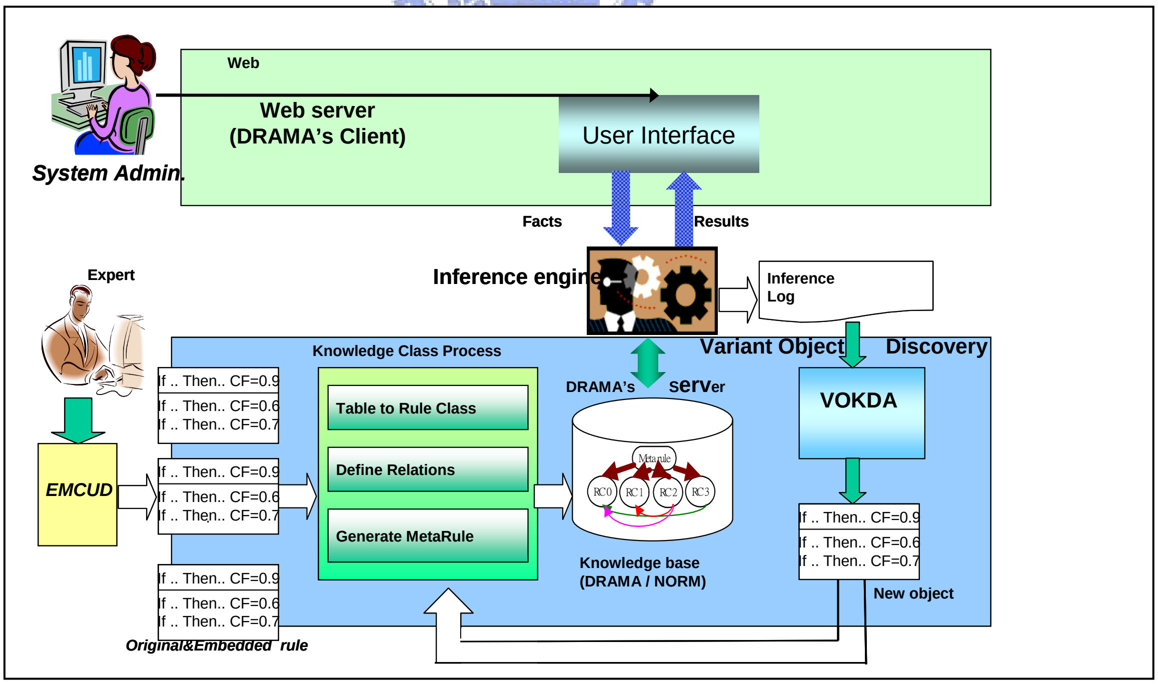


Figure 4.1: System Architecture of VODKA implementation

4.2 Expert System Shell

Knowledge representation is very important in expert systems for two reasons. First, expert system shells are designed for a certain type of knowledge representation such as rules or logic. Second, the way in which an expert system represents knowledge affects the development, efficiency, and maintenance of the system.

4.2.1 Overview of the Expert System Develop Tool – DRAMA[9]

Because of object oriented relation in DRAMA, we can clearly model the different columns in acquisition tables as the knowledge class. We use DRAMA as our expert system shell. The architecture of DRAMA is client-server and its knowledge base structure is object-oriented. The kernel knowledge model is named NORM (New Object-Oriented Rule-base Model).

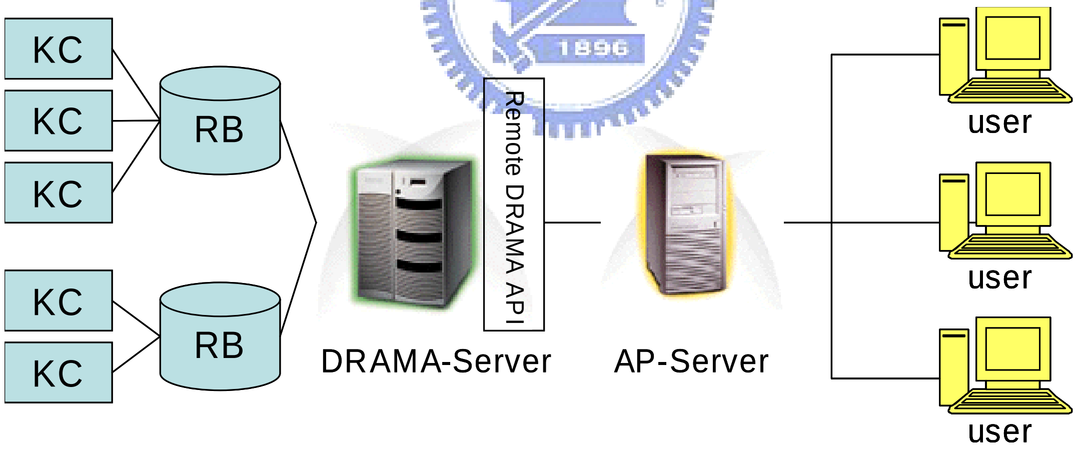


Figure 4.2: DRAMA client-server architecture

As shown in Figure 4.2, the purpose of the DRAMA's server is to load, manage and use the knowledge bases according to the knowledge service that users need. DRAMA server contains many different rule bases, and provides different APIs for the AP servers to connect. The AP server employs

DRAMA's APIs to provide user-friendly web pages for users to use expert systems.

In essence, each knowledge module is corresponding to the knowledge class (KC) structure of DRAMA. And there are four kinds of relation between knowledge classes: Acquire, Trigger, Reference, and Extension-Of.

4.2.2 Object-oriented, rule-based knowledge representation

One of the most popular types of expert systems is the Rule-based system which uses rules as its knowledge representation scheme. There are several advantages of the rule format: First, it is a natural knowledge representation, in the form of "IF … Then…" type which is similar to the human cognitive process. The second is modularity that makes it easy to construct, to debug, and to maintain expert system. Finally rules are easy to explain. Although rule-based is powerful enough in many applications, it still has several disadvantages in maintenance and construction, e.g., it is hard to incrementally construct the knowledge.

Therefore, the maintenance of rule-based expert system is a critical research issue; for example, object-oriented technology is used to integrate rule-based paradigms and object-oriented programming paradigms. The integration both of them seems a feasible way. Rule-based paradigm provides us with general facilities for deductive retrieval and pattern matching. On the other hand, the object-oriented programming provides us with a clear intuitive structure for programs in the form of class hierarchies.

4.2.3 Overview of NORM

NORM (New Object-Oriented Rule Base Model) [9]which is the knowledge representation scheme also takes advantages of object-oriented technology to provide high maintainability, reusability, share-ability, and abstraction for rule-based system.

Imitation of the way by which human learn a new concept is the key notion to develop NORM, which contains knowledge classes and the relationships between knowledge classes. In NORM, a knowledge class represents a kind of concept that human realized. The facility of relationships is that if we want to learn another new concept, we may refer to certain concepts that have been learned in our mind. By this way we can quickly build the new concept. Thus, a Rule-Base (RB) can record various knowledge concepts in a specific domain and each Knowledge Class (KC) in the RB represents different concept of the domain knowledge. The structure of NORM is shown in Figure 4.3.

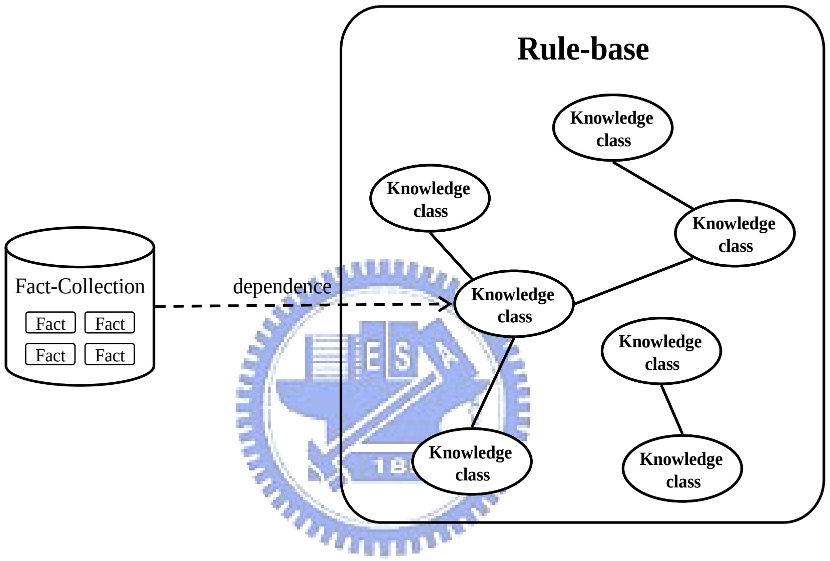


Figure 4.3: New Object-Oriented Rule Base Model

A Knowledge Class (KC) consists of rules, relations with other KCs and fact declarations as shown in Figure 4.4. The working model is composed of different KCs and the transferences (e.g., Trigger, Acquire, Reference, and Extension-of, etc.) between KCs. The relationships between KCs are divided into two kinds – dynamic and static. Trigger and Acquire are dynamic because they are activated conditionally in the action part of a rule. In a contrast, Reference, and Extension-of are static.

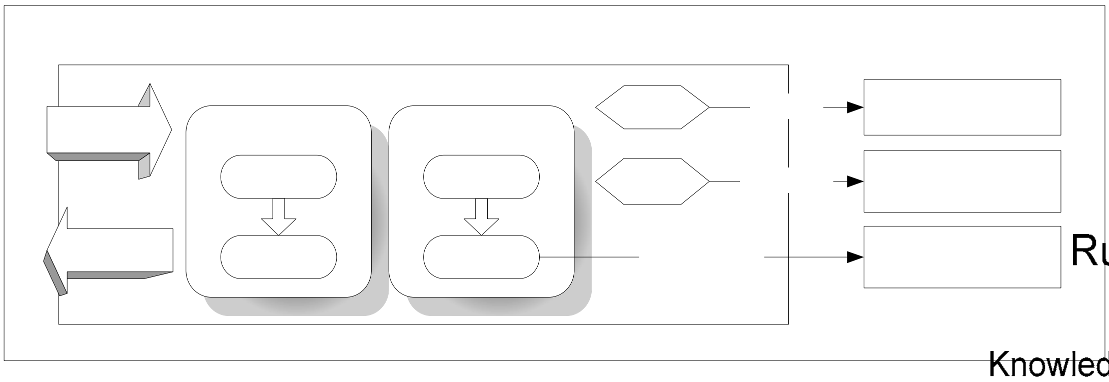


Figure 4.4: The knowledge class in a rule base

Trigger

Trigger is transferring one problem another one. Basically different problem solving methods will use different knowledge types; however, a knowledge class will contain only one knowledge type. In Trigger relationship, it triggers another knowledge class with current facts as knowledge transfer. This means that the remnant knowledge in original knowledge should not be considered. For example, the relation between meta rule and other rule classes generated by EMCUD is "Trigger"

Acquire

Acquire relationship presents the acquirement. After Acquire process, the original inference process will continue and only facts predefined in the target knowledge class will be carried back. When a problem can be divided into sub-problems, we will use the Acquire relationship to handle these conditions.

Reference

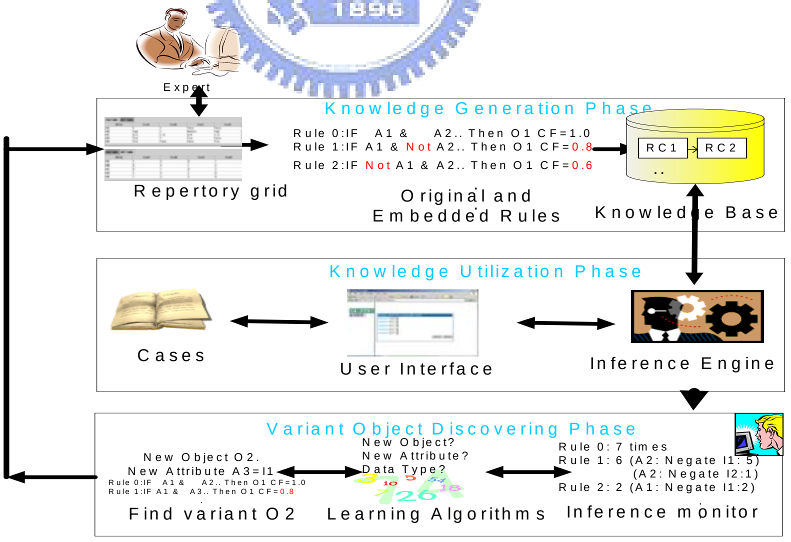
Reference represents the associations between different concepts. Through the Reference relationship, the knowledge contained in referred knowledge class is regarded as the base knowledge and it will be taken into consideration first. The variant objects' rule class have the Reference relationship with those of their original objects.

Extension-of

The activities of Extension-of relationship include extension and modification. Therefore, it supports the overriding mechanism, including the overriding of facts and rules.

4.3 The Framework of EMCUD-DRAMA-VODKA

In the following, we propose a framework, consisting of EMCUD, DRAMA, and VODKA. We integrate these into EMCUD-DRAMA-VODKA which has three phases: Knowledge Generating phase, Knowledge Utilization phase, and Knowledge Discovering phase. Firstly, in Knowledge Generating phase, EMCUD helps knowledge engineers to extract the knowledge from experts and generate original and embedded rules. Experts will be asked to fill in the acquisition and attribute ordering grids with their domain knowledge. The knowledge contained in these grids with their implicit meaning will be also extracted by EMCUD. Besides, we provide the Table validation, checking the similarity of attributes. In Knowledge Utilization phase, DRAMA is our expert system shell and its inference log is collect and analyzed In Knowledge Discovering phase, some ambiguous attributes are generated by our Variants Learning Algorithm. Variants are discovered and the original acquisition table and AOT are appropriately adjusted. The whole process is illustrated in Figure 4.5.



Expert
Knowledge Generation Phase
Rule 0:IF A1 & A2.. Then O1 CF=1.0
Rule 1:IF A1 & Not A2.. Then O1 CF=0.8
Rule 2:IF Not A1 & A2.. Then O1 CF=0.6
RC1
RC2
Repertory grid
Original and Embedded Rules
Knowledge Base
Knowledge Utilization Phase
Cases
User Interface
Inference Engine
Variant Object Discovering Phase
New Object?
New Attribute?
Data Type?
New Object O2.
New Attribute A3=I1
Rule 0:IF A1 & A2.. Then O1 CF=1.0
Rule 1:IF A1 & A3.. Then O1 CF=0.8
Rule 0: 7 times
Rule 1: 6 (A2: Negate I1: 5)
(A2: Negate I2:1)
Rule 2: 2 (A1: Negate I1:2)
Find variant O2
Learning Algorithms
Inference monitor

Figure 4.5: The framework of EMCUD-DRAMA-VODKA

4.3.1 Knowledge Generation Phase

EMCUD is used in this phase to extract the expertise and the embedded meaning. Furthermore, the definition the Meta rules is beneficial to integrate the generated rules into the OORB architecture, Then the inference engine can load and infer the right knowledge base. In addition, we also propose the Calculating Attribute Similarity Algorithm (CAS) to evaluate the attributes that experts provides .This is done by check whether the attributes is too similar to discriminate between objects well. The definition of similarity is given to calculate the similarity of pairs of attributes. Then we get the value of each pair of attributes. When these values are greater than the given threshold, a warning message is issued. The procedure is to acquire the acquisition table and AOT, and then check the similarity of each pair s columns in AOT'. The following step is to generate the original and embedded rules by EMCUD. Then the Meta rules are generated. The final step is to transform these rules into the rule classes. Moreover the links and connections among algorithms are shown in Figure 4.6.

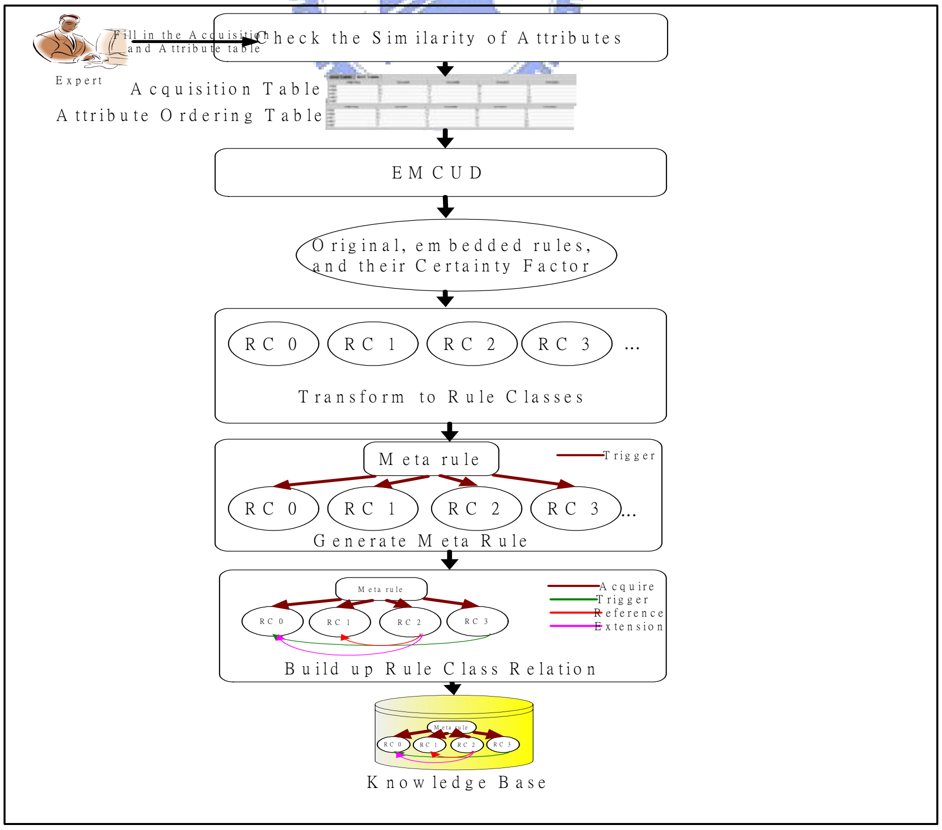

Fill in the Acquisition and Attribute table
Expert
Check the Similarity of Attributes
Acquisition Table
Attribute Ordering Table
EMCUD
Original, embedded rules, and their Certainty Factor
RC 0
RC 1
RC 2
RC 3
...
Transform to Rule Classes
Meta rule
Trigger
Generate Meta Rule
Acquire
Trigger
Reference
Extension
Build up Rule Class Relation
Knowledge Base

Figure 4.6: Links and connection between algorithms in Phase I

Algorithm 4.1: Calculating Attribute Similarity Algorithm (CASA)

```
Input: the AOT, the similarity threshold Hs
Output: the pair of similar columns

Step 1: Calculate the similarity
            For each pair of columns
                Calculate the similarity
Step 2: If the similarity > Hs
              Output this pair of attributes
```

To assure the attribute can effectively differentiate two objects, Hwang[4] defined the similarity function to calculate the similarity of two columns. The maximum possible distance between two attribute ordering columns can be derived by assuming that the worse case occurs for each pair of corresponding columns; that is, the elements of the two columns ranging from 1 to 5 are

$$C_1 = <1,2,3,4,5> \; and C_2 = <5,4,3,2,1>$$

Therefore, for the columns with n numbers, the maximum possible distance is

$$\sum_{i=1}^{n}(n-i) = \frac{n\times(n-1)}{2}.$$

Moreover, for the AOT's with m columns, the maximum possible distance is

$$m\times\sum_{i=1}^{n}(n-i) = m\times\frac{n\times(n-1)}{2}.$$

Definition: The similarity of between columns $C_{i,k}$ and $C_{j,k}$ is

$$SIMILARITY(C_{i,k}, C_{j,k}) = 1 - \frac{DISTANCE(C_{i,k}, C_{j,k})\times 2}{n\times(n-1)}.$$

And if the similarity of two columns is greater than the given threshold, a warning message is issued.

Meta Rule Creation

Meta rule which contains the information about how to trigger the correct rule class facilitates the guidance of inference. The idea is derived from the notion of maximum set of all rules, including its original and embedded rules in one rule class.

Algorithm 4.2: Generating Meta Rule Algorithm (GMRA)

```
Input: The original rule and embedded rules
Output: The meta rule

Step 1: For each attribute with an assigned value
            If all original and embedded rules contain this attribute
                Add it to the Meta rule's predicate
Step 2: Output the meta rule
```

Example 4.1: An example of constructing meta rule

Given a rule class RC_1 and its original and embedded rules

If A and B and C and D then Goal G

If A and B and C and $\neg$ *D then Goal G,*

try to find the maximum set of these rules' condition,

Then we have the meta rule:

If A and B and C and C then RC_1

4.3.2 Knowledge Utilization Phase

In this phase, the prototype system has been integrated to DRAMA to keep an inference log. The inference log, which records the count of each fired rules with lower certainty factor and the assigned facts is collected and analyzed by Recording algorithm for uncertainty fired rules. If these counters are greater than the given thresholds, we would go to Knowledge Discovering Phase. Moreover, the links connections among algorithms are shown in Figure 4.7.

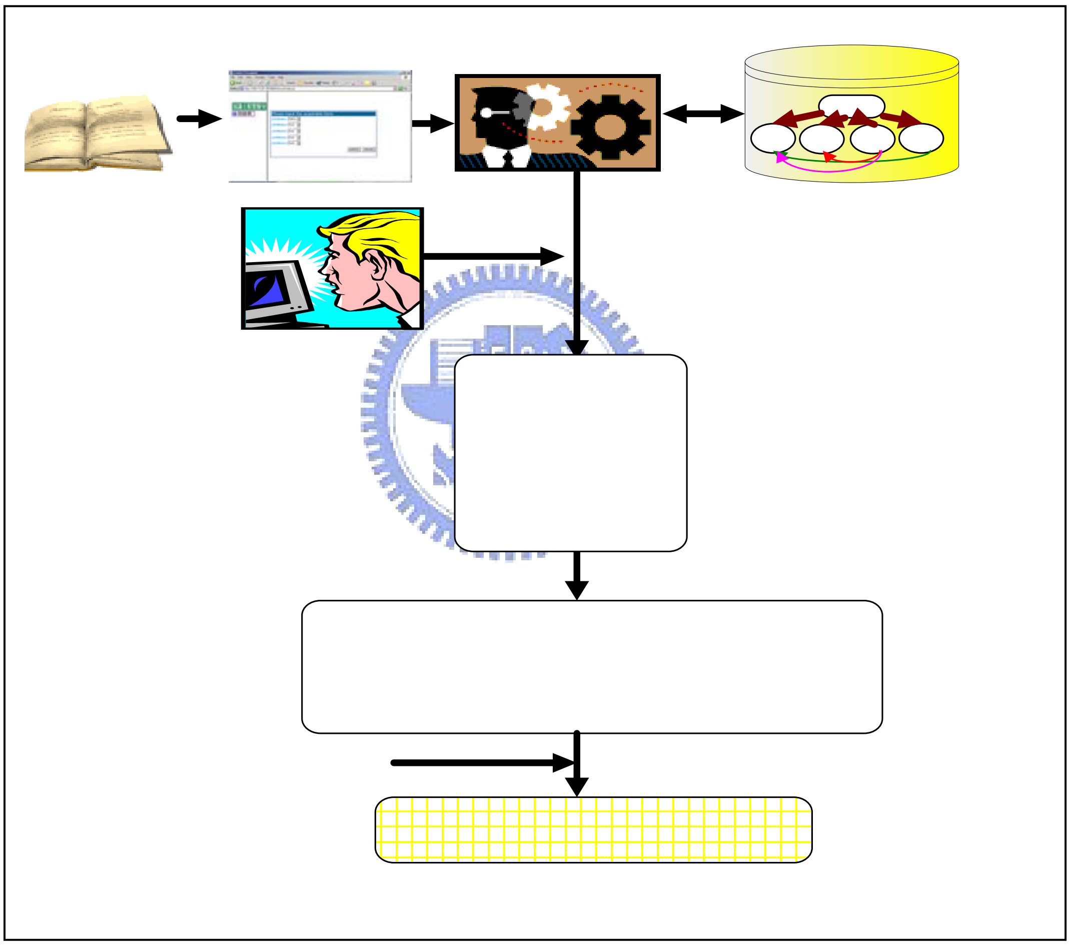


Figure 4.7: Links and connection between algorithms in Phase II

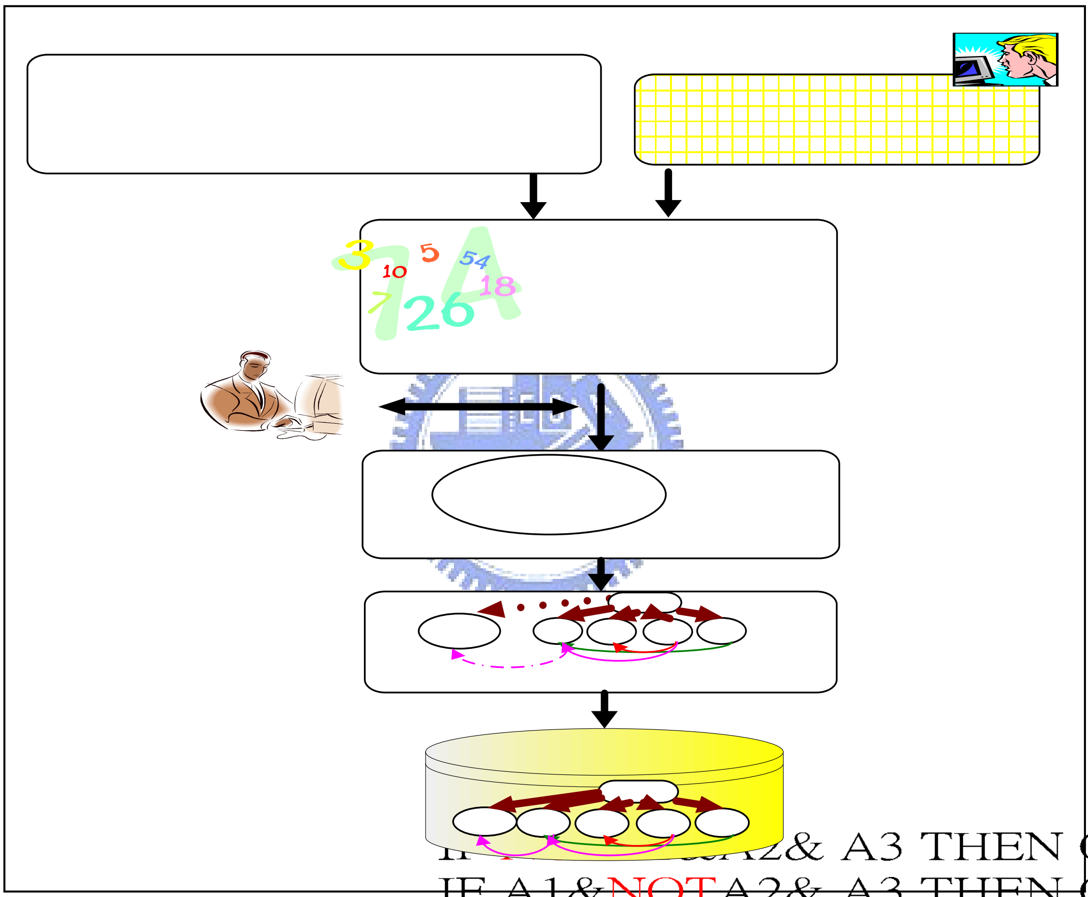
Cases User Interface Infere

Inference monitor

00:0
00:0
00:0
00:1

4.3.3 Knowledge Discovery Phase

As mentioned above, we use the Variants Learning algorithm to find the new variant object. After adding a new variant object, we have to consider the two operations: changing the attribute data type and adding a new attribute. Finally, we transform the new object to the rule class. And this will enhance the original knowledge base. Additionally the links connections among algorithms are shown in Figure 4.8.



IF A1&NOT A2& A3 THEN O1 CF=0.7
IF A1&NOTA2& A3 THEN O1 CF=0.6
IF A1&A2&NOT A3 THEN O1 CF=0.5
Embedded rules And Certainty Factor
Embedded Rules and their Certainty Factor
Learnin
Interact with Expert
Expert

Figure 4.8: Links and connection between algorithms in Phase III

Chapter 5. Case Study: Computer Worm

In this chapter, the overview of computer worm is given to briefly introduce the worm's life cycle. And then we describe the environment and developing tools of our system in the following section. Finally, an example is given to demonstrate the worm classification.

5.1 Worm Overview

A worm is a program that self-propagates across a network exploiting security flaws in widely-used services. [1][11]The Internet worm of 1988 was the first well-known replicating program that self-propagated across a network by exploiting security vulnerabilities in host software. Generally speaking, there are four stages in a worm Life-cycle: Target selection, Exploitation, Infection, and Propagation. In Target Selection Phase, a worm performs reconnaissance and simply probes potential victim to see if it's running a service on a particular port. If the service is running, the worm goes to Exploitation phase. In Exploitation Phase, a worm compromises the target by exploiting a particular vulnerability and published exploits. If succeed, the worm goes to Infection Phase. In Infection Phase, the worm sets up on the newly infected machine and goes to the Propagation Phase. In Propagation Phase, the worm attempts to spread by choosing new targets. And another victim will enter the Target Selection Phase. The procedure is shown in Figure 5.1.

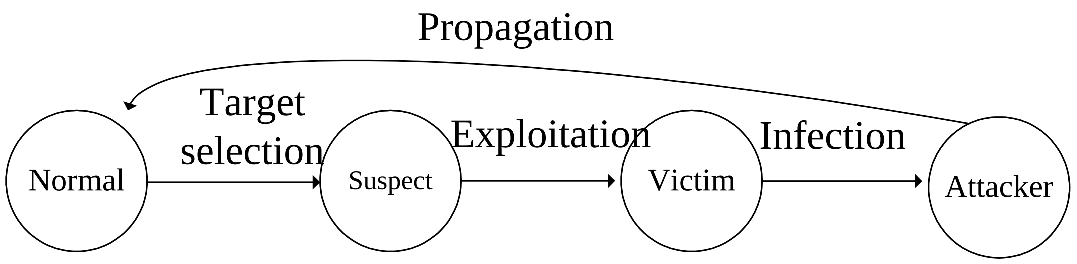


Figure 5.1: Worm Transition Stages

5.2. Experiment Environment:

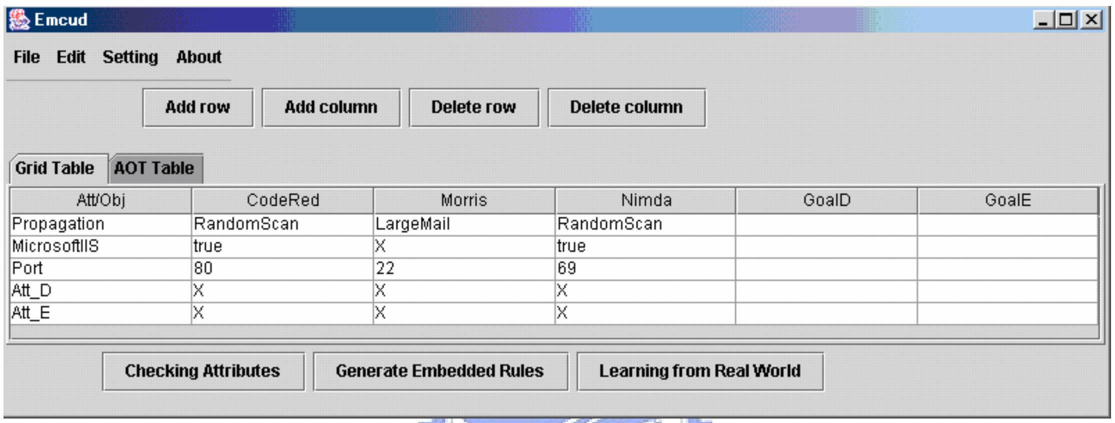
The VODKA method is now being developed with the inference engine DRAMA. The rules which are generated by EMCUD are converted to XML format because of its portability. The whole system is integrated with a user interface which is a WINDOWS based apache server. The initial protocol type of the repertory grid tool was written in JAVA programming language. It is used to implementation this knowledge acquisition system since two different features of Java. Firstly, Java is portable. So knowledge engineers can run VODKA on different platforms. In addition, it keeps language consistent with inference engine. And the programming developer can be easily maintained. The environment information is listed in Table 5.1.

Table 5.1: The related information of implementation

Attribute	**Description**	**Value**
Operating System	The OS to develop and execute the prototype system.	Platform Independent
Programming Language	The programming language used to develop the prototype system.	Java language with JDK 1.4.1
KA Approach	The Knowledge Acquisition approach.	EMCUD
The expert system shell	Rule inference engine	DRAMA 2.5

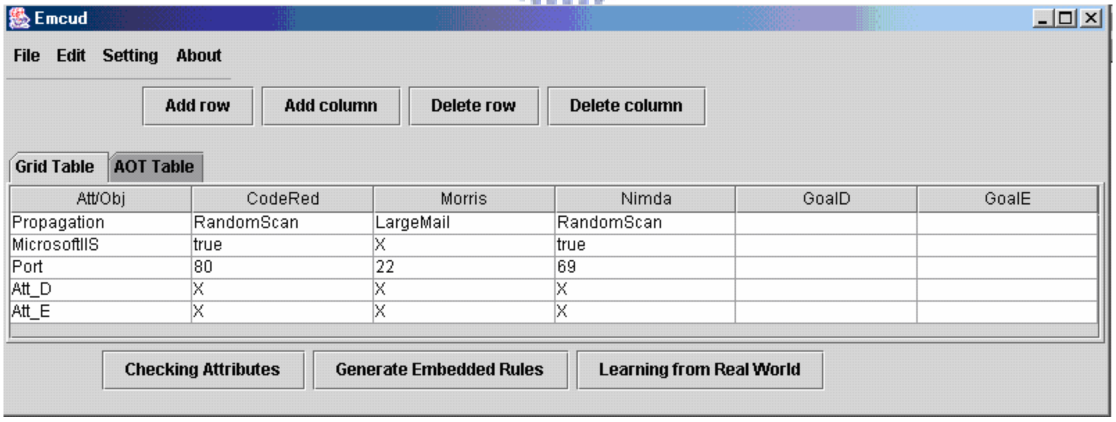
5.3. System Implementation Demonstration:

The worm classification acquisition table and AOT are shown Figure 5.2 and Figure 5.3 respectively .The mapping function setting is shown in Figure 5.4. What is more, the calculating attribute similarity is shown in Figure 5.5.The process of generating embedded rules is shown in Figure 5.6. And the process of Variants Learning algorithm is shown in Figure 5.7. Finally, the example of original and embedded rules in XML format is shown in Figure 5.8.



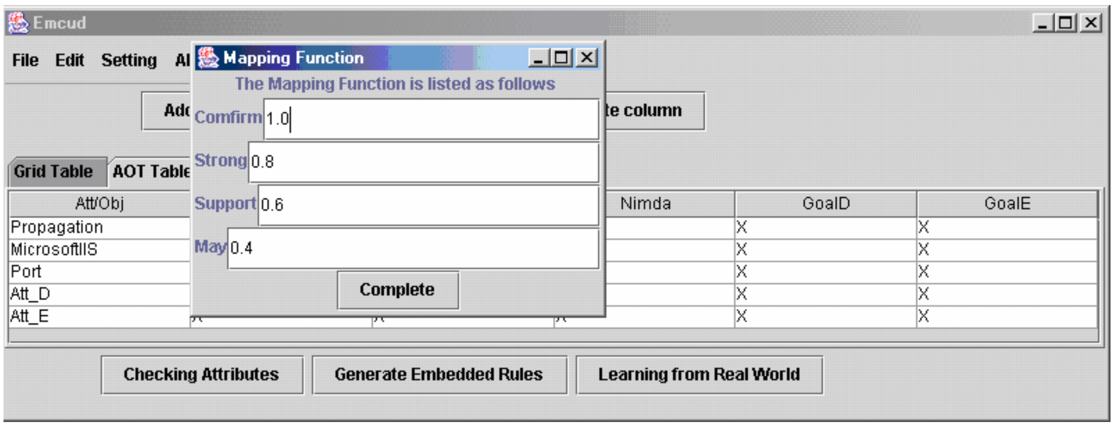
Att/Obj	CodeRed	Morris	Nimda	GoalD	GoalE
Propagation	RandomScan	LargeMail	RandomScan		
MicrosoftIIS	true	X	true		
Port	80	22	69		
Att_D	X	X	X		
Att_E	X	X	X		

Figure 5.2: An example of worm classification acquisition table.



Att/Obj	CodeRed	Morris	Nimda	GoalD	GoalE
Propagation	RandomScan	LargeMail	RandomScan		
MicrosoftIIS	true	X	true		
Port	80	22	69		
Att_D	X	X	X		
Att_E	X	X	X		

Figure 5.3: An example of worm classification AOT.

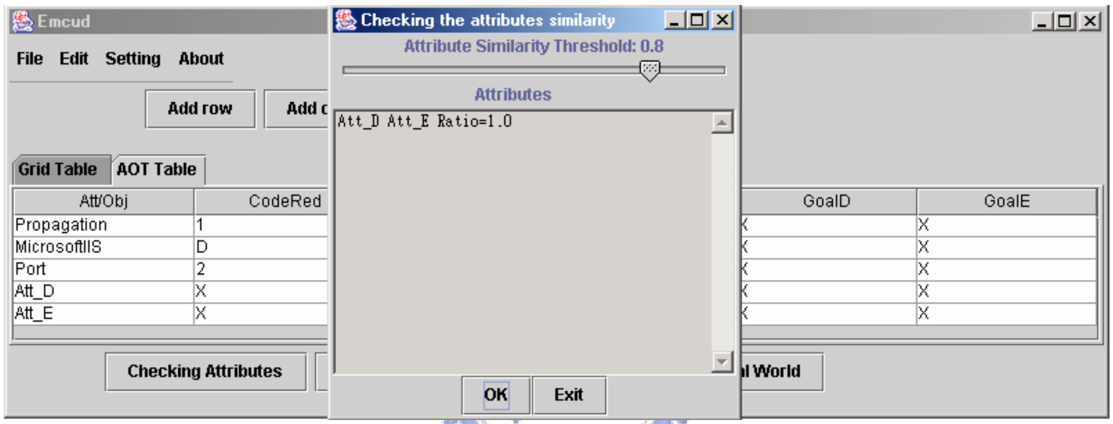


The Mapping Function is listed as follows

Comfirm	1.0
Strong	0.8
Support	0.6
May	0.4

Att/Obj	Nimda	GoalD	GoalE
Propagation		X	X
MicrosoftIIS		X	X
Port		X	X
Att_D		X	X
Att_E		X	X

Figure 5.4: An example of setting mapping function.



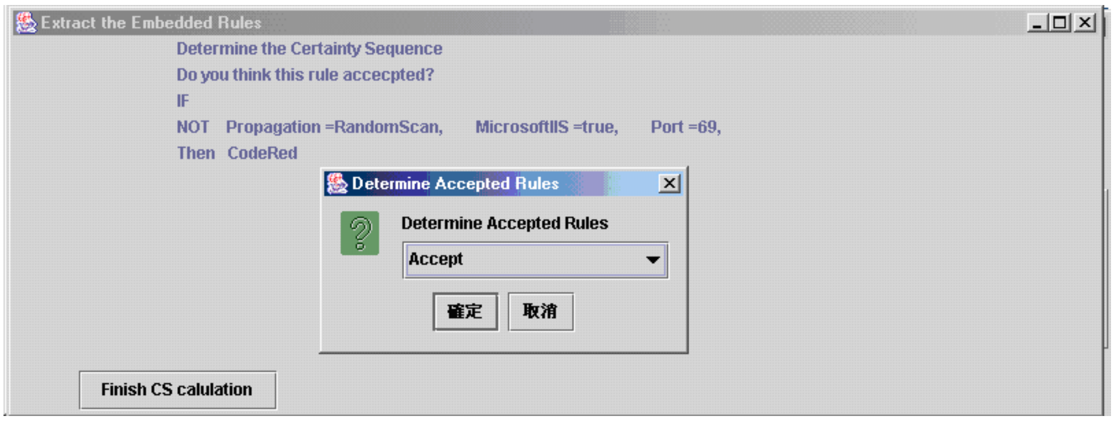
Attribute Similarity Threshold: 0.8

Attributes

```
Att_D Att_E Ratio=1.0
```

Att/Obj	CodeRed	GoalD	GoalE
Propagation	1	X	X
MicrosoftIIS	D	X	X
Port	2	X	X
Att_D	X	X	X
Att_E	X	X	X

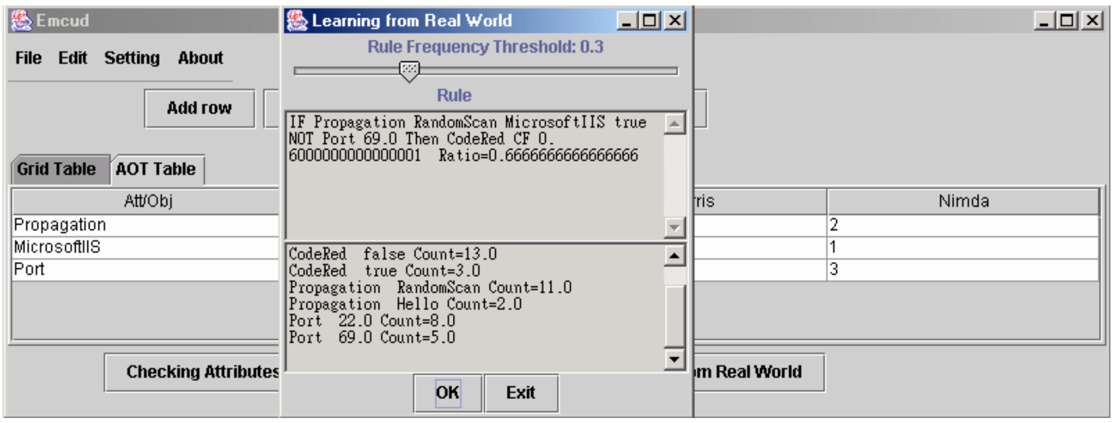
Figure 5.5: An example of calculating attribute similarity.



Determine the Certainty Sequence
Do you think this rule acccpted?
IF
NOT Propagation =RandomScan, MicrosoftIIS =true, Port =69,
Then CodeRed

Determine Accepted Rules
Accept

Figure 5.6: An example of generating embedded rules.



Att/Obj	ris	Nimda
Propagation		2
MicrosoftIIS		1
Port		3

```
IF Propagation RandomScan MicrosoftIIS true
NOT Port 69.0 Then CodeRed CF 0.
6000000000000001  Ratio=0.6666666666666666
```

```
CodeRed  false Count=13.0
CodeRed  true Count=3.0
Propagation  RandomScan Count=11.0
Propagation  Hello Count=2.0
Port  22.0 Count=8.0
Port  69.0 Count=5.0
```

Figure 5.7: An example of Variants Learning algorithm.

```
 - <RuleClass name="CodeRed">
<Fact name="CodeRed" type="String" value="false" possibility="1" />
<Fact name="Propagation" type="String" value="RandomScan" possibility="1" />
<Fact name="MicrosoftIIS" type="String" value="true" possibility="1" />
<Fact name="Port" type="String" value="69" possibility="1" />
- <Rule name="R0" weight="1" certainty-factor="1.0">
- <Condition>
- <Expression operator="And">
- <Expression operator="And">
- <Expression operator="=">
<Expression operator="Fact" type="string" value="Propagation" />
<Expression operator="Const" type="string" value="RandomScan" />
   </Expression>
- <Expression operator="=">
<Expression operator="Fact" type="string" value="MicrosoftIIS" />
<Expression operator="Const" type="string" value="true" />
   </Expression>
   </Expression>
- <Expression operator="=">
<Expression operator="Fact" type="string" value="Port" />
<Expression operator="Const" type="string" value="69" />
   </Expression>
   </Expression>
   </Condition>
- <Action>
- <Assignment>
<AssignedFact>CodeRed</AssignedFact>
- <AssignedValue>
<Expression operator="Const" value="true" type="String" />
   </AssignedValue>
   </Assignment>
   </Action>
   </Rule>
```

Figure 5.8: An example of a set of DRAMA rules

Chapter 6. Conclusion

To discover the variants objects and elevate the CF of rules, we propose a Variant Object Discovering Knowledge Acquisition. After adding a new variants object, there are two extra operations: changing attribute and adding a new attribute, which could assist knowledge engineers in improving the quality of the knowledge acquisition. In order to evaluate the performance of these operations, the object function is defined to evaluate the results. These are the basis of our experiment. We also propose a framework of EMCUD-DRAMA-VODKA which has three phases: Knowledge Generating phase, Knowledge Utilization phase, and Knowledge Discovery phase to implement our notion.

Moreover we employ VODKA on the domain of the computer worm classification. Until now this has not been applied to worm classification domain. We clarify the attributes of existing worms. A series of computer worms and their variants are identified and classified.

In addition, the prototype system has been implemented and integrated to DRAMA, a productive rule base system. The utilities provided in DRAMA enable the knowledge engineers to design and perform knowledge acquisition process based on our proposed mechanism in this work.

However, there are still some interesting problems which could be further discussed. To begin with, the multi-tier knowledge acquisition might be an interesting research topic. It needs to carefully define the hierarchy relation among these objects. And it still needs further discussion about various operations such as merging objects among different families.

References

[1] P. Crowther and J. Harthnett, “Using repertory grids for knowledge acquisition for spatial expert system” Proceeding on In Intelligent Information System, November 1996.

[2] A. Dixit and E. Pindyck, “Investment under uncertainty.” Princeton NJ. Princeton University Press, 1994.

[3] G.J. Hwang and S.S. Tseng, “On building a medical diagnostic system of acute exanthema.” Journal of Chinese Institute of Engineers, Vol. 14, No. 2, pp. 185-195, 1991.

[4] G.J. Hwang and S.S. Tseng, “EMCUD: A knowledge acquisition method which captures embedded meanings under uncertainty.” International Journal of Man-Machine Studies, Vol. 33, No. 4, pp. 431-451, 1990.

[5] G.J. Hwang, “Knowledge acquisition for fuzzy expert systems.” International Journal of Intelligent Systems. Vol. 10, pp. 541-560, 1995.

[6] J .Jose, Castro-Schez, Nicholas R. Jennings, X. Luo, and Nigel R. Shadbolt. “Acquiring domain knowledge for negotiating agents: a case of study.” International of Human-Computer Studies. September 2003.

[7] G. A. Kelly, “The psychology of personal constructs.” Vol. 1 NY:W.W Norton,1955

[8] D.M. Kienzle, M. C. Elder, “Recently worms: a survey and trends.” WORM’03, October 2003

[9] Y. T. Lin, S. S. Tseng, and C. F. Tsai, “Design and implementation of New Object-Oriented Rule base Management system.” Journal of Expert Systems with Applications, Vol. 25, pp. 369-385. 2003.

[10] D. Moore, C. Shannon, G. M. Voelker, S. Savage, “Internet quarantine: requirements for containing self-propagating Code.” INFOCOM 2003, 2003.

[11] J. Mirkovic, J. Martin, P. Reiher, “A taxonomy of DDoS attacks and DDoS defense Mechanisms.” University of California, Los Angeles　Technical report #020018

[12] D. Pan,Q. Zbeng, A. Zeng, and J. Hu, “A novel self-optimizing approach for knowledge acquisition.” IEEE Transaction on Systems, Man, and Cybernetics, Vol. 32, No. 4, pp. 505-514 ,July 2002,

[13] M. Polanyi, “The tacit dimension, London: Routledge and Kegan Paul.” New York: Anchor Books. (108 + xi pages), 1967.

[14] E.H. Shortliffe and B.G. Buchanan, “A model of inexact reasoning in medicine.” Math. Bioscience, Vol. 23, pp. 351~379, 1975.

[15] N. Weaver, V. Paxsonm, S. Staniford, and R. Cunningham and U. Maulik, “A taxonomy of computer worms.”　WORM’03, October 2003