

# 國立交通大學

資訊工程學系

博士論文

動態知識擷取方法之研究

A Study of Knowledge Acquisition Methodologies  
for Dynamic Knowledge



研究生: 林順傑

指導教授: 曾憲雄 博士

中華民國 九十五年 十月

# 動態知識擷取方法之研究

## A Study of Knowledge Acquisition Methodologies for Dynamic Knowledge

研究生: 林順傑

Student: Shun-Chieh Lin

指導教授: 曾憲雄 博士

Advisor: Dr. Shian-Shyong Tseng

國立交通大學  
資訊工程學系  
博士論文



Submitted to Department of Computer Science

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Computer Science

October 2006

Hsinchu, Taiwan, Republic of China

中華民國 九十五年 十月

# 動態知識擷取方法之研究

學生: 林順傑

指導教授: 曾憲雄 博士

國立交通大學 資訊學院

資訊工程系

## 摘要

知識擷取是在建立知識庫系統中的一個主要瓶頸。由於知識爆炸，知識可以被歸納成靜態知識 ( Static Substantive Knowledge ) 和動態知識 ( Dynamic Substantive Knowledge ) 兩大類。在過去 20 多年中，有很多的研究學者提出很多知識擷取方法，從專家那邊萃取出靜態的知識，然而，這些方法在擷取知識的過程中，因為缺乏足夠多的資訊，所以並沒有討論到如何發覺包括變種知識 ( Variant Knowledge ) 和演化性知識 ( Evolutional Knowledge ) 兩類的動態知識。因此，如何蒐集到足夠多的資訊，並用來通知專家有新演化的物件產生，而且可以利用並擴展舊有的知識庫，在知識擷取的領域中，也逐漸變成一個重要的議題。部分現存的知識擷取系統，採取建構個人建構理論 ( Personal Construct Theory ) 上發展出來的知識表格 ( Repertory Grid ) 技術來擷取在一個限定領域間，分辨並區分開不同物件的靜態知識。EMCUD ( Embedded Meaning Capturing and Uncertainty Deciding ) 是一種用來擷取隱含知識的技術。它在 1990 年被提出來協助專家萃取知識的隱含意義並協助專家決定每一條隱含規則 ( embedded rule ) 的信賴程度，用來擴展使用傳統知識表格方法產生的原始規則 ( original rule )。然而，EMCUD 一樣因為缺乏足夠多的資訊而無法擁有發現新演化物件產生的能力。我

們的想法是希望可以藉由觀察知識庫各個低信賴程度的隱含規則被推論的行為，包括頻率以及趨勢變化並藉此用來學習可能的新演化物件，然後再引導專家根據這些推論行為的趨勢來萃取便是這些物件的動態知識。在這篇博士論文中，我們將提出一個包含推論記錄檔蒐集階段、知識學習階段以及知識精鍊階段等三個階段新的知識擷取方法，*Dynamic EMCUD*，來協助專家察覺到新演化物件的產生並萃取出這些物件的隱含規則。*Dynamic EMCUD* 在推論記錄檔蒐集階段可以協助專家蒐集足夠的推論記錄。在隨著時間改變的環境中，在知識學習階段中可以透過觀察頻繁的推論行為和演化行為的趨勢，讓專家察覺到新演化物件的產生。最後，在知識精鍊的階段，*Dynamic EMCUD* 可以將一個小的多資料型態知識表格和一個小的屬性序列表格 (Attribute Ordering Table, AOT) 個別整合到一個主要的多資料型態知識表格和主要的屬性序列表格中，並用來調整弱隱含知識來達到表格演化的能力。進一步來說，我們的方法可以很容易的延伸成包括多個區域的知識庫系統和一個聯合的知識庫系統的聯合式的架構來協助整合從各個搭載 *Dynamic EMCUD* 的區域知識庫系統所產生的演化物件的知識。並且協助專家可以容以的利用足夠多的環境資訊來發覺更多其它新的物件知識。我們提出五個演算法來幫助專家容易的萃取新物件的隱含規則。電腦蠕蟲和分散式阻斷服務偵測以及警報分類模式建立兩個應用可以用來評估 *Dynamic EMCUD* 的效能，結果顯示新的變種物件可以被快速發覺並可以快速的通知專家，並協助他們利用 *Dynamic EMCUD* 萃取出新演化物件的隱含規則。

**關鍵詞：**知識擷取、知識表格、隱含知識擷取、入侵偵測、電腦蠕蟲、分散式阻斷服務

# A Study of Knowledge Acquisition Methodologies for Dynamic Knowledge

Student: Shun-Chieh Lin

Advisor: Dr. Shian-Shyong Tseng

Department of Computer Science

College of Computer Science

National Chiao Tung University

## Abstract

Knowledge acquisition is known to be a critical bottleneck of building knowledge based systems. Due to the explosion of knowledge, substantive knowledge can be classified into static substantive knowledge and dynamic substantive knowledge. Many knowledge acquisition methodologies have been proposed to systematically elicit rules of static substantive knowledge from domain experts in the past twenty years. However, none of these methods discusses the issue of discovering dynamic substantive knowledge including variant knowledge and evolutionary knowledge due to the lack of sufficient information. Hence, how to collect sufficient information to help experts notice the occurrence of new evolved objects and to reuse and extend the original knowledge base becomes increasingly important in the knowledge acquisition field. Most of the existing systems employ the Repertory-Grid test originally developed by Personal Construct Theory in eliciting static substantive knowledge to identify different objects and distinguishing these objects in a selected domain. *EMCUD* (Embedded Meaning Capturing and Uncertainty Deciding), one of a

Repertory Grid based knowledge acquisition tools, has been proposed to elicit the embedded meanings of knowledge (embedded rules bearing on objects and object attributes) to classify objects and guide experts to decide the certainty degree of each embedded rule using an attribute ordering table (AOT), which records the relative importance of each attribute to each object, for extending the coverage of original rules. However, it still lacks the ability to discover the occurrence of new evolved objects due to insufficient information. Our idea is to monitor the frequent inference behaviors and the trend of weak embedded rules with lower certainty degree and learn the candidates of new evolved objects and then guide the experts to extract the dynamic knowledge of these objects according the trend of inference behaviors. In this dissertation, we will propose a new iteratively knowledge acquisition method, *Dynamic EMCUD* which includes Log Collecting Stage, Knowledge Learning Stage, and Knowledge Polishing Stage, to notify experts to extract the embedded rules of new evolved objects. The *Dynamic EMCUD* can collect sufficient inference log in Log Collection Stage and then notify experts the occurrence of evolved objects through observing the frequent inference behaviors and tracing the trend of evolutionary behaviors over time in a changing environment in Knowledge Learning Stage. In the Knowledge Polishing Stage, the *Dynamic EMCUD* can integrate a small acquisition table increment and a small attribute ordering table (AOT) increment into the main acquisition table and the main AOT, respectively, for adapting the weak embedded rules to achieve the ability of grid evolution. Moreover, our method can be easily extended as a collaborative framework (including  $n$  local *KBSs* and a collaborative *KBS*) to integrate the new knowledge of new evolved objects generated from every local *KBSs* (each *KBS* deploy a *Dynamic EMCUD*) and help experts easily discover some other new evolved objects in the collaborative *KBS* with sufficient context. Five algorithms are proposed to help expert easily extract the embedded rules

of new objects. Two applications including in worms and distributed DoS detection, and alert classification model construction are used to evaluate the performance of *Dynamic EMCUD*. The results show that the new variants can be discovered and experts can be easily notified to quickly extract the knowledge of new objects according to the *Dynamic EMCUD*.

**Keywords:** Knowledge acquisition, Repertory grid, *EMCUD*, Intrusion detection, Computer worm, Distributed DoS



## 誌謝

盼呀盼的，終於取得夢寐以求的博士學位，對自己來說，算是完成小時候的夢想。對於家人，也算是有個交代，尤其是陪在身邊一直默默支持鼓勵我的可愛老婆 念怡，以及在去年才向這個世界報到的寶貝女兒 宸好。猶記得七年前，從未離開家鄉的我，帶著忐忑不安的心情，進入了交大校園就讀，在這浩瀚的學術殿堂裡開始了我的求學生活，期間雖然經歷了許多的風風雨雨，但仍感謝妳的體諒與包容，讓我可以無後顧之憂來完成這篇論文。對妳們無盡的感謝，絕非筆墨可以形容。

回首這碩博士班七年多的求學過程中，從對新環境的適應、人際關係的培養、資格考的歷練、領袖風範的訓練以及同儕間對於問題真理的激烈討論等，一切均一步步扎扎实實的走過。而完成這篇博士論文，最應該感謝的便是從碩士班期間，就一直旁邊諄諄教誨的恩師 曾憲雄教授，在浩瀚無涯的研究學海中，引導我朝著正確的方向前進。他不僅奠定了我在研究領域上的基礎，訓練我獨立思考解決問題以及批判突破的能力。更透過產學計畫的執行，培養我那些許的領導與統御的能力。而在待人處事方面，他也以自身為典範，讓我們在不斷的磨練與修正，讓我可以潛移默化的學習到面面俱到的解決問題的技巧。這些一切的收穫，遠遠超過完成博士論文，取得博士學位所帶來的意義；溢於文字外的心情僅能在此致上最深的感激。

此博士論文的完成，也非常感謝從論文研究計劃書口試、校內口試到校外口試一路給予我許多論文修改建議的 孫春在教授與 胡毓志教授，讓我可以全新觀點，重新檢視我的研究貢獻；在校內口試中，感謝 彭文志博士對論文的分析方法分析的重要；以及在校外口試中給予我寶貴意見的高雄大學 蘇豐文教授以及 洪宗貝教授、台南大學 黃國禎教授、與成功大學 朱治平教授，對於論文方法與結果呈現，給於精闢的見解與指導，由於他們的協助，讓此論文最後的成果能夠更加完整並增加整體論文的可讀性。

更不能忘記的，是一起奮鬥的知識工程實驗室的夥伴們，雖然每一年相處的夥伴都不盡相同，但不變的是彼此的交誼與切磋，都是我能夠順利走到這裡的助力，我也將帶著這裡的種種的回憶，往下一個人生旅程持續邁進。

僅將本篇論文的完成，獻給每一位給予我幫助及支持我的家人與朋友。



# Table of Contents

<b>Abstract (In Chinese)</b> .....	<b>I</b>
<b>Abstract (In English)</b> .....	<b>III</b>
<b>Acknowledgement</b> .....	<b>VI</b>
<b>Table of Contents</b> .....	<b>VII</b>
<b>List of Figures</b> .....	<b>IX</b>
<b>List of Tables</b> .....	<b>X</b>
<b>List of Algorithms</b> .....	<b>XI</b>
<b>Chapter 1 Introduction</b> .....	<b>1</b>
<b>Chapter 2 Related Work</b> .....	<b>8</b>
2.1 Knowledge Acquisition Systems.....	8
2.2 Repertory Grid Methodology and Relevant Systems .....	9
2.3 Elicitation of Embedded Meanings .....	12
2.4 Problems of Repertory Grid Knowledge Acquisition Methods .....	17
<b>Chapter 3 Dynamic Knowledge Acquisition Based Upon EMCUD</b> .....	<b>24</b>
3.1 The Concept of Dynamic EMCUD .....	24
3.2 Inference Log Collecting Based upon Meta Rule .....	26
3.3 The NEO-Learning Module .....	27
3.4 Grid Merging.....	32
3.5 Collaborative Framework of Dynamic EMCUD .....	33
3.6 Implementation of Dynamic EMCUD.....	39
<b>Chapter 4 Variant Knowledge Acquisition</b> .....	<b>41</b>
4.1 Idea .....	41
4.2 Variant Objects Discovering Knowledge Acquisition (VODKA).....	42
4.3 The Analysis of VODKA .....	46
4.4 Experiments.....	48
<b>Chapter 5 Evolutional Knowledge Acquisition</b> .....	<b>57</b>
5.1 Trend Evolution Analysis.....	57
5.2 Capturing Evolutional Trend Using AST.....	58
5.3 Constructing the Dynamic AOT .....	60
5.4 Adjusting Certainty Factor of Collaborative Dynamic Knowledge .....	64
5.5 Experiments.....	65
<b>Chapter 6 Application in Worms and DDoS Detection</b> .....	<b>74</b>
6.1 The Background of Worms and DDoS Attack .....	74
6.2 The Framework Worm Immune Service Expert System .....	76
6.3 DDoS Intrusion Tolerance.....	79
6.4 Knowledge Base Maintenance .....	91
6.5 Experiments.....	103
<b>Chapter 7 Application in Alert Classification Model Construction</b> .....	<b>109</b>
7.1 Introduction.....	109
7.2 Decision Support System Architecture.....	111
7.3 Experiments.....	121

<b>Chapter 8 Conclusion and Future Work .....</b>	<b>126</b>
<b>Reference.....</b>	<b>129</b>
<b>Appendix A Introduction of DDoS.....</b>	<b>136</b>
<b>Appendix B The Example of Knowledge Class in DDoS Intrusion Tolerance .....</b>	<b>139</b>
<b>Appendix C The Examples for Rule Base Partitioning .....</b>	<b>147</b>
<b>Appendix D Rule Class Construction Algorithms of Model Constructing Phase .....</b>	<b>151</b>
<b>Appendix E The Overview of The Related Tools .....</b>	<b>160</b>
<b>Appendix F The Case Study of e-Learning Using VODKA.....</b>	<b>162</b>



# List of Figures

Figure 3.2 The Flow of VODKA .....	29
Figure 3.3 The Flow of TEA .....	31
Figure 3.4 The Framework of Collaborative Knowledge Acquisition.....	34
Figure 4.1 The Time of Generating Rules Using Different Grid Size.....	48
Figure 5.1 Unfolding Step of Constructing AST .....	59
Figure 5.2 Reconstructing Step of Constructing Dynamic AOT .....	60
Figure 5.3 Worm Ontology Construction Flow.....	66
Figure 5.4 Example of Initial Nimda Concept Tree .....	69
Figure 5.5 The Updated Nimda Ontology after Discovering Nimda.B.....	71
Figure 5.6 The Updated Nimda Ontology after Discovering Nimda.E.....	72
Figure 6.1 The Collaborative Framework for Worm Detection.....	77
Figure 6.2 The Experimental Environment for Detecting Computer Worms .....	78
Figure 6.3 The Ontology of DDoS.....	80
Figure 6.4 Relationships Between of Knowledge Classes .....	81
Figure 6.5 The Framework of KA Process .....	85
Figure 6.6 An Example of Users' Behavior .....	89
Figure 6.7 The DDoS Intrusion Tolerance System Using Dynamic EMCUD .....	91
Figure 6.9 An IDS Prototype System Based RP-MES.....	105
Figure 6.11 The Performance Comparison .....	108
Figure 7.1 The Framework of Decision Support System.....	112
Figure 7.2 An Attack Tool Being Run Against Three Targets.....	114
Figure 7.3 Meta-rules of Classification Rule Classes for On-line Monitoring ...	119
Figure 7.4 Decision Support System Prototype in Experiments.....	122
Figure 7.5 Alert Reduction Rate of Normal Behavior Classification Model.....	123
Figure 7.6 Observations of Percentages of Different Suspicious Flags .....	124
Figure A.1 The General Topology of DDoS Attacks.....	137
Figure B.1 System State Diagram.....	139
Figure B.2 Role State Diagram .....	140
Figure C.1 Part of The Network Ontology.....	148
Figure D.1 Three Types of Alert Behavior Classification Rule Classes .....	151
Figure D.2 The Procedure of Normal Behavior Classification Rule Class Construction.....	154

# List of Tables

Table 2.1 The Illustrative Example of a Repertory Grid with Ratings .....	10
Table 2.2 An Example of Acquisition Table.....	14
Table 2.3 An Example of AOT .....	15
Table 2.4 The Original Rule and Embedded Rules of Nimda3.....	15
Table 2.5 The Original Rule and Embedded Rules of Nimda1 and Nimda2 .....	16
Table 2.6 The Acquisition Table of Four Computer Worms .....	19
Table 2.7 The AOT Table of Four Computer Worms.....	20
Table 2.8 Partial Detection Rules Generated by <i>EMCUD</i> .....	21
Table 2.9 The Mask Table of Ignored Attributes.....	22
Table 4.1 The Partial Inference Logs of Blaster.....	44
Table 4.2 The Partial Inference Logs of Nimda .....	49
Table 4.3 The New Variant Acquisition Table of Nimda.B.....	50
Table 4.4 The Partial Inference Logs of CodeRed .....	51
Table 4.5 The New Variant Acquisition Table of CodeRed.II .....	52
Table 4.6 The New Variant Acquisition Table of Blaster.B .....	53
Table 4.7 The Adjusted Main Acquisition Table of Simple Computer Worms ....	54
Table 4.8 AOT Table of Simple Computer Worms.....	54
Table 4.9 The Rules Generated from Table 4.7 and Table 4.8 .....	55
Table 5.1 An Example of Original Nimda AT.....	69
Table 5.2 An Example of Original Nimda AOT .....	69
Table 5.3 An Example of Nimda AST .....	70
Table 5.4 An Example of Updated Nimda AT After Discovering Nimda.B.....	71
Table 5.5 An Example of Integrated Nimda AT .....	71
Table 5.6 An Example of Updated Nimda AOT After Discovering Nimda.B.....	71
Table 5.7 An Example of Integrated Nimda AT After Discovering Nimda.E.....	72
Table 5.8 An Example of Updated Nimda AOT After Discovering Nimda.E.....	72
Table 6.1 The Ratio of Discovering New Evolved Worm.....	104
Table 6.2 The Cluster Number with Different Similarity Threshold Settings and Number of Rules.....	107
Table 6.3 Accuracy Comparisons.....	107
Table 6.4 Comparison of Number of Clusters.....	108
Table A.1 The DDoS Attacks Developed from 1998 to 2002 .....	136
Table C.1 Encodings of Expressions in RB.....	149
Table F.1 The Learning Sequence of Students.....	163
Table F.2 The Maximal Frequent Learning Patterns of Good Students .....	164

# List of Algorithms

Algorithm 2.1 EMCUD Algorithm .....	14
Algorithm 3.1 The Dynamic EMCUD Algorithm .....	26
Algorithm 3.2 The Grid Merging Algorithm.....	32
Algorithm 4.1 VODKA Algorithm.....	43
Algorithm 6.1 The Characteristic Training Algorithm.....	88
Algorithm 6.2 Rule Base Partitioning Algorithm.....	100
Algorithm 6.3 Meta Apriori Algorithm .....	102
Algorithm D.1 The Normal Behavior Classification Rule Class Construction Algorithm .....	155
Algorithm D.2 The Suspicious/Intrusion Behavior Classification Rule Class Construction Algorithm.....	159



# Chapter 1

## Introduction

As we know, knowledge based 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 solutions, such as disease diagnosis, investment prediction, and computer science. A well-known fundamental problem to the development of knowledge based systems is the acquisition of the expert knowledge (the formation of real-world knowledge to some computerized knowledge representation) that makes these systems work. As a matter of fact, knowledge acquisition is known to be the critical bottleneck in building knowledge based systems [25].

The knowledge can be divided into two groups: substantive knowledge and strategic knowledge. Substantive knowledge is used to draw conclusions from the evidence it has, that is, to interpret input data and identify the current state, and strategic knowledge is used to decide what to do next according to the state. In [30], Gruber gave a good example to explain both kinds of knowledge: A military pilot follows the strategy of taking evasive action when in danger of being fired on. The pilot must use substantive knowledge to assess the situation “Am I in danger of being attacked?” and strategic knowledge to respond “Climb to 30000 feet.” In general, substantive knowledge is used to identify relevant states of the world and strategic

knowledge is used to evaluate the utility of possible actions when a state is given. [30]

Substantive knowledge is represented explicitly in knowledge base, and is acquired directly from experts, manually or with some help from automated tools. For instance, in a computer worm detection application, the knowledge engineer consults the literature and interviews domain experts to acquire the knowledge of well-known worms (objects) using pre-defined attributes. This kind of substantive knowledge is treated as static because the environment is assumed stable in any time. In other words, the static knowledge remains the same when the environment is changed as time goes on. However, the environment of the cyber world is changing rapidly. New worms will be evolved from old worms or be developed to threaten the Internet and may cause the failure of worm detection knowledge based system. This kind of substantive knowledge is treated as dynamic knowledge which means that the knowledge will be updated or derived from well-known knowledge due to the adaptation of the changing environment with the times.

Many knowledge acquisition methodologies and related tools, e.g., *NeoETS* [10], *AQUINAS* [11], *KITTEN* [64], *EMCUD* [34], *KADS* [81], *KAMET* [14], have been proposed to improve the quality of the elicited static substantive knowledge (rules in knowledge base) in the past twenty years. Most of the existing systems employ the Repertory-Grid test originally developed by George Kelly's Personal Construct Theory [39] in eliciting substantive knowledge, which could be used as an efficient knowledge acquisition technique in identifying different objects and distinguishing these objects in a domain.

With time goes on, some substantive knowledge might be modified or evolved from the original knowledge to adapt in a dynamic environment due to the adoption of new conditions. Some other substantive knowledge could be incrementally created to classify new objects due to the explosion of the knowledge. The dynamic knowledge includes variant knowledge and evolutionary knowledge. The variant knowledge is usually derived from original objects, which means that the knowledge is changed as time goes on in the stable environment. The evolutionary knowledge is changed over time due to the changing environment. For example, in a computer worm application, a famous worm, Nimda is the first worm to modify existing web sites to start offering infected files for download by using Unicode exploit to infect IIS web server. As the time goes by, Nimda.B, a variant of Nimda family, is developed to infect victim hosts through different attached file in e-mail. Although many knowledge acquisition methods have been proposed to rapidly build the knowledge base, the acquisition of dynamic knowledge has been hardly discussed. To acquire dynamic knowledge, the experts are required to be aware of the occurrence of new objects in knowledge acquisition systems. However, it is still difficult for experts to be aware of the new object without any additional related information.

*EMCUD* (Embedded Meaning Capturing and Uncertainty Deciding) was proposed to elicit the embedded meanings of knowledge (embedded rules bearing on  $m$  objects and  $k$  object attributes) to classify  $m$  objects ( $O_1, O_2, \dots, O_m$ ) based upon repertory grids principles, which represents the information that domain experts take for granted but are implicit to the people who are not familiar with the application domain, and guide experts to decide the certainty degree of each embedded rule for extending the coverage of generated original rules. To simplify our discussion, assume



some objects in  $O_I$  class, which are classified by original rules of  $O_I$ , belong to the original object class ( $OO_I$ ) of  $O_I$ . The other objects in  $O_I$  class, which are classified by embedded rules of  $O_I$ , belong to the extended object class ( $EO_I$ ) of  $O_I$ . However, some embedded rules may be with marginally acceptable certainty factor (CF) values due to the weak suggestions of domain experts. Due to the ability of embedded rules, some objects can not be classified by the original rule but might be able to be classified by the other embedded rules with different certainty degree. Hence, these objects might be evolved with the times and could be classified by the embedded rules of  $O_I$  with weak CF values. This kind of objects is singled out to be a variant object class ( $VO_I$ ) of  $O_I$  because the similar characteristics of these objects (the related ambiguous attributes or minor attributes) might become more and more important and need to be classified into a specific variant object class in  $EO_I$  after refining the ambiguous attributes or adding some new attributes to improve the classification ability. The variant of an original object in this dissertation stands for a subset of the original object class having some different characteristics.

Although *EMCUD* extends the ability of knowledge acquisition systems to elicit substantive knowledge with the embedded rules, it is still limited to discover the dynamic knowledge of original objects due to the lack of the sufficient information. Owing to the different background and dynamic knowledge which can be changed as times goes by, the domain knowledge constructed at a time may become obsolete in the near future. Moreover, some evolutionary knowledge needs to be evolved for adopting in a dynamic environment. It may result in the difficulty of observing the occurrence of new knowledge for human experts. Hence, how to collect sufficient relevant information to help experts notice the occurrence of dynamic knowledge and

reuse the original rule base becomes one important issue. Since the relative importance of each attribute to each object could be represented using attribute ordering table, some minor attributes can be relaxed or ignored to capture the embedded meanings with acceptable CF. In other words, these kinds of attributes can be ignored to be not used to classify the object with lower CF value. With the changing environment, new knowledge derived from old objects might be classified by embedded rules with the ignored attribute-value and marginally acceptable CF, and can not be distinguished from original objects.

In this dissertation, we will propose new knowledge acquisition methods which collect useful information to monitor the inference behaviors of weak embedded rules and to trace information over time in order to efficiently update the time-related knowledge in a dynamic environment. A *Dynamic EMCUD* which is an iterative process to assist experts in being aware of the occurrence of dynamic knowledge according to the analyzing results of inference behaviors is proposed. Each iteration consists of three Stages: Log Collecting Stage, Knowledge Learning Stage, and Knowledge Polishing Phase. A collaborative knowledge acquisition framework (including local *KBSs* and a collaborative *KBS*) based upon *Dynamic EMCUD* will be proposed to monitor the frequent inference behaviors of weak embedded rules and to trace the evolved behaviors of objects with the times from multiple *KBSs* for assisting experts in efficiently obtaining the dynamic knowledge. Each local *KBS* deploys *Dynamic EMCUD* module to monitor the frequent inference behaviors of weak embedded rules to iteratively construct an acquisition table increment. The AOT increment could be constructed using entropy or time series analysis technique to analyze the importance of each attribute to each object with the times to facilitate the

acquisition and adaptation of dynamic knowledge without too many interactions with experts in a changing environment.

*Variant Objects Discovering Knowledge Acquisition (VODKA)* will be proposed to learn the new variant object in classification *KB* according to the occurrence frequency of these objects. The goal of the *VODKA* is to facilitate the acquisition of new inference rules for a classification *KBS* which identifies an object from its attribute-values in a small acquisition table increment. The new rules should be able to cope with these new objects which are similar to those previous known original rules in the *KBS* (they are object variants). Consequently, we enrich the knowledge base constructed by the *VODKA* and hence ease the effort of constructing the domain knowledge in a dynamic environment.

Because the static *EMCUD* may not be adaptive to the variant knowledge, a *Trend Evolution Acquisition (TEA)* for constructing dynamic knowledge will be thirdly proposed to adapt knowledge with time by recording each interested attribute's information in each time point and update the evolutionary knowledge base if necessary in a period of time. Consequently, a knowledge base can become more robust, flexible, and perform more learning from experiences during inference. The *VODKA* generates a small acquisition table increment of new objects, and the *TEA* generates an AOT increment. Finally, we use a *Grid Merging* approach to integrate the acquisition table increment and AOT increment into the original main acquisition table and the main AOT respectively for generating corresponding embedded rules of new objects.

However, some new evolved objects might be invisible or insignificant under each local *KBS* with *Dynamic EMCUD*, the profile of each *KBS* and the infrequent logs are analyzed in the collaborative *KBS* to collaboratively assist experts in discovering new objects. The infrequent inference logs can be analyzed by *Dynamic EMCUD* and corresponding profiles to discover the interesting knowledge of new objects which is unseen in each *KBS*. In order to acquire a meaningful CF value of each new discovered embedded rule of evolved objects, the CF value of each new embedded rule of evolved objects could be adjusted in the collaborative *KBS* based upon three cases in the CF adjusting function.

Based upon the collaborative framework, the dynamic knowledge could be elicited from the main acquisition table, which results in the ability of knowledge evolution. We illustrate two applications in worm and DDoS intrusion detection, and alert model construction to evaluate the utility of *Dynamic EMCUD*. We setup an experimental environment consisting of a firewall to filter computer worm traffic from Internet (normal traffic) and an attacking traffic generator to randomly generate various worms to infect a victim according the constructing models. The results show the *Dynamic EMCUD* is useful for assisting experts easily to be aware of the new variant worms and the corresponding knowledge can be quickly extracted.

# Chapter 2

## Related Work

Several knowledge acquisition methodologies and related systems are introduced in this chapter. Then Repertory Grid, one of the popular indirect knowledge acquisition techniques, and the elicitation of embedded meaning and some problems of traditional knowledge acquisition methodologies are discussed.

### 2.1 Knowledge Acquisition Systems

Since the knowledge in many domains, the experience of domain experts, is continuously growing, many knowledge acquisition methodologies have been proposed to help knowledge engineers acquire the useful knowledge and then to transfer these knowledge into knowledge base or other computerized representation forms. In general, there are three approaches for knowledge acquisition [21][34][48]:

- (1) Interviewing experts by experienced knowledge engineers: interviewing experts is usually time-consuming if the communication between domain experts and knowledge engineers is insufficient.
- (2) Machine learning: learning the knowledge by collecting many useful cases and instances with/ without the involvement of domain experts [57]. However, the quality of the results usually relies on the selected training cases.
- (3) Knowledge acquisition systems: assisting domain experts to generate useful rules

using knowledge acquisition systems with/ without the help of knowledge engineers. These tools could reduce the effort of communication between knowledge engineers and domain experts and could reduce the risk and difficulty of selecting the suitable training cases.

The interviewing approach could be used to acquire dynamic knowledge by manually rebuilding the knowledge base. However, the experts may not be aware of the occurrence of dynamic knowledge. Since the machine learning is used to learn the knowledge from useful cases, the discovered knowledge is limited to classify the new evolutionary knowledge. This is caused by the lack of insufficient context information. In the past decades, many knowledge acquisition systems, e.g., *NeoETS* [10], *AQUINAS* [11], *KITTEN* [64], *EMCUD* [34], *KADS* [81], *MCRDR* [38], *KAMET* [14], *MedFrame/CADIAG-IV* [7][41][44] have been developed to build prototypes and to iteratively elicit the knowledge from domain experts. However, all of these systems can not efficiently acquire dynamic knowledge due to the lack of sufficient information and the experts may not be aware of the occurrence of evolutionary knowledge.

## **2.2 Repertory Grid Methodology and Relevant Systems**

Repertory Grid, based on Kelly's Personal Construct Theory [39] which reports how people make sense of the world, could be used as an efficient knowledge acquisition technique in identifying different objects and distinguishing these objects in a domain. It is the basis of several computer assisted knowledge acquisition tools, such as *ETS* [8][9], *AQUINAS* [11] and *KSSO* [26].

A single repertory grid represented as a matrix whose columns have element objects (labels) and whose rows have construct attributes (labels) can classify a class of objects, or individuals. The value assigned to an element-construct pair need not be Boolean. Grid values have numeric ratings, probabilities, and other characteristics, where each value reflects the degree of a construct to an element. Then, the expert is asked to fill the grid with 5-scale ratings, where “1” represents the most relevant attribute to the object; “2” represents that the attribute may be relevant to the object; “3” represents “unknown” or “no relevance”; “4” represents that the object may have the opposite characteristic; “5” represents the most relevant opposite characteristic to the object. The whole concept of Repertory Grid technique can be described as following steps:

- (1) Elicit all of the element objects, e.g.,  $E_1, E_2, E_3, E_4, E_5$  from the expert.
- (2) Elicit the construct attributes (and their opposites), e.g.,  $C_1, C_2, C_3, C_4 (C_1', C_2', C_3', C_4')$ , from the expert. Each time three elements are chosen to ask for a construct to distinguish one element from the other two.
- (3) Rate all of the [element, construct] entries of the grid with value range from 1 to 5.

An illustrative example is given in Table 2.1.

**Table 2.1 The Illustrative Example of a Repertory Grid with Ratings**

Element Construct	$E_1$	$E_2$	$E_3$	$E_4$	$E_5$	
$C_1$	5	1	5	1	1	$C_1'$
$C_2$	4	4	4	1	4	$C_2'$
$C_3$	1	4	5	1	4	$C_3'$
$C_4$	1	4	4	5	5	$C_4'$

As Repertory Grid technique has been widely used by researchers, some extensions have been made to enrich its representative ability for covering more

knowledge, the value assigned to an element-construct pair may be Boolean, numeric ratings, probabilities, etc. For example, Dixit and Pindyck [23], and Hwang [33] extended the Repertory Grid technique to the fuzzy table, in which constructs were fuzzy attributes that could be rated by means of fuzzy linguistic terms from a finite set. Castro-Schez et al. [15] developed a technique using a fuzzy repertory grid for acquiring the finite set of attributes or variables that the expert used in characterizing and discriminating a set of elements.

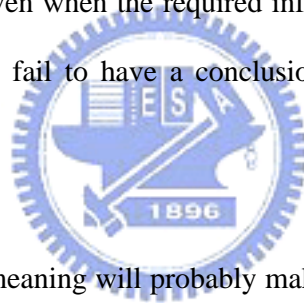
Moreover, several models have been proposed for handling uncertainties in expert systems through generating more meaningful rules from the Repertory Grid oriented approaches. *EMYCIN* certainty factor model was first used to decide on the degree of the belief of a rule for uncertain reasoning [67]. *Embedded Meaning Capturing and Uncertainty Deciding (EMCUD)* knowledge acquisition system was proposed to extract rules with embedded meaning from repertory grids by defining the impacts of the constructs to each element [34] and was successive applied in a medical diagnostic system for acute exanthema [35]. WebGrid, Calgary's web-based knowledge modeling and inference tool, is based on Repertory Grid elicitation and analysis [65].

However, none of these methodologies discusses the issue of discovering dynamic knowledge. Therefore, a new collaborative knowledge acquisition system based upon *EMCUD* is hence proposed in this dissertation to help domain experts be aware of the occurrence of dynamic knowledge and to create additional attributes for extracting dynamic knowledge through the observations of the interested inference results and the time based analysis.



## 2.3 Elicitation of Embedded Meanings

The embedded meanings referred to here represent the information that domain experts take for granted but are implicit to the people who are not familiar with the application domain. For example, a physician may describe the typical feature of Measles to be 3 to 4 days of fever, cough, desquamation, and brick-red maculopapular, but usually he/she does not mean only when all of these features happen, then the patient has Measles. It is possible that patient does not have a cough or desquamation while Measles can still be implied only with less certainty. Embedded meanings are likely to be ignored during the process of knowledge acquisition, especially in some application domains such as medical diagnosis. This is the reason why experts can usually make a conclusion even when the required information is not complete while most of expert systems may fail to have a conclusion if the premise part are only partially matched.



The lack of embedded meaning will probably make an expert system fail to infer some cases being trivial to experts. The initial knowledge and the embedded meanings will make the same conclusion with different certainties; therefore, their relationships may be used to guide experts to decide the degree of certainties for embedded meanings. *SEEK* [59] and *SEEK2* [28] have been proposed to obtain embedded meanings by some efficient refinement processes. However, the major problem with *SEEK* and *SEEK2* is the case database being assumed to be available because it is difficult to collect sufficient cases in some applications.

Moreover, it would be also time-consuming and boring for experts to offer a conclusion for each case in the database before starting the refinement procedure.

Thus, *EMCUD* is proposed to elicit the embedded meanings of knowledge from the existing hierarchical repertory grids given by experts [34]. Additionally, it will also guide experts to decide the certainty degree of each rule with embedded meaning for extending the coverage of generated original rules. *EMCUD* can be used to elite part of dynamic knowledge since the embedded meaning included. However, it is still weak to acquire more dynamic knowledge due to insufficient context information.

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. The values in each AOT entry, a pair of attribute and object, may be labeled “X”, “D” or an integer number. “X” means no relationship existing between the attribute and the object. “D” means that the attribute dominates the object, i.e., if the attribute is not equal to the entry value, it is impossible for the object to be implied. Integer numbers are used to represent for the relative important degree of the attribute to the object instead of dominating the corresponding object. If the attribute does not equal the attribute-value, it is still for the object to be implied. The larger integer number implies the attribute being more important to the object.

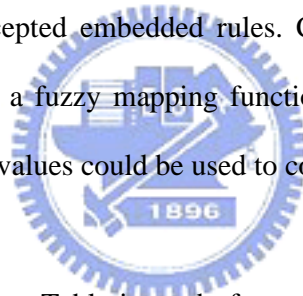
Using AOT, the original rules generate some rules with embedded meaning, and the Certainty Factor (CF) of each rule, which is between -1 and 1, could be determined to indicate the degree of supporting the inference result. The higher CF is, the more reliable result is. The *EMCUD* algorithm is listed as Algorithm 2.1.

### Algorithm 2.1 EMCUD Algorithm

**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 automatically the CF of each rule with embedded meaning.

All rules generated by *EMCUD* could be categorized into two classes: original and embedded rules with acceptable CF value, and discarded rules with unacceptable CF value, according to the confidence degree of domain experts. To decide the CF value of each embedded rule, we have to first decide on the upper and the lower bounds of CF values of accepted embedded rules. CF values of each rule can be automatically determined by a fuzzy mapping function. Thus, the useful embedded rules with corresponding CF values could be used to cover more uncertainty cases.



It is called the Acquisition Table instead of repertory grid to distinguish it from the grids derived by applying other methods. An example of acquisition table for Nimda, a worldwide popular computer worm, is shown in Table 2.2, and the other example of AOT is shown in Table 2.3.

**Table 2.2 An Example of Acquisition Table**

Attribute \ Object	Nimda1	Nimda2	Nimda3
<b>Mail_Attachment</b>	Readme.exe	puta!!.scr	null
<b>Upload_Medium</b>	Admin.dll	Admin.dll	cool.dll
<b>Executed_File_Name</b>	Riched20.dll	Riched20.dll	httpodbc.dll

**Table 2.3 An Example of AOT**

Attribute \ Object	Nimda1	Nimda2	Nimda3
Mail_Attachment	2	1	X
Upload_Medium	3	4	3
Executed_File_Name	3	4	D

The original and embedded rules generated according to Nimda3 in the third column of the Table 2.2 are shown as Table 2.4.

**Table 2.4 The Original Rule and Embedded Rules of Nimda3**

Rule #	Conditions			Conclusion	CF
	Mail_Attachment	Upload_Medium	Executed_File_Name	Object	
R <sub>Nimda3,0</sub>	null	cool.dll	httpodbc.dll	Nimda3	0.8
R <sub>Nimda3,1</sub>	-	cool.dll	httpodbc.dll	Nimda	0.6
R <sub>Nimda3,2</sub>	-	¬(cool.dll)	httpodbc.dll	CodeRed	0.4

The original rule of Nimda3 is numbered as R<sub>Nimda3,0</sub> “IF (Mail\_Attachment = null) AND (Upload\_Medium = cool.dll) AND (Executed\_File\_Name = httpodbc.dll) THEN Nimda3” with the CF = 0.8.

Since the ordering value between Mail\_Attachment and Nimda3 is “X”, Mail\_Attachment in the premise of original rule should be eliminated, and Upload\_Medium can be negated as R<sub>Nimda3,2</sub> when its ordering value is nether “D” nor “X”; however, Executed\_File\_Name should always exist in every embedded rule because it dominates Nimda3. After that, a Certainty Sequence (CS) value is used to represent the degree of certainty for an embedded rule, which is calculated by negating some predicates of its original rule by following formula (2.1),

$$CS(R_i) = \sum(AOT[Att_k, Obj_j]) \quad (2.1)$$

where Att<sub>k</sub> belongs to the attribute set of R<sub>i</sub>, and Obj<sub>j</sub> is the object of R<sub>i</sub>. So, from above example, the CS value of R<sub>Nimda3,2</sub> is 3 + 5 = 8.

Finally, after all the CS values are calculated, rules would be sorted according to the CS values and interacted with experts by acquiring upper-bound (UB) and lower-bound (LB). The CF value can now be generated by the following formula (2.2),

$$CF(R_i) = UB(R_i) \left( \frac{CS(R_i)}{MAX(CS)} \times (UB(R_i) - LB(R_i)) \right) \quad (2.2)$$

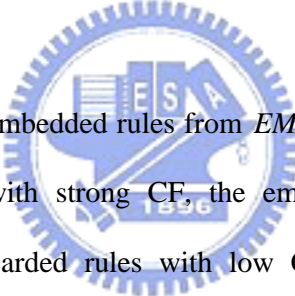
where  $MAX(CS_i)$  is the maximum CS value in all embedded rules generated from the original  $R_a$  with the same object. So, CF value of  $R_{Nimda3, 2}$  is 0.4 after the calculation, and part of results followed the example above are shown as Table 2.5.

**Table 2.5 The Original Rule and Embedded Rules of Nimda1 and Nimda2**

Rule #	Conditions			Conclusion	CF
	Mail Attachment	Upload Medium	Executed File Name	Object	
$R_{Nimda1, 0}$	Readme.exe	Admin.dll	Riched20.dll	Nimda1	0.8
$R_{Nimda1, 1}$	$\neg$ Readme.exe	Admin.dll	Riched20.dll	Nimda1	0.7
$R_{Nimda2, 0}$	puta!!.scr	Admin.dll	Riched20.dll	Nimda2	0.8
$R_{Nimda2, 1}$	$\neg$ puta!!.scr	Admin.dll	Riched20.dll	Nimda2	0.6

Since embedded rules with weak acceptable CF values usually mean domain experts may lack the strong confidence, objects matching weak embedded rules derived from original objects may be the candidates of new variants. For example, the object satisfying the conditions of the embedded rules with  $CF = 0.5$  means the expert might suggest that it would be marginally classified into the object class and the negated attributes of the embedded rule might be not clearly defined. Therefore, the fired frequencies of this kind of weak embedded rules should be used to discover the occurrence of new variant objects.

With the changing environment, the adaptation of the acquired rules should be required to cope with the dynamic knowledge. However, experts may not be aware of the occurrence of the candidates of new variants and may have insufficient evidence to construct the dynamic knowledge of the variants using conventional repertory grid approaches. Although *EMCUD* could be used to generate more useful embedded rules for covering more similar cases, it still lacks the ability of grid evolution for coping with new dynamic knowledge; e.g., *EMCUD* should manually regenerate the original and embedded rules again by the interaction with domain experts after collecting sufficient information about these knowledge. Therefore, enhancing the adaptation ability of embedded rules becomes increasingly important to achieve the ability of grid evolution in classification *KBS*.



In this dissertation, the embedded rules from *EMCUD* are categorized into three classes: the original rules with strong CF, the embedded rules with marginally acceptable CF, and the discarded rules with low CF. Hence, a new knowledge acquisition methodology is proposed to discover the occurrence of new variant objects using the fired frequency of embedded rules with marginally acceptable CF.

## **2.4 Problems of Repertory Grid Knowledge Acquisition Methods**

With the changing environment, the adaptation of the acquired rules should be required to cope with the new variants. However, experts may not be aware of the occurrence of the candidates of new variants and may have insufficient evidence to construct the knowledge of the variants using conventional Repertory Grid approaches. Although *EMCUD* could be used to generate more useful embedded rules for covering more similar objects in extended object class, it still lacks the ability of

grid evolution for singling these new variants out; e.g., *EMCUD* should manually regenerate the original and embedded rules to classify these variant objects by the interaction with domain experts after collecting sufficient information about these variants. Therefore, enhancing the adaptation ability of embedded rules becomes increasingly important to achieve the ability of grid evolution in classification *KBS*.

In this dissertation, the embedded rules from *EMCUD* are categorized into three classes: the original rules with strong CF, the embedded rules with marginally acceptable CF, and the discarded rules with low CF. Hence, a new knowledge acquisition methodology is proposed to discover the occurrence of new variant objects using the fired frequency of embedded rules with marginally acceptable CF. A simple computer worm detection prototype in Example 2.1 is used to illustrate the inability for discovering variants using *EMCUD*.

### **Example 2.1 The Example of Classifying Four Computer Worms**

In recent years, computer worm is dramatically increasing to threaten the reliability of Internet. Table 2.6 shows the acquisition table of four computer worms [40][51][50][80] including Nimda, CodeRed, Blaster, and Welchia using five attributes including 300-thread, System reboot, DoS type, Mail\_Attachment, and TCP port. The 300-thread means 300 threads with Boolean are simultaneously executed by one program. The system reboot Boolean attribute will be set to True if the system has been automatically rebooted. The attacking methodologies of worms could be classified into one kind of DoS type with String attribute [50]. The email attached file attribute with Set data type is also a useful attribute to classify these worms. Most of worms could communicate each other using different TCP port with Set data type.

**Table 2.6 The Acquisition Table of Four Computer Worms**

Object \ Attribute	Nimda	CodeRed	Blaster	Welchia
300-thread (A <sub>1</sub> )	X	True	X	X
System reboot (A <sub>2</sub> )	X	True	True	True
DoS type (A <sub>3</sub> )	Email flood	TCP flood	Windows Update flood	ICMP flood
Mail_Attachment (A <sub>4</sub> )	{sample.exe; puta!!scr}	X	X	X
TCP port (A <sub>5</sub> )	X	{80}	{135;4444}	{80;135}

An example of constructing an AOT table from the acquisition table shown in Table 2.6 is given as follows:

**EMCUD:** If DoS type is not equal to Email flood, is it possible for Nimda to be implied?

**EXPERT:** No.

The answer means the DoS type dominate Nimda, and hence AOT [Nimda,DoS type] = "D".

**EMCUD:** If Email attached file is not equal to any element of {sample.exe, puta!!scr}, is it possible for Nimda to be implied?

**EXPERT:** YES.

The answer means that Email attached file does not dominate Nimda. The questions for 300-thread and Nimda will not be asked, since the entry [Nimda, 100-thread] is labeled "X". Therefore, the entry AOT [Nimda, 300-thread] is labeled "X", too. This is the same as the entries AOT [Nimda, System reboot] and AOT [Nimda, TCP port]. The entry AOT [Nimda, Mail\_Attachment] is set to be 1, since the Email attached file is the only attribute that does not dominate Nimda. If there are more than one attributes do not dominate the object, e.g. the System reboot, the DoS



type, and the TCP port do not dominate Blaster, the following questions will be asked by *EMCUD*.

- (1) Is System reboot more important than DoS type?
- (2) Is System reboot less important than DoS type?
- (3) Is System reboot as important as DoS type?

The expert indicates that System reboot is as important as DoS type to Blaster. Moreover, the expert also indicates that System reboot is more important than TCP port to Blaster; and hence the entries  $AOT [Blaster, System\ reboot] = AOT [Blaster, DoS\ type] = 2$  and  $AOT [Blaster, TCP\ port] = 1$ . After each entry value of AOT is determined, shown in Table 2.4, the embedded meaning implied by the AOT could be extracted.



**Table 2.7 The AOT Table of Four Computer Worms**

Object Attributes	Nimda	CodeRed	Blaster	Welchia
A <sub>1</sub>	X	2	X	X
A <sub>2</sub>	X	1	2	2
A <sub>3</sub>	D	1	2	1
A <sub>4</sub>	1	X	X	X
A <sub>5</sub>	X	X	1	2

Now we use the first column of Table 2.6 to show the information implied by an AOT. The column expresses the following meanings:

- (1) A<sub>3</sub> dominates Nimda: If A<sub>3</sub> is not equal to Email flood, it is impossible for Nimda to be implied.
- (2) A<sub>4</sub> does not dominate Nimda: If A<sub>4</sub> is nether equal to sample.exe nor puta!!scr, it is still possible for Nimda to be implied.

In practice, the hierarchy rules could be generated while hierarchical grids are given. To simplify the discussion, Table 2.8 shows partial detection rules (simple rules) of a classification *KBS* based upon the Table 2.6 and Table 2.7 to classify these worms using five attributes in single grids.  $R_{i,j}$  represents the  $j$ -th highest rank of CF in object  $i$ , and the highest rank is 0. The  $R_{1,0}$  is the original rule of Nimda to classify the original Nimda objects and  $R_{1,1}$  is the embedded rule of Nimda to classify the extended Nimda objects.

**Table 2.8 Partial Detection Rules Generated by *EMCUD***

Rule #	Conditions					Conclusion	CF
	A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>	A <sub>4</sub>	A <sub>5</sub>	Object	
$R_{1,0}$	-	-	Email flood	(sample.exe;puta!!scr)	-	Nimda	0.8
$R_{1,1}$	-	-	Email flood	$\neg$ (sample.exe;puta!!scr)	-	Nimda	0.4
$R_{2,0}$	True	True	TCP flood	-	-	CodeRed	0.8
$R_{2,1}$	True	False	TCP flood	-	-	CodeRed	0.6
$R_{2,2}$	False	True	TCP flood	-	-	CodeRed	0.4
$R_{2,3}$	True	False	$\neg$ (TCP flood)	-	-	CodeRed	0.4
$R_{3,0}$	-	True	Windows update flood	-	{135;4444}	Blaster	0.7
$R_{3,1}$	-	True	Windows update flood	-	$\neg$ {135;4444}	Blaster	0.57
$R_{3,2}$	-	False	Windows update flood	-	{135;4444}	Blaster	0.43
$R_{3,3}$	-	True	$\neg$ (Windows update flood)	-	{135;4444}	Blaster	0.43
$R_{3,4}$	-	False	Windows update flood	-	$\neg$ {135;4444}	Blaster	0.3
$R_{4,0}$	-	True	ICMP flood	-	{80;135}	Welchia	0.8
$R_{4,1}$	-	True	$\neg$ (ICMP flood)	-	{80;135}	Welchia	0.67
$R_{4,2}$	-	True	ICMP flood	-	$\neg$ {80;135}	Welchia	0.53
$R_{4,3}$	-	True	$\neg$ (ICMP flood)	-	$\neg$ {80;135}	Welchia	0.4

The *Mask Table* of minor attributes shown in Table 2.9 indicates the minor attributes for all embedded rules [76]. Each row in *Mask Table* is a bit vector of attributes, where the  $i$ <sup>th</sup> bit is set to 1 representing the  $i$ <sup>th</sup> minor attribute is negated or ignored. For example, the  $M_{2,3}$  (0, 1, 1, 0, 0) means the 2<sup>nd</sup> and 3<sup>rd</sup> minor attributes in  $R_{2,3}$  are ignored.

**Table 2.9 The Mask Table of Ignored Attributes**

Mask #	A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>	A <sub>4</sub>	A <sub>5</sub>
M <sub>1,0</sub>	0	0	0	0	0
M <sub>1,1</sub>	0	0	0	1	0
M <sub>2,0</sub>	0	0	0	0	0
M <sub>2,1</sub>	0	1	0	0	0
M <sub>2,2</sub>	1	0	0	0	0
M <sub>2,3</sub>	0	1	1	0	0
M <sub>3,0</sub>	0	0	0	0	0
M <sub>3,1</sub>	0	0	0	0	1
M <sub>3,2</sub>	0	1	0	0	0
M <sub>3,3</sub>	0	0	1	0	0
M <sub>3,4</sub>	0	1	0	0	1
M <sub>4,0</sub>	0	0	0	0	0
M <sub>4,1</sub>	0	1	0	0	0
M <sub>4,2</sub>	0	0	0	0	1
M <sub>4,3</sub>	0	1	0	0	1

In Internet, each worm can be represented as a set of attribute-value pairs. We can automatically collect such attribute-value pairs and feed them into our classification *KBS* to classify them in the suitable category. Since new worms might have been derived from old discovered worms, the difference between their attribute-values seems to be slight. As mentioned above, *EMCUD* could generate lots of embedded rules with different CF values for accommodating the knowledge of the changed worms due to the property of minor attributes; e.g.,  $R_{1,1}$  “*IF (DoS type = Email flood) AND  $\neg$  (Mail\_Attachment = (sample.exe; puta!!scr)) THEN Nimda*”, a marginally acceptable embedded rule with CF = 0.4, may be fired by a new Nimda variant which is treated as a member of original Nimda class. If this rule has been fired frequently due to a specific email attached file attribute-value “*readme.exe*”(more evidence of the occurrence of the candidates of Nimda variants have been gathered), a new original rule “*IF (DoS type = Email flood) AND (Mail\_Attachment = readme.exe) THEN Nimda.B*” , a subset of extended Nimda object class namely Nimda.B, with CF = 0.8 together with an embedded rule “*IF (DoS type = Email flood) AND  $\neg$  (Mail\_Attachment = readme.exe) THEN Nimda.B*”

with  $CF = 0.5$  could be generated to single the Nimda.B class out of the extended Nimda object class.

In summary, to acquire dynamic knowledge, the experts are required to be aware of the occurrence of new objects in the interviewing approach and knowledge acquisition systems. However, it is still difficult for experts to be aware of the new object without any additional related information. The machine learning approaches which can learn the useful model according to the selected training cases also lack the ability of discovering dynamic knowledge unless more context information can be included in the training process. Although many knowledge acquisition methodologies and related tools have been proposed to improve the quality of the elicited static knowledge by domain experts with/without knowledge engineers in the past twenty years, most of them are lack of the ability of discovering dynamic knowledge unless rebuilding the knowledge base in the dynamic environment. Therefore, a new knowledge acquisition method to acquire the dynamic knowledge is required.

## Chapter 3

# Dynamic Knowledge Acquisition Based Upon

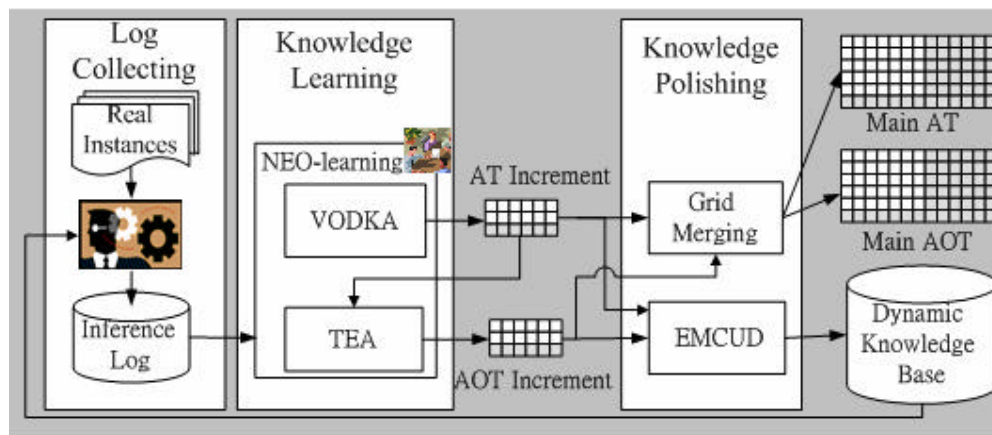
## EMCUD

Although many researchers proposed new knowledge acquisition approaches to acquire different domain knowledge, few of them discuss the acquisition of dynamic knowledge due to the changing environment as time goes on. These traditional knowledge acquisition methodologies are weak to discover dynamic knowledge due to the lack of insufficient context information to notice experts the occurrence of dynamic knowledge. New objects might be discovered using incremental learning methods with enough new cases and the experts should be able to be aware of the occurrence of these objects to acquire the knowledge of them again. The knowledge acquisition systems should be capable of representing the dynamic knowledge. Therefore, we propose *Dynamic EMCUD* combining the advantages of interviewing, machine learning and knowledge acquisition systems to collect sufficient information for assisting experts to be aware of dynamic knowledge.

### 3.1 The Concept of *Dynamic EMCUD*

As we know, generating rules in *EMCUD* would be cost inefficient if the size of Acquisition Table (AT) and Attribute Ordering Table (AOT) are too large. After collecting sufficient information of new evolved objects, *EMCUD* has to manually

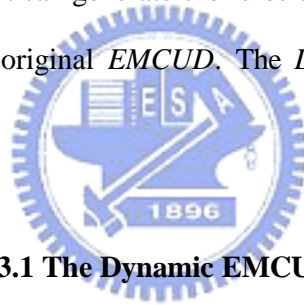
regenerate the original and embedded rules to classify these new objects with the large main AT. Therefore, the concept of *Dynamic EMCUD* shown in Figure 3.1 is proposed to help experts incrementally generate the dynamic knowledge based upon a *New Evolved Object learning (NEO-learning)* module for enhancing the explanation power of the original embedded knowledge base.



**Figure 3.1 The Concept of Dynamic EMCUD**

The dynamic knowledge base (embedded rule base) will be created according to the original main AT and AOT table using *EMCUD*. Then the inference behaviors (facts/attribute-value pairs) will be collected iteratively based upon the initially constructed knowledge base to discover the candidates of the variants during Log Collecting Stage. The NEO-learning module is proposed in *Dynamic EMCUD*, including *Variant Object Discovering Knowledge Acquisition (VODKA)* and *Trend Evolution Acquisition (TEA)* to help domain experts construct a small AT increment and an AOT increment, respectively, after confirming the occurrence of new variant objects in Knowledge Learning Stage. *VODKA* and *TEA* will be detailedly described in Chapter 4 and Chapter 5, respectively.

The ignored attribute-value pair of the minor attribute will be treated as an item and a set of ignored attribute-value pairs will be treated as a transaction to discover the association between interesting attribute-value pairs. The AT increment, which can be generated by monitoring the frequency of the weak embedded rules using *VODKA*, is used to record the new evolved objects and the attributes which are updated or added to generate the dynamic knowledge. The AOT increment is used to help experts to generate the adaptive relative importance of each attribute to each object as time goes on by tracing the importance evolving trends of all attributes during a time interval in *TEA*. Through integrating the AT increment and the AOT increment into the main AT and the main AOT respectively using Grid Merging algorithm in Knowledge Polishing Stage, it can generate the rules of new evolved objects with the grid evolution ability using original *EMCUD*. The *Dynamic EMCUD* is shown as Algorithm 3.1.



**Algorithm 3.1 The Dynamic EMCUD Algorithm**

**Input:** The original main *AT*, *AOT* and embedded rule base *RB*.  
**Output:** The rules with embedded meaning about variants.

**Stage I:** Collect all facts of the weak embedded rules as inference log of the *RB*.  
**Stage II:** Generate the new variants acquisition table *AT'*.  
    **Step 1:** Discover large itemsets *L* using the inference log.  
    **Step 2:** Generate *AT'* using *L* and additional attributes provided by experts.  
    **Step 3:** Update the *AOT'* according to *AT'*.  
**Stage III:** Use *EMCUD* to generate rules of new variants.  
    **Step 1:** Generate rules according to *AT'* and *AOT'*.  
    **Step 2:** Merge *AT'* into original main acquisition table *AT*.  
    **Step 3:** Merge *AOT'* into original main *AOT*.

### 3.2 Inference Log Collecting Based upon Meta Rule

Without loss of generality, assume there are  $k$  attributes to classify  $m$  objects in the main acquisition table. Thus, the total number of the embedded rules used in

*Dynamic EMCUD* is limited. In order to assist domain experts in noticing and analyzing the occurrence of the candidates of variant objects, the following four meta rules are used in *Dynamic EMCUD* to collect the frequent inference log (fact/ raw data) of weak embedded rules to help experts notice the occurrence of new objects.

**$MR_1$ : IF  $R_{i,j}$  is fired THEN Increase  $C_{i,j}$  by one.**  
 **$MR_2$ : IF  $CF(R_{i,j}) \leq TH_{CF}$ , THEN Log  $R_{i,j}$ .**  
 **$MR_3$ : IF  $C_{i,j} \geq TH_{cnt}$  AND  $CF(R_{i,j}) \leq TH_{CF}$  THEN Run **VODKA Algorithm** to acquire the variants acquisition table increment AND Reset *TimeOut*.**  
 **$MR_4$ : IF  $TimeOut = TH_{period}$  THEN Run **VODKA Algorithm** AND Reset *TimeOut*.**

The meta rule  $MR_1$  is used to count the fired frequency of each embedded rule ( $C_{i,j}$ ). The meta rule  $MR_2$  means that all facts (attribute-value pairs) of the embedded rules with marginally acceptable CF lower than strong CF bound threshold ( $TH_{CF}$ ) are logged as a record, ( $R_{i,j}, A_1, A_2, \dots, A_k, CF(R_{i,j})$ ). The meta rule  $MR_3$  means that if there exists one weak embedded rule with fired frequency exceeding the warning line threshold ( $TH_{CNT}$ ), new variants may be discovered iteratively using *VODKA*. The meta rule  $MR_4$  means that *VODKA* will be executed periodically to refresh the new variants acquisition table. The *TimeOut* will be reset when  $MR_3$  or  $MR_4$  is triggered.

### 3.3 The NEO-Learning Module

As we know, the *KBS* is proposed to help experts solve the difficult problems in a specific domain based upon the pre-constructed static knowledge base. However, the new objects will be developed or discovered as times goes on and might result in the inefficiency of *KBS*. Based upon the embedded rules generated by *EMCUD*, some new evolved objects may be classified into well-known object class by the weak embedded rule with weak CF which is not strongly suggested by experts. Through monitoring the frequency of these weak embedded rules, the candidates of new evolutionary objects might be discovered to notice the experts. Therefore, the

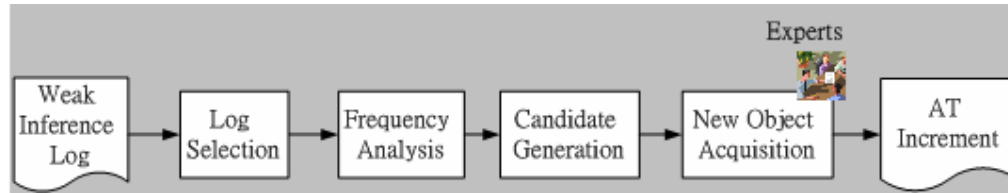


characteristics of these candidates of new objects could be extracted from these collected inference logs. The evidence of the new objects can be confirmed by experts and some attributes could be modified and added when the dynamic knowledge is needed to be singled out. Moreover, the relationships between these inference logs might be represented as the significance of each attribute to each new object. Hence, analyzing the evolving trends of all attribute should be useful in capturing the realistic significance of the attribute to the object.

The NEO-learning module can help experts analyze the interesting inference logs of weak embedded rules to learn the evidence of new evolved objects using the *VODKA* to notice experts the occurrence of the new objects. Based upon the confirmed new objects, the relationships of all attributes of each object are analyzed to set the significance of the attribute with the times using *TEA* to help experts decide the CF values of the embedded rules of new objects, which can be generated using *EMCUD* according to the discovered objects stored in an AT increment and an AOT increment. Finally, the AT increment and the AOT increment will be integrated with the main AT and the main AOT, respectively.

### **3.3.1 Frequent Events Analysis**

*EMCUD* lacks the ability of grid evolution for singling the new evolved objects out of well-known objects since experts may be unaware of the occurrence of the new evolved objects without sufficient information. Hence, we propose *VODKA* to monitor the frequent behaviors of interesting inference logs of the weak embedded rules with the lower CF values for helping experts notice the occurrence of the new objects.



**Figure 3.2 The Flow of VODKA**

The novelty of the *VODKA* shown in Figure 3.2 is to collect the inference logs of weak embedded rules from each *KBS* to learn the candidates of new evolved objects for experts to make a confirmation. The minor attribute-value pairs between inference logs of weak embedded rules are useful to help experts discover new knowledge and determine whether new object is evolved based upon fired frequency. For each object, if its inference logs of weak embedded rules are frequent, the frequent minor attribute-value pairs could be treated as candidates of new evolved objects. Furthermore, new attributes or attribute-values of the new object could be defined and used to generate a small AT increment. Hence, these candidates will be used to help experts single the new objects out of the extended object class using the new object acquisition module based upon the AT increment.

Therefore, if the new objects are confirmed by experts, the related ambiguous attributes (minor attributes), which might result in the marginally acceptable CF values of weak embedded rules, could be refined or new attributes could be added to improve the classification ability. If the initial data type of a minor attribute is too rough to describe the object, a superior data type is recommended and the values of the attribute in both original object and new evolved object should be modified.

For example, the *BOOLEAN* data type may be refined to *SINGLE VALUE* data

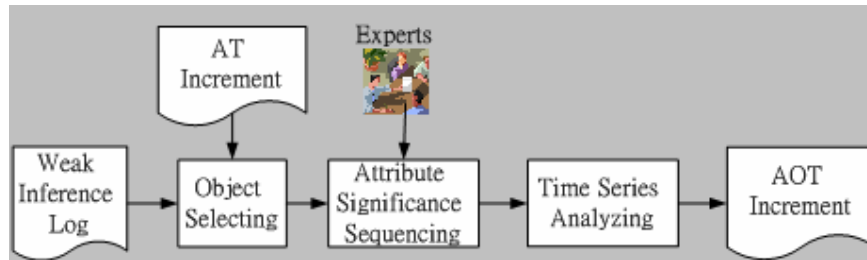
type (Hwang and Tseng, 1990). If changing the data type still can not discriminate the new variants from original objects, acquiring new attributes from domain experts will be suggested in the new objects acquisition module. According to the complexity of relations between objects and attributes or even the relations between different tables, it is hard for experts to cooperate with each other in building every column and every row for each table. Finally, the result of new objects and corresponding attributes can be used to construct the AT increment.

### 3.3.2 Trend Evolution Analysis

Although the original idea of constructing AOT makes *EMCUD* more adaptive to elicit embedded meanings, the relative importance of all attributes to each object could be adjusted since the dynamic knowledge may change or evolve with the times. It means that some embedded rules, which are recommended by experts now, may become uncertain in the near future. Each object in the AOT is decomposed to record the relative importance of each attribute to the object with the times. Since the traditional Repertory Grid-based KA methods do not record the evolved trend of each new object and the *EMCUD* is difficult in deciding the ordering of all attributes of the object by experts, the *TEA*, which can discover the evolution of the relative importance of each attribute to each object with the times, is proposed to help experts monitor the significant importance changing of all attributes to each object in a time interval.

As shown in Figure 3.3, the object can be singled out of the old object according to the viewpoints of experts or the learning results of the frequency events analysis. Each attribute can be simply assigned as “0” or “1” in each time point for indicating

whether it is important to each object or not, where “0” represents the attribute is considered as the unimportant attribute to the object and “1” represents the attribute is important to the object. The domain expert can then decide which attributes are required to be traced with the times if some ordering values of the attributes are hard to be decided immediately.



**Figure 3.3 The Flow of TEA**

The “0” or “1” is called an attribute event  $e_t$  of each object in a time point  $t$ , and the attribute event sequence of “0” and “1” is recorded in a table to capture the evolved behavior of each object. Hence, the AOT increment can be generated for evolving the relative importance of each attribute to each object (ordering values) according to the sequence of “0” and “1” events with the times using a time series analysis approach. Since the “1” means an attribute is important to an object, the consecutive “1” recorded in consecutive time points indicates that relative importance of the object should become higher. On the contrary, the consecutive “0” indicates that the relative importance of the object should be lower. Hence, a simplified time series analysis is proposed to capture the trend meaning and incrementally adjust the CF value of each rule. Let the initial value of each signal sequence be the original AOT value of the attribute to the object.

### 3.4 Grid Merging

In order to maintain the new discovered new object, we propose grid merging algorithm shown in Algorithm 3.2 to integrate the AT increment and AOT increment into the main AT and the main AOT, respectively. Therefore, the small AT and the small AOT instead of the whole large main AT and the main AOT are used to update the embedded rule base using *EMCUD*.

#### Algorithm 3.2 The Grid Merging Algorithm

**Input:** The main *AT*, main *AOT*, AT increment *AT'*, and AOT increment *AOT'*.  
**Output:** The updated main *AT* and main *AOT*

**Step1:** Integrate the AT increment *AT'* into the main *AT*.  
**Step1.1:** Append each new object and each new attribute in the *AT'* as a new column and row in the main *AT*, respectively.  
**Step1.2:** Ask experts to fill the values of the modified attributes of other objects in the main *AT* if necessary.  
**Step1.3:** Ask experts to examine the values of the new attributes of other objects in the main *AT* if necessary.

**Step2:** Integrate the AOT increment *AOT'* into the main *AOT*.  
**Step2.1:** Expand the size of the main *AOT* according to the main *AT* updated in the **Step1**.  
**Step2.2:** Fill the corresponding *AOT* values according to the *AOT'*.  
**Step2.3:** Refine the values of all old attributes to each old object in the main *AOT* using the *Trend Evolution Acquisition* if necessary.

**Step3:** Reset the AT increment *AT'* and the *AOT* increment *AOT'*.

To merge the AT increment into the main AT, each new evolved object should be appended as a new column in the main AT and each new added attribute should be appended as a new row in **Step 1.1**. In order to maintain the correctness of the main AT, the values of all modified or new added attributes to each object should be acquired by experts if necessary. Since the size of AOT need equal the size of AT, the size of the main AOT should be expanded in **Step 2.1** according to the main AT updated in **Step 1**. Besides the value of all attributes to each new object in AOT increment, the other values of all old attributes to each old object could also be

learned using the trend evolution analysis to obtain the relative importance at time  $t$ .

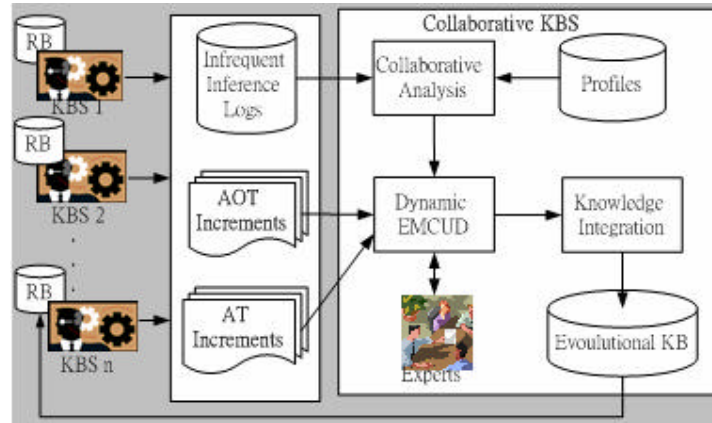
### **3.5 Collaborative Framework of *Dynamic EMCUD***

Although the *VODKA* and *TEA* in *Dynamic EMCUD* can be used to single the new objects out of extended object class and to generate their corresponding rules with adjusted CF values, the CF value of embedded rules of these objects might be inconsistency since they might be in different environments. Therefore, a collaborative knowledge acquisition framework, which is consisting of several local *KBSs* and a collaborative *KBS*, based upon *Dynamic EMCUD* is required to integrate the knowledge discovered in every local *KBS*. Moreover, some new evolved objects which may occur infrequently in each local *KBS* (but may be frequent in the collaborative *KBS*) can not be found. Hence, how to collect sufficient context information to notify experts of the occurrence of the dynamic knowledge is an important issue. Therefore, the collaborative knowledge acquisition framework can collect the relevant information as time goes on and help experts discover these new objects based upon sufficient context.

#### **3.5.1 The Framework of Collaborative *Dynamic EMCUD***

In a dynamic environment, a collaborative *Dynamic EMCUD* framework shown in Figure 3.4 is proposed to analyze the correlations of interesting inference logs of embedded rules between multiple local *KBSs* in a dynamic environment to discover the new evolved objects. Each *KBS* can monitor the frequent inference behaviors of weak embedded rules to construct an AT increment and analyze the significant change of the importance to evolved objects to construct an AOT increment for adjusting the relative importance of each attribute to each object with the times. Several heuristics

are proposed to help experts adjust the CF values of the discovered knowledge of the new evolved objects from the collection of inference logs.



**Figure 3.4 The Framework of Collaborative Knowledge Acquisition**

As we know, the expert system and *KBS* are usually designed for solving the difficult problems in a specific domain. In Figure 3.4, each *KBS* may have different configurations to represent such kind of expert systems or *KBS*s with an embedded rule bases. When the system is operating, some inference logs of cases including old and new will be recorded. By analyzing the relationships between these inference logs using the collaborative heuristics with sufficient context information, the occurrence of candidates of dynamic knowledge could be discovered. The collaborative framework consists of log collector, *Dynamic EMCUD*, collaborative analysis, knowledge integration, and Profiles to learn the knowledge of dynamic behaviors and to record the configurations of different environment.

*Dynamic EMCUD* is deployed in each local *KBS* and collaborative *KBS* to discover significant variant knowledge by monitoring the inference behaviors of weak embedded rules. Moreover, the sequence of inference log will be considered to

discover the relations between similar embedded rules to monitor the occurrence of evolutionary knowledge as time goes on. All discovered information will be collected and further analysis using collaborative analysis by considering the static profiles and dynamic behaviors stored in Profiles, since some insignificant variant or evolutionary knowledge might evade the frequency-based and time-based analysis. Finally, the *Dynamic EMCUD* will incrementally integrate the discovered evolutionary knowledge, which is confirmed by experts, into the evolutionary knowledge base.

Some new evolved objects may occur in some *KBSs* with similar profiles, e.g., the SQL server running on Windows operation system, the correlations between inference logs and profiles might be useful for helping experts discover them. Finally, for the discovered object, the CF value of the new embedded rule should be recalculated. Hence, a CF adjusting method is proposed to combine the knowledge of new objects discovered in each *KBS* and the collaborative *KBS*.

### **3.5.2 The Category of Context Information**

Since substantive knowledge can evolve within the dynamic environment as time goes on, how to acquire and represent the dynamic context information becomes an important issue. The context information can be classified into two categories: static profiles and dynamic behaviors information.

#### **(1) Static Profile:**

In the real world, the environment includes individuals, the relationships between individuals, and the related configurations. The environment could be considered as a collection of network properties and each individual has its own properties in the



environment. Therefore, the static profile can be considered as environment configuration and individual configuration. The environment configurations describe the environment, members in the environment, the status of the environment, and other relative properties rely on the selected domain. The individual configurations describe the individual ID, Location, Role of individual, and other relative properties depending on the domain. Through the static profile, we could classify the knowledge occurred in similar configuration.

## **(2) Dynamic Behaviors:**

Some knowledge will evolve to adapt the dynamic environment due to the natural of knowledge evolution. By clearly representing the behaviors of individual and environment, the trend of individual and environment can be easily acquired. The individual trend consists of the sequence of status of each time period and the occurrence of events pair. Also, the other relative properties should be also considered in each domain. Like the individual trend, the evolutionary trend is also combining the sequence of environment statuses and other properties to analyze the trend of environment for capturing evolutionary knowledge.

As we know, XML is a standard language that is understandable. We design an XML based language that facilitates the machine readability for the collaborative KA framework to model the context information. In this model, not only the static profile but also the dynamic behaviors can be modeled using XML based description because the structure of XML is regular expression. Furthermore, the stored context can be easily reused, and the representation can be extended to describe new added properties of individual or environmental profile and behaviors due to the

standardized property of XML.

### 3.5.3 The Collaborative Heuristics

Since some invisible or unrecognizable behaviors might be ignored without sufficient information, collaborative multiple *KBSs* to collect more evidence of evolutionary knowledge becomes more important. The static profile and dynamic behaviors of individual and environment could be used to help discover evolutionary knowledge since they could assist experts to trace of changing behaviors. The static profile can be used to analyze which kinds of behaviors could occur in the different profiles since some behaviors might exist in similar environment. The dynamic behavior can be used to analyze the similar behaviors in different environment as time goes on. Based upon the collection of the sufficient context information including the unrecognizable or invisible behaviors in single sensor to discover the evolutionary knowledge, the static profile and dynamic behaviors of individual and environment could be used to help discover evolutionary knowledge. Four collaborative heuristics for discovering dynamic knowledge and three collaborative weighting heuristics for collecting sufficient evidence from multiple sensors are proposed to analyze the relationship between them.

#### **Dynamic Knowledge Heuristics:**

- (1) Environment-Insensitive Heuristic: similar behaviors results to similar symptoms in different profiles. This is useful to consider the frequency context information to discover variant knowledge.
- (2) Service-Sensitive Heuristic: similar behaviors result to different symptoms due to different profiles. This is used to analyze the relationships between collected

events by comparing with the corresponding profiles.

- (3) **Symptom-Sensitive Heuristic:** different behaviors result to similar symptoms due to polymorphic (similar) profiles due to the similar living conditions is existence. These similar behaviors should be considered the same to obtain more evidence to discover the occurrence of evolutionary knowledge.
- (4) **Time-Sensitive Heuristic:** different behaviors result to different symptoms in polymorphic (similar) profiles due to the evolution of behaviors in the changing environment. This is useful to trace the behaviors for discovering the evolutionary knowledge.

#### **Collaborative Weighting Heuristics:**

- (1) **Half-life-sensitive Heuristic:** Adventitious behaviors disappear in a long time and the significant of the behaviors should be degraded for adapting knowledge base. This is useful to avoid the interference with adventitious events in a time period for discovering evolutionary knowledge.
- (2) **Location-Sensitive Heuristic:** behaviors occurred in different sensors should be considered as different evidence, since different sensor may play different role in different location. The importance need to be considered to adjust the discovered evidence.
- (3) **Attribute-Sensitive Heuristic:** different attributes evolution should be considered as different evidence since the relative importance of each attribute to each object is various. An attribute might be the key factor of evolutionary knowledge and hence needs to be considered as the most important evidence.

For example, the SQL Slammer uses UDP port 1434 to exploit a buffer overflow

in a MS SQL server to simply switch off this port of the victim host. The collaborative *KBS* can learn this knowledge based upon the infrequent logs reported from some local *KBS*s according to the same service stored in profile. Some new objects may occur in similar environment.

These heuristics will be applied in the NEO-learning module to assist experts in being aware of the occurrence of new evolved objects.

### **3.6 Implementation of Dynamic EMCUD**

*Dynamic EMCUD* is implemented by *DRAMA* [46], a new object-oriented rule base system platform implemented using pure Java language, to refine the embedded rule base by observing the behaviors of weak embedded rules. It includes *DRAMA* Server, Console, Knowledge Extractor, and Rule Editor. Also, it provides Application Programming Interface (API) to access *DRAMA* server in *DRAMA* integrated systems. There are four basic relations between knowledge concepts defined in *DRAMA*: Reference, Extension-of, Trigger and Acquire. The Reference relation represents the association of two different knowledge classes (KCs) if the KCs have common piece of knowledge, which is useful for using original knowledge to construct new knowledge. Extension-of relation is used to extend or modify the KC constructed by other people, which is useful for knowledge sharing and exchanging. The Trigger and Acquire relations are used to represent the interaction of different KCs. The Log Collecting Stage is encoded by meta-rules according to the specific domain in *DRAMA*; the *VODKA*, *TEA*, *Grid Merging* and *EMCUD* are implemented using the JSP to make a communication channel using the *API* provided by *DRAMA*. We implemented a worm detection prototype system to evaluate the performance of

*Dynamic EMCUD* to incrementally integrate evolved knowledge into knowledge base in Chapter 6.

In this chapter, the *Dynamic EMCUD* based upon Repertory Grid is proposed to elicit the embedded meanings of knowledge. *Dynamic EMCUD* can generate an AT increment and an AOT increment to represent the evolved objects and to record the relative importance of each attribute to each object for capturing the embedded meanings with acceptable CF value by relaxing or ignoring some minor attributes. *Dynamic EMCUD* can monitor the frequent inference behaviors of weak embedded rules to construct an AT increment for classifying variant objects and analyze the significant change of the importance to evolved objects to construct an AOT increment for evolutionary objects by adjusting the relative importance of each attribute to each object with the times. Moreover, a collaborative knowledge acquisition framework will be proposed to analyze the correlations of interesting inference logs of embedded rules between multiple *KBSs* with *Dynamic EMCUD* in a dynamic environment to discover the new evolved objects.

## Chapter 4

### Variant Knowledge Acquisition

The *Dynamic EMCUD* based upon Repertory Grid is proposed in Chapter 3 to elicit the embedded meanings of knowledge by the generation of the AT increment of new evolved objects using *VODKA* by relaxing or ignoring some minor attributes. The details of *VODKA* are given as follows.

#### 4.1 Idea

Although *EMCUD* and other similar approaches could be manually rerun to acquire variant knowledge from domain experts to classify new variant objects, it might be costly and hard to obtain the knowledge due to the insufficient information about variants. As mentioned above, assume some objects in  $O_l$  class belong to the original object class ( $OO_l$ ) of  $O_l$ , which can be classified by original rules of  $O_l$ . The other objects in  $O_l$  class classified by embedded rules of  $O_l$  belong to the extended object class ( $EO_l$ ) of  $O_l$ , where  $OO_l \dot{\cup} EO_l$ . In the  $EO_l$ , some evolved objects can be classified by the embedded rules of  $O_l$  with weak CF values, which are singled out to be a variant object class ( $VO_l$ ) of  $O_l$  with the significant attributes emerged from minor attributes. That is,  $VO_l \dot{\subset} EO_l$  and  $VO_l \cap OO_l = \emptyset$ , where  $l \in i \in m$ . Because the embedded rules with diverse CF values represent different supports to classify objects, the ones with marginally acceptable CF might be triggered by some candidate of a new variant. Therefore, our idea is to analyze the inference behaviors of weak

embedded rules (the weak suggestion by experts) to construct the new variants acquisition table for extracting new variant knowledge.

## 4.2 Variant Objects Discovering Knowledge Acquisition (VODKA)

After the interested inference logs are collected, the acquisition table increment of new objects will be generated through interacting with domain experts based upon the observation of inference log. An ignored attribute-value pair, i.e. (*DoS type = TCP flood*), is treated as an item and the transaction is represented as a set of ignored attribute-value pairs, i.e.,  $\{(DoS\ type = TCP\ flood), (TCP\ port = \{135;4444\})\}$ . The inference log could be automatically transformed into the transaction database (*D*) and the item set (*I*) using the *Mask Table* of ignored attributes. In order to obtain the candidates of new variants, we apply Apriori algorithm [1][2] to discover the large itemsets (*L*) for providing more useful information.

After generating the large itemsets, new variants acquisition table might be elicited based upon the new variants acquiring algorithm. The new objects using unclear attributes would be singled out accordingly, if the experts reconfirm the addition of the new variant object. Thus, one of three recommendations including adding a new attribute-value of a minor attribute, modifying the data type of a minor attribute, adding a new attribute, will be further given to adjust the main acquisition table. If a new evolved variant object is singled out, the new attribute-value of the minor attribute could be added to represent the characteristic of new objects. If the initial data type of certain attribute is too rough to describe the object, a superior data type is recommended and the values of the attribute in both original object and variant should be modified.

For example, the BOOLEAN data type may be refined to SINGLE VALUE data type [34]. If changing the data type still can not discriminate the new variants from original objects, acquiring a new attribute from domain experts will be suggested in *VODKA*. Thus, the new variants acquisition table will be created iteratively using the discovered large itemsets of *VODKA* shown in Algorithm 4.1. However, adding a new attribute, which is very time consuming to create new row in new variant acquisition table, is the last choice for classifying variant objects.

#### Algorithm 4.1 *VODKA* Algorithm

**Input:** Inference log and the main acquisition table  $T$ , the minimal support  $d$ .  
**Output:** The new variant object class  $VO$ , new attribute set  $AN$ , and new variants acquisition table  $T'$ .

**Step 1:** Transform inference log into the transaction data set  $D$ .  
**Step 2:** Discover large itemsets  $L$  by  $d$  using  $D$ .  
**Step 3:** For each large itemset, ask experts to determine whether it belongs to new variant or not.  
**Step 4:** If new variant is confirmed, ask experts to acquire the related information about new variant.  
 Store  $VO_{new}$  in  $VO$ .  
 Add a new column to represent the new variant  $VO_{new}$  in  $T'$ .  
 Ask experts to confirm whether changing the data type of the attribute  $A_i$  is needed or not, where  $l \in i \in k$ .  
**Step 4.1:** If no data type needs to be changed, Suggest the **Recommendation I**.  
 Add a new attribute-value of  $A_i$  of the  $VO_{new}$ .  
 Else, ask experts to confirm whether adding a new attribute is needed or not.  
**Step 4.2:** If no new attribute needs to be added, Suggest the **Recommendation II** to modify the data type of  $A_i$ .  
 Ask experts to acquire the mapping function of values between original and new data types.  
 Add a column in  $T'$  to represent the original object with new mapping values.  
 Else, **Suggest the Recommendation III**.  
 Ask experts to acquire the values of the new attribute  $A_{new}$ .  
 Add a new row in  $T'$  to represent the new variant  $A_{new}$ .  
 Store  $A_{new}$  in  $AN$ .  
 Add a column in  $T'$  to represent the original object with new attribute-value if needed.  
**Step 5:** Return  $VO, AN, T'$ .



Besides interaction with domain experts, the computational cost of the algorithm is dependent on the **Step 2**. The size of collecting inference log and minimal support threshold setting will affect the computational cost. For different type of inference log, different learning methods can be selected to learn and discover the candidate behaviors of variants.

#### Example 4.1 The Variant Learning Example of a Blaster Worm

In this example, assume the fired sequence of some embedded rules of Blaster worms with marginally acceptable CF values are given in Table 4.1.

**Table 4.1 The Partial Inference Logs of Blaster**

Rule #	A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>	A <sub>4</sub>	A <sub>5</sub>	Object	CF
R <sub>3,4</sub>	17	False	Windows Update flood	-	X	Blaster	0.3
R <sub>3,2</sub>	100	False	Windows Update flood	-	{135}	Blaster	0.43
R <sub>3,4</sub>	8	False	Windows Update flood	-	X	Blaster	0.3
R <sub>3,2</sub>	119	False	Windows Update flood	-	{4444}	Blaster	0.43
R <sub>3,4</sub>	17	False	Windows Update flood	-	X	Blaster	0.3
R <sub>3,2</sub>	100	False	Windows Update flood	-	{135}	Blaster	0.43
R <sub>3,4</sub>	11	False	Windows Update flood	-	X	Blaster	0.3
R <sub>3,2</sub>	76	False	Windows Update flood	-	{4444}	Blaster	0.43
R <sub>3,4</sub>	66	False	Windows Update flood	-	X	Blaster	0.3
R <sub>3,2</sub>	100	False	Windows Update flood	-	{135}	Blaster	0.43

Assume minimal support is set to 30%; the large itemsets  $L$  including  $L_1 = \{(A_2 = False); (A_5 = X)\}$  and  $L_2 = \{(A_2 = False, A_5 = X)\}$  will be provided to experts for further recommendation; i.e.,  $L$  will be used to generate the acquisition table

increment  $AT'$  according to the recommendations suggested by *VODKA*.

**VODKA:** Does the attribute-value pair ( $A_2 = False$ ) belong to any new variant object?

**EXPERT: Yes.** /\* It means that a new variant contains the selected attribute-value pair ( $A_2 = False$ ). Otherwise, the large itemset is discarded and another large itemset is chosen to examine. \*/

**VODKA:** What is the name of the new variant object?

**EXPERT:**  $VO_{new}$ .

A new column will be added in  $AT'$  to represent the variant,  $VO_{new}$ , separated from the original object.

**VODKA:** Is the data type of  $A_2$  required to be changed?

**EXPERT:** No. /\* It means the data type no need to change after adding new variant (**Recommendation I**). Otherwise, *VODKA* will ask the following question. \*/

The row representing  $A_2$  with new attribute-value and one column representing the variant object will be created in  $AT'$ .

**VODKA:** Is any new attribute required to be added?

**EXPERT:** No. /\* It means no need to add new attribute and **Recommendation II** is then suggested. Otherwise, *VODKA* will suggest **Recommendation III**. \*/

### **Recommendation II:**

**VODKA:** What is the new name and new value set of the attribute  $A_2$ ?

**EXPERT:**  $N_{DT_{new}}, V_{DT_{new}}$ . /\* For each old value in  $A_2$ , *VODKA* will ask experts to define the mapping between old and new value sets. \*/

The row representing  $A_2$  with new mapping values and two columns representing the original and variant objects will be created in  $AT'$ .

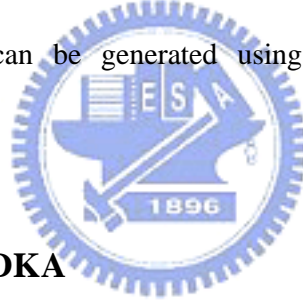
### Recommendation III:

**VODKA:** What is the name and value set of the new attribute-value pair?

**EXPERT:**  $A_{new}, AV_{new}$ . /\* *VODKA* will ask experts to provide a set of values ( $AV_{new}$ ) of the new attribute  $A_{new}$ . \*/

A new row representing the useful attribute namely  $A_{new}$  with a set of value ( $AV_{new}$ ) and two columns representing original object and new variant will be added in  $AT'$ .

If all large itemsets are confirmed, the acquisition table increment  $AT'$  can be integrated into the main acquisition table and the corresponding embedded rules of discovered variant object can be generated using *EMCUD* in the Knowledge Polishing Stage.



### 4.3 The Analysis of VODKA

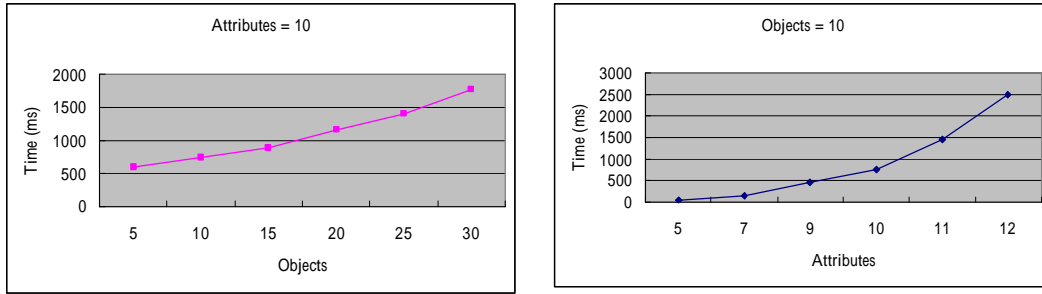
The cost of running *VODKA* can be divided into two categories: computational cost and interaction cost. Assume there are  $k$  attributes to classify  $m$  objects in the original main acquisition table, where the grid size is  $k*m$ . For simplifying our discussion, we use  $ER_{k,m}$  to represent the total number of embedded rules in the classification *KBS*, where  $ER_{k,m} < m*2^k$ .

In Log Collection Stage, assume  $n$  instances are matched by the classification *KBS*. For each instance, it has  $P_e$  probability to be classified by weak embedded rules; hence the size of interesting inference log database in *VODKA* is  $n * P_e$ .

In Knowledge Learning Stage, the computational cost of *VODKA* is dominated by the learning algorithm we selected. In this dissertation, Apriori algorithm is used to learn the candidates of variant worms. Hence, the computational cost is  $O(Apriori)$ . For example, if the size of database has  $n$  transactions (each transaction has  $k$  attributes) and the maximal length of large itemsets is  $len$ , then the time complexity of traditional Apriori algorithm is  $O(n*k*len)$ .

Assume  $L$  large itemsets are discovered and used to notify experts to determine the existence of the variants. For each embedded rule, assume it has  $P_v$  probability to evolve a variant; hence  $P_v*ER_{k,m}$ , denoted  $V$ , variants might be discovered. Therefore, the order of interaction with experts is  $V$ , where  $V < L$ .

In Knowledge Polishing Stage, the *Grid Merging* integrates new acquisition table into original acquisition table. The computational cost of *Grid Merging* for generating rules is dependent on the size of acquisition table increment (*GRID*), denoted  $O(GRID)$ . For example, using our *Dynamic EMCUD* to generate embedded rules, it costs 0.05 ms ~ 0.15 ms to generate one rule. Figure 4.1 shows that the time of generating rules using different grid size. The computational time is approximately linear growing when setting a fixed attribute number with different number of objects in Figure 4.1 (a), and the growth rate is exponential when setting a fixed object number with different number of attributes in Figure 4.1 (b).



(a) Various number of objects

(b) Various number of attributes

**Figure 4.1 The Time of Generating Rules Using Different Grid Size**

In short conclusion, the cost of *VODKA* consists two parts: computational cost and interaction cost. The size of interesting inference log database:  $n * P_e$ .

- Computational cost:  $O(Apriori) + O(GRID)$ .
- Interaction cost:  $V$ .



#### 4.4 Experiments

With the rapid development of network technology, the network security becomes one of the most important issues today. To prevent network environment from intrusions, lots of researches and different systems are proposed to detect, filter, or prevent intrusions properly. In recent years, computer worms are grown dramatically to influence the wild computer networks due to the property of easily modifying the source code of original computer worms to create new variant for escaping the detection of related systems, e.g., Symantec Norton [72], Network Viruswall [74], etc. The detailed description of computer worms will be shown in Chapter 6. The case study of computer worms is used to evaluate the performance of *VODKA*.

In our worm detection prototype system, the knowledge of computer worms can be divided into several KCs, including the service provided by host may be infected by certain worms and then produced some symptoms in host or network. The related attributes of various computer worms can be collected by some probe tools and used to evaluate the ability of *VODKA*, which deployed in the prototype system. The details will be described in Chapter 7.

The following example shows the variant objects in this domain can be discovered by *VODKA*, where  $TH_{CNT}$  is set to 4,  $TH_{CF}$  is set to 0.7, and the minimal support is set to 30%.

**Recommendation I: Elicitation new variants by adding a new attribute-value of a minor attribute**

Nimda worm, a famous Email flooding worm, can be propagated to victims through the attached files in email. By monitoring the attached filename in email, we can discover the large itemset  $L = (A_4 = readme.exe)$  shown in Table 4.2 according to the embedded rule  $R_{1,1}$  in Table 2.6.

**Table 4.2 The Partial Inference Logs of Nimda**

Rule #	A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>	A <sub>4</sub>	A <sub>5</sub>	Object	CF
R <sub>1,1</sub>	6	True	Email flood	readme.exe	{137}	Nimda	0.4
R <sub>1,1</sub>	300	False	Email flood	sample1.exe	{25}	Nimda	0.4
R <sub>1,1</sub>	17	True	Email flood	readme.exe	{137}	Nimda	0.4
R <sub>1,1</sub>	14	False	Email flood	readme.exe	{25}	Nimda	0.4
R <sub>1,1</sub>	4	False	Email flood	readme.exe	{445}	Nimda	0.4
R <sub>1,1</sub>	19	False	Email flood	readme.exe	{80}	Nimda	0.4
R <sub>1,1</sub>	44	False	Email flood	hash.exe	{25}	Nimda	0.4
R <sub>1,1</sub>	38	True	Email flood	readme.exe	{138}	Nimda	0.4
R <sub>1,1</sub>	300	False	Email flood	inter.exe	{25}	Nimda	0.4
R <sub>1,1</sub>	28	False	Email flood	readme.exe	{25}	Nimda	0.4

Based upon the large itemsets, *VODKA* will ask the following questions.

**VODKA:** Does the attribute-value pair ( $A_4 = readme.exe$ ) belong to any new variant object?

**EXPERT:** Yes.

**VODKA:** What is the name of the new variant object?

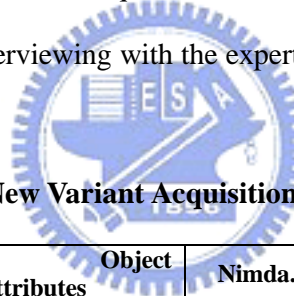
**EXPERT:** Nimda.B.

**VODKA:** Is the data type of  $A_4$  required to be changed?

**EXPERT:** No.

Consequently, the new variant acquisition table of Nimda.B shown in Table 4.3 will be generated through interviewing with the experts in this iteration.

**Table 4.3 The New Variant Acquisition Table of Nimda.B**



Attributes	Object	Nimda.B
Threads		X
System Reboot		X
DoS Type		Email flood
Mail Attachment		{readme.exe}
TCP Port		X

Hence, an original rule “*IF (DoS type = Email flood) AND (Mail Attachment = (readme.exe)) THEN Nimda.B, CF=0.8*” and an embedded rule “*IF (DoS type = Email flood) AND  $\neg$  (Mail Attachment = (readme.exe)) THEN Nimda.B, CF=0.5*” of the Nimda.B will be generated using *EMCUD* based upon the Nimda.B acquisition table.

**Recommendation II: Elicitation new variants by changing the data types of attributes**

In the priori generation of CodeRed worm, generating numerous threads to attack the victim through launching TCP flooding is one of the famous characteristics. Hence, it is useful to detect the CodeRed by analyzing the generated anomaly threads in the protected system. For the partial inference logs of CodeRed shown in Table 4.4, the following shows the values in Boolean data type will be logged as the integer value of the attribute  $A_1$  instead of True/False value.

**Table 4.4 The Partial Inference Logs of CodeRed**

Rule #	A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>	A <sub>4</sub>	A <sub>5</sub>	Object	CF
R <sub>2,2</sub>	150	True	TCP flood	-	{80}	CodeRed	0.4
R <sub>2,2</sub>	600	True	TCP flood	-	{80}	CodeRed	0.4
R <sub>2,1</sub>	300	False	TCP flood	-	{80}	CodeRed	0.6
R <sub>2,2</sub>	600	True	TCP flood	-	{80}	CodeRed	0.4
R <sub>2,2</sub>	150	True	TCP flood	-	{80}	CodeRed	0.4
R <sub>2,1</sub>	300	False	TCP flood	-	{80}	CodeRed	0.6
R <sub>2,2</sub>	600	True	TCP flood	-	{80}	CodeRed	0.4
R <sub>2,2</sub>	600	True	TCP flood	-	{80}	CodeRed	0.4
R <sub>2,2</sub>	600	True	TCP flood	-	{80}	CodeRed	0.4
R <sub>2,2</sub>	300	True	TCP flood	-	{80}	CodeRed	0.4

Using above inference log, the large itemset  $L = (A_1 = 600)$  can be discovered based upon the embedded rule  $R_{2,2}$  since the minimal support 30% is satisfied. Then, *VODKA* will ask the following questions.

**VODKA:** Does the attribute-value pair  $(A_1 = 600)$  belong to any new variant object?

**EXPERT:** Yes.

**VODKA:** What is the name of the new variant object?

**EXPERT:** CodeRed.II.

**VODIKA:** Is the data type of  $A_1$  required to be changed?



**EXPERT:** Yes.

**VODKA:** Is any new attribute required to be added?

**EXPERT:** No.

**VODKA:** Can the *Single Value* data type be used to change the original *Boolean* data type of  $A_1$ ?

**EXPERT:** Yes.

**VODKA:** What is the new name and new value set of the attribute  $A_1$ ?

**EXPERT:** Threads, (100, infinite).

**VODKA:** What is the new value of the original True value of the attribute  $A_1$  in CodeRed?

**EXPERT:** 100.

Therefore, the new variant acquisition table of CodeRed.II shown in Table 4.5 will be generated.



**Table 4.5 The New Variant Acquisition Table of CodeRed.II**

Attributes	Objects	CodeRed	CodeRed.II
Threads		300	600
System Reboot		True	True
DoS Type		TCP flood	TCP flood
Mail Attachment		X	X
TCP Port		X	X

Consequently, an original rule “*IF (Threads = 600) AND (System reboot = True) AND (DoS type = TCP flood) THEN CodeRed.II, CF 0.9*” and an embedded rule “*IF  $\neg$  (Threads = 600) AND (System reboot = True) AND (DoS type = TCP flood) THEN CodeRed.II, CF=0.3*” will be generated to classify the CodeRed.II.

### Recommendation III: Elicitation new object through adding new attributes

As mentioned in Example 4.1, we can obtain the large itemsets  $L = \{(A_2 = False); (A_5 = X); (A_2 = False, A_5 = X)\}$ , which will be used to elicit the embedded rules of the new variant. The symbol “X” means no attribute-value of  $A_5$  is logged, similar to “Do not care” attribute, and  $(A_2 = False, A_5 = X)$  will also be pruned too. Therefore, the *VODKA* will ask the following questions.

**VODKA:** Does the attribute-value pair  $(A_2 = False)$  belong to the new variant object?

**EXPERT:** Yes.

**VODKA:** What is the name of the new variant object?

**EXPERT:** Blaster.B.

**VODKA:** Is the data type of  $A_2$  required to be changed?

**EXPERT:** Yes.

**VODKA:** Is any new attribute required to be added?

**EXPERT:** Yes.

**VODKA:** What is the name and value set of the new attribute?

**EXPERT:** UDP port, (0, 65535).

**VODKA:** What is the value of the UDP port attribute in Blaster.B and Blaster?

**EXPERT:** 69, X (means Do not care).

Hence, the Blaster.B acquisition table shown in Table 4.6 is generated.

**Table 4.6 The New Variant Acquisition Table of Blaster.B**

Attributes	Objects	Blaster	Blaster.B
300-thread ( $A_1$ )		X	X
System Reboot ( $A_2$ )		True	False
DoS Type ( $A_3$ )		Windows Update flood	Windows Update flood
Mail Attachment ( $A_4$ )		X	X
TCP Port ( $A_5$ )		{135;4444}	{135;4444}
UDP Port ( $A_6$ )		X	69

Consequently, a new original rule “*IF (System reboot = False) AND (DoS type = Windows update flood) AND (UDP port = {69}) THEN Blaster.B, CF=0.8*” and an embedded rule “*IF (System reboot = False) AND (DoS type = Windows update flood) AND  $\neg$  (UDP port = {69}) THEN Blaster.B, CF=0.5*” of new variant Blaster.B will be generated to classify Blaster.B.

**Table 4.7 The Adjusted Main Acquisition Table of Simple Computer Worms**

Objects Attributes	Nimda	Nimda.B	CodeRed	CodeRed .II	Blaster	Blaster.B	Welchia	Welchia. II
Threads	X	X	100	600	X	X	X	X
System reboot	X	X	True	True	True	False	True	True
DoS type	Email flood	Email flood	TCP flood	TCP flood	Windows Update flood	Windows Update flood	ICMP flood	ICMP flood
Mail_Attachment	{samle.exe ; puta!!scr}	{readme.exe }	X	X	X	X	X	X
TCP port	X	X	X	X	{135;4444}	{135;4444}	{80;135}	{80;135;445;3127}
UDP port	X	X	X	X	X	69	X	X

As shown in Table 4.7, four variants (Nimda.B, CodeRed.II, Blaster.B, and Welchia.II) have been successfully singled out using *VODKA* after several iterations. Table 4.8 shows the AOT table after interacting with domain experts manually using *EMCUD*

**Table 4.8 AOT Table of Simple Computer Worms**

Objects Attributes	Nimda	Nimda.B	CodeRed	CodeRed. II	Blaster	Blaster.B	Welchia	Welchia. II
Threads	X	X	2	2	X	X	X	X
System reboot	X	X	1	1	2	2	2	2
DoS type	D	D	1	1	2	2	1	1
Mail_Attachment	1	1	X	X	X	X	X	X
TCP port	X	X	X	X	1	1	2	2
UDP port	X	X	X	X	X	1	X	X

Table 4.9 shows the new embedded rule base of the discovered variants and original worms. If more real instances can be used, the embedded rule base will evolve and become more precise for classifying the computer worms.

**Table 4.9 The Rules Generated from Table 4.7 and Table 4.8**

R <sub>1,0</sub> :	IF (DoS type=Email flood) AND ( <i>Mail_Attachment</i> =(sample.exe;puta!!scr)) THEN Nimda, CF=0.8
R <sub>1,1</sub> :	IF (DoS type=Email flood) AND ( <i>Mail_Attachment</i> = $\neg$ (sample.exe;puta!!scr)) THEN Nimda, CF=0.4
R <sub>2,0</sub> :	IF (Threads=300) AND (System reboot=True) AND (DoS type=TCP flood) THEN CodeRed, CF=0.8
R <sub>2,1</sub> :	IF (Threads=300) AND $\neg$ (System reboot=True) AND (DoS type=TCP flood) THEN CodeRed, CF=0.6
R <sub>2,2</sub> :	IF $\neg$ (Threads=300) AND (System reboot=True) AND (DoS type=TCP flood) THEN CodeRed, CF=0.4
R <sub>2,3</sub> :	IF (Threads=300) AND $\neg$ (System reboot=True) AND $\neg$ (DoS type=TCP flood) THEN CodeRed, CF=0.4
R <sub>3,0</sub> :	IF (System reboot=True) AND (DoS type=Windows update flood) AND (TCP port={135;4444}) THEN Blaster, CF=0.7
R <sub>3,1</sub> :	IF (System reboot=True) AND (DoS type=Windows update flood) AND $\neg$ (TCP port={135;4444}) THEN Blaster, CF=0.57
R <sub>3,2</sub> :	IF (System reboot=False) AND (DoS type=Windows update flood) AND (TCP port={135;4444}) THEN Blaster, CF=0.43
R <sub>3,3</sub> :	IF (System reboot=True) AND $\neg$ (DoS type=Windows update flood) AND (TCP port={135;4444}) THEN Blaster, CF=0.43
R <sub>3,2</sub> :	IF (System reboot=False) AND (DoS type=Windows update flood) AND $\neg$ (TCP port={135;4444}) THEN Blaster, CF=0.3
R <sub>4,0</sub> :	IF (System reboot=True) AND (DoS type=ICMP flood) AND (TCP port={80;135}) THEN Welchia, CF=0.8
R <sub>4,1</sub> :	IF (System reboot=True) AND $\neg$ (DoS type=ICMP flood) AND (TCP port={80;135}) THEN Welchia, CF=0.67
R <sub>4,2</sub> :	IF (System reboot=True) AND (DoS type=ICMP flood) AND $\neg$ (TCP port={80;135}) THEN Welchia, CF=0.53
R <sub>4,3</sub> :	IF (System reboot=True) AND $\neg$ (DoS type=ICMP flood) AND $\neg$ (TCP port={80;135}) THEN Welchia, CF=0.4
R <sub>5,0</sub> :	IF (DoS type=Email flood) AND ( <i>Mail_Attachment</i> =readme.exe) THEN Nimda.B, CF=0.8
R <sub>5,1</sub> :	IF (DoS type=Email flood) AND $\neg$ ( <i>Mail_Attachment</i> = readme.exe) THEN Nimda.B, CF=0.5
R <sub>6,0</sub> :	IF $\neg$ (Threads=600) AND (System reboot=True) AND (DoS type=TCP flood) THEN CodeRed II, CF=0.9
R <sub>6,1</sub> :	IF (Threads=600) AND (System reboot=True) AND (DoS type=TCP flood) THEN CodeRed II, CF=0.3
R <sub>7,0</sub> :	IF (System reboot=False) AND (DoS type=Windows update flood) AND (UDP port={69}) THEN Blaster.B, CF=0.8
R <sub>7,1</sub> :	IF (System reboot=False) AND (DoS type=Windows update flood) AND $\neg$ (UDP port={69}) THEN Blaster.B, CF=0.5
R <sub>8,0</sub> :	IF (System reboot=True) AND (DoS type=ICMP flood) AND (TCP port={80;135;445;3127}) THEN Welchia II, CF=0.8
R <sub>8,1</sub> :	IF (System reboot=True) AND (DoS type=ICMP flood) AND $\neg$ (TCP port={80;135;445;3127}) THEN Welchia II, CF=0.5

In order to evaluate the effectiveness of *VODKA*, we apply *VODKA* on the e-learning application and replace the sequencing mining approach in this case study in the Appendix F.

In this chapter, we proposed *VODKA* methodology to iteratively discover the new variants objects through observing the behaviors of those embedded rules with marginally acceptable CF to assist domain experts in singling the variant objects out. Each iteration collects the sufficient inference behaviors of weak embedded rules and proposes three recommendations, including adding a new attribute-value of an attribute, changing the data type of an attribute, or adding a new attribute, to help experts discover the new variants according to the learned large itemsets. Additionally, we use Grid Merging to integrate the new variants acquisition table into the main acquisition table for adapting the weak embedded rules. A computer worm detection prototype based upon *DRAMA* has been implemented and deployed in an experimental environment to evaluate the performance of *VODKA*. The results show that new worm variants can be singled out of the corresponding extended worm object classes after the occurrence of worm instances in collected inference logs. Also, an e-learning case is shown that the variant learning behaviors can be discovered to assist teachers in preparing new learning content and learning sequence.

## Chapter 5

# Evolutional Knowledge Acquisition

Although *VODKA* is useful to discover variant knowledge, it is still weak to discover the insignificant variant or the variant evolutionary trend as time goes on. Hence, we propose *Trend Evolution Acquisition (TEA)* using time-based tracing technology to help experts trace the evolution of variants and generate new evolutionary knowledge to adjust the CF value of original embedded rules in each iteration of *Dynamic EMCUD*.

### 5.1 Trend Evolution Analysis

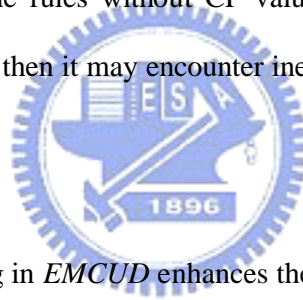
TEA not only applies *VODKA* to help experts construct the Attribute Signaling Table (AST) of each object but also offers more robust information to learn the evolutionary knowledge according to the evolved sequences of objects over time in a dynamic environment. The method focuses on maintaining an AOT increment of objects in *EMCUD* to represent the evolutionary behaviors of each object which is evolved as time goes by. An AOT increment records the relative importance of each attribute to each object for capturing the embedded meanings with acceptable CF value by relaxing or ignoring some minor attributes.

*TEA* consists of two Steps including Unfolding Step and Reconstructing Step to complete the adoption of AOT by monitoring the trend of inference behaviors. The

Unfolding Step is used to decompose the original AOT into several AST to keep the trend evolution of each attribute to each object, and The Reconstructing Step is used to construct the adoption of each AOT value in dynamic AOT according to an AST.

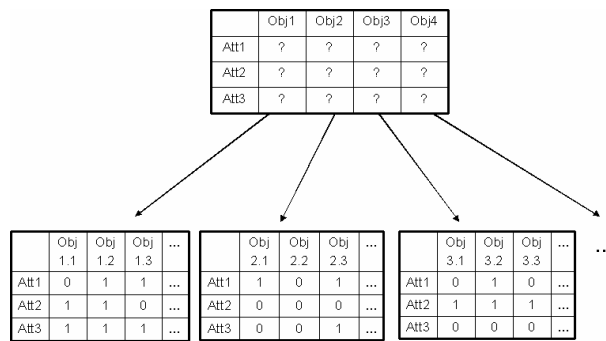
## 5.2 Capturing Evolutional Trend Using AST

As mentioned before, *VODKA* did solve several main problems in *EMCUD* but still not enough. Although the original idea of constructing dynamic AOT makes *EMCUD* more adaptive to elicit embedded meanings, it may difficult to assign the ordering values to all attributes since the knowledge is considered dynamic not static. It means that some rules today may become uncertain in the near future and vice versa. Moreover, *VODKA* learns the rules without CF values, and if the CF value is not adaptive in the past few days then it may encounter inefficiency in learning the variant object.



Although the AOT using in *EMCUD* enhances the ability of partially matching in initially embedded rule base to extend the coverage of recognized variant objects, however, it is still weak to immediately response the changing of environment according to the analyzing results of inference behaviors. Hence, *TEA* is to automatically adjust the AOT value according to tracing the trend of the evolutional behaviors in the changing environment. The importance of some minor attributes to each object might change in each time point according to the experts' point of view or the learning results from *VODKA*. Hence, a Boolean value can be simply assigned as “0” or “1” to each attribute in each time point, where “0” represents the attribute is considered unimportant to the object while “1” represents important to the object in this time point.

The “0” or “1” is called an attribute event  $e_t$  of each object in a time point  $t$ , and the attribute event sequence of “0” and “1” is recorded in AST to capture the evolved behavior of each object. Hence, the AOT increment can be generated for evolving the ordering values of attributes according to the sequence of “0” and “1” events recorded in AST with the times.



**Figure 5.1 Unfolding Step of Constructing AST**

The Unfolding Step of construction AST shown in Figure 5.1 records each specific information in each time point. Each entry can be filled by experts manually or by *VODKA* automatically according the observations of evolutionary evidence of each object. The importance degree values of attributes to objects might change in different time points in a dynamic environment.

Not all the attributes are considered to be working well with the mechanism because of some characteristics of the attribute. The domain expert can decide which attributes are required to be traced as time goes on if ordering values of the attributes are hard to be decided immediately. There are two ways of constructing AST:

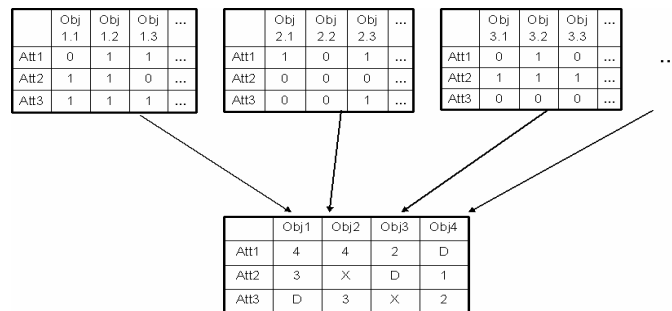


- (1) Interacted with human experts. It is designed to acquire attribute signals from a domain expert in every time point for deciding whether the attribute is important or not.
- (2) Interacted with *VODKA*. As mentioned above, *VODKA* can be used to learn and level up the certainty factor of each embedded rule. It can be helpful to decide attribute signal of importance or unimportance by directly mapping certain embedded rule to the AST.

After constructing AST, an entropy function or gracefully accumulating function can be proposed to capture the trend meaning and incrementally adjust the CF value of each embedded rule.

### 5.3 Constructing the Dynamic AOT

The reconstructing step shown in Figure 5.2 reconstructs the dynamic AOT by renewing the ordering values according to each “0” and “1” signal recorded with the times. Since the ordering values are recalculated at the present time according to all the information traced in a time interval, the AOT is considered more flexible and robust. Hence, to reconstruct the dynamic AOT will obtain the trend of evolutionary knowledge.



**Figure 5.2 Reconstructing Step of Constructing Dynamic AOT**

Based upon the AOT increment generated in evolutionary knowledge acquisition, the embedded rules generated by *Dynamic EMCUD* will be classified into original and embedded rules.

To capture the importance of each attribute to an object in a time interval, two scoring functions including *Entropy Function* and *Gracefully Accumulating Function* are proposed to learn the trend of evolutionary behaviors of new objects. Let the initial value of each signal sequence be the original AOT value of the attribute to the object.

### **(1) Entropy Function**

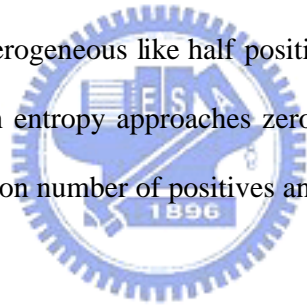
To transform the original AOT value to the representative values in AOT in a dynamic environment, the number of positives (recorded as “1”) surpasses the number of negatives (recorded as “0”), the entropy weight approximates zero and vice versa. It is obvious to assign higher ordering value when most of information represent important; otherwise, the lower ordering value would be assigned. However, medium ordering, considered uncertain degree of the attribute to objects, would be assigned when the entropy weight approximates one. That is, it is usually uncertain to decide a decision when getting half positive advices and half negative advices at a time. When the set is heterogeneous, but the characteristic of attribute is time relevant then it can be considered homogenous in some ways.

It is not simple to level up the ordering values of attributes when the attribute signal event is “1” or level down when it is “0”, instead; the entropy formula (5.1) is applied in the mechanism of transforming AST to AOT,

$$Entropy = -\frac{p}{p+n} \log_2 \left( \frac{p}{p+n} \right) - \frac{n}{p+n} \log_2 \left( \frac{n}{p+n} \right), \quad (5.1)$$

where “p” represents positive and “n” represents negative. In this case, the positive means that the attribute is decided as an important attribute to this object such as signal “1” in AST, and the negative is not an important attribute such as signal “0” in AST.

As you move from perfect balance or perfect homogeneity, entropy moves smoothly between zero and one. That means, the entropy is zero when the set is perfectly homogeneous like all positive or negative instances, and the entropy is one when the set is perfectly heterogeneous like half positive and half negative instances. Then it is obvious that when entropy approaches zero, the ordering value should be either “D” or “X” depending on number of positives and negatives in the set.



For example, suppose there are two sets in AST recorded as “000111” and “111000”. Both of “000111” and “111000” in entropy function are considered uncertain because of perfectly heterogeneous. Since these two sets can be considered time relevant, it is very certain that attribute with signals “000111” should be assigned a higher ordering since it is important in the present time intervals. Hence, there is a time bonus weighting to adjust each ordering value at final. Accordingly, each CF value of a rule can now be leveled up or down automatically after integrating AOT increment into the main AOT.

## (2) Gracefully Accumulating Function

Since the normal behavior may not change rapidly, a behavior scoring function is designed to calculate the score of each behavior to determine the trend of historical behaviors. Therefore, besides entropy function, the basic idea of the scoring function is to incrementally adjust the weight of each behavior. If a current behavior is normal, the score becomes larger; otherwise, it becomes smaller. The initial score value of each user is given 0.

Since the knowledge will be updated or evolved in a dynamic environment, the CF value of each embedded rule may be adjusted because the relative importance of the object may change. A dynamic AOT Adjusting Function (5.2) is designed to generate the updated AOT value at time  $t$  by accumulating the collection of attribute event  $e_t$  at time point  $t$  based upon the previous AOT value at time  $t-1$ . If the attribute event  $e_t$  is assigned as 1 then  $\gamma$  is set to 1, which represents the increment is added into the AOT value at time  $t-1$ . Otherwise,  $\gamma$  is set to -1 if the  $e_t$  at time  $t$  is 0, which represents the decrement subtracted by the AOT value at time  $t-1$ . Hence, the ordering values can be refined with the times according to the collected information in a changing environment.

$$AOT(t) = AOT(t-1) + \mathbf{g} \times f(g(t)), \quad \begin{cases} \mathbf{g} = 1, & \text{if } e_t = 1 \\ \mathbf{g} = -1, & \text{if } e_t = 0 \end{cases} \quad (5.2)$$

Where  $f(g(t))$ , which is formally defined in formula (5.3), is used to decide the increment or the decrement of the corresponding the AOT value at each time point  $t$ ,  $\alpha$ , which is used to adjust the curvature of the AOT Delta Function, increases resulting in rapidly increasing or decreasing of the CF value, and  $\mathbf{b}$ , which means the

weight of the number of consecutive “1” or consecutive “0” received, decreases resulting in larger increment or decrement. In order to limit  $f(g(t))$  between 0 and 1, the constant  $c$  is suggested to be smaller than -3.

$$f(g(t)) = \begin{cases} 0 & , \text{ if } e_t \neq e_{t-1} \\ \frac{1}{1 + e^{a \times (c+g(t)) \times b}} & , \text{ if } e_t = e_{t-1} \end{cases} \quad (5.3)$$

Where  $g(t)$ , the Continuous Events Accumulating Function given in formula (5.4), is used to record the number of consecutive “1” or consecutive “0” received at time  $t$ .

$$g(t) = \begin{cases} 1, & \text{ if } e_t \neq e_{t-1} \\ g(t-1)+1, & \text{ if } e_t = e_{t-1} \end{cases} \quad (5.4)$$

#### 5.4 Adjusting Certainty Factor of Collaborative Dynamic Knowledge

Since *Dynamic EMCUD* can be extended to the collaborative framework, each discovered dynamic knowledge learned from different local *KBSs* should be integrated to further applied. Three cases are used to assist experts in adjusting the CF values of the discovered knowledge of the new evolved objects from the collection of inference logs. Assume there are  $n$  local *KBSs* and each new evolved object may be discovered in  $p$  local *KBSs*, different CF values of a given embedded rule could be generated in each *KBS*. For  $p > 0$ , the CF Adjusting Function shown in formula (5.4) is proposed to help experts obtain the average of different CF value of a given embedded rule in each local *KBS* and adjust the scale of the CF increment or decrement (*DCF*) according to the discover of the new object in the collaborative *KBS*.

For each new embedded rule  $R_i$ , let the CF value be  $CF(R_i)$  and let the  $CF(R_i^j)$  be the CF value of each embedded rule  $R_i$  discovered in the  $j^{th}$  local *KBS*.

$$CF(R_i) = \frac{\sum_{j=1}^n CF(R_i^j)}{p} + \mathbf{d} \times \Delta CF \quad (5.4)$$

Depending on whether the new objects are discovered in the collaborative *KBS* or not, the coefficient  $\mathbf{d}$  can be defined as follows.

**Case 1:** the new object can be discovered in the collaborative *KBS*.

$\mathbf{d}$  is set to  $p/n$ .

**Case 2:** the new object can not be discovered in the collaborative *KBS*.

$\mathbf{d}$  is set to  $(p-n)/n$ .



For  $p = 0$ , since the new object can not be discovered in any local *KBS*, the new object could be discovered in the collaborative *KBS* according to the correlations of profiles. Therefore, the CF Adjusting Function could be reduced to formula (5.5), where the  $CF(R_i^c)$  is the CF value of the new discovered rule in the collaborative *KBS* due to different configurations of profile.

$$CF(R_i^c) = CF(R_i^c) + \mathbf{d} \times \Delta CF \quad (5.5)$$

$\mathbf{d}$  is set to  $-2$ .

## 5.5 Experiments

Up to now, many antivirus products have been developed to discover worms, virus or Trojan horse in a computer system. However, these products are hard to

automatically discover the variants of worms because the signature based approach fails when the signatures are changed. To overcome the weakness, we propose a worm detection prototype system, which has neo-learning module, to enhance the ability of commercial antivirus products by the collaborative framework. The case of worm detection is given to illustrate the idea of *TEA*. First, the domain ontology construction flow will be described and then Nimda worm is used as an example.

### 5.5.1 Computer Worm Ontology Construction

One of the purposes of applying ontology is to provide domain of discourse that is understandable by human and computers. Since ontology can be represented by machine readable markup languages such as RDF, the knowledge can be shared for different knowledge bases automatically through computers processing. Moreover, the reusability of ontology has become increasingly important to developers of intelligent systems.

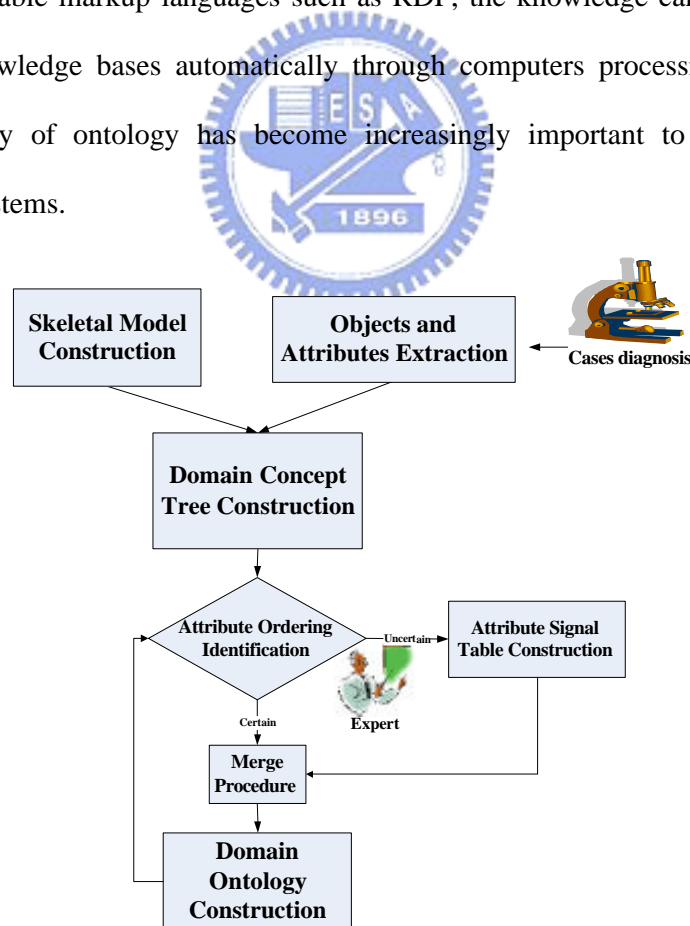


Figure 5.3 Worm Ontology Construction Flow

In this experiment, the ontology is not only reusable but also adaptive to the current environment. Also, we construct the ontology based upon a concept tree consisting of several prior knowledge including skeletal model and real cases provided by knowledge engineers and domain experts in *TEA*. In Figure 5.3, the flow of constructing worm ontology is illustrated to help experts construct the ontology more easily, the following four Steps are proposed:

**Step 1: Skeletal model construction.**

Create the skeletal worm model by identifying each worm with six general attributes including the basic information, the service, the exploitation, the carrier, the symptoms and the defense instruction.

**Step 2: Concept tree construction.**

Since it is often easier and more accurate for experts to provide critical cases rather than domain ontology, the power of critical cases described in terms of relevant objects and attributes to build domain ontology is remarkable. Therefore, after case diagnosing a concept tree is created based upon the skeletal model in **Step 1**.

**Step 3: Concept tree transformation.**

After concept tree is created, it is transformed into AOT, and attribute ordering will be next acquired from experts. Then the original EMCUD can be processed to generate the initial rules. However, because it is not easy to identify some attribute ordering values precisely, the attribute which is uncertain to identify the ordering value should be traced and analyzed with time by constructing an AST. Each attribute signal is recorded in each time interval, when the attribute appears important the



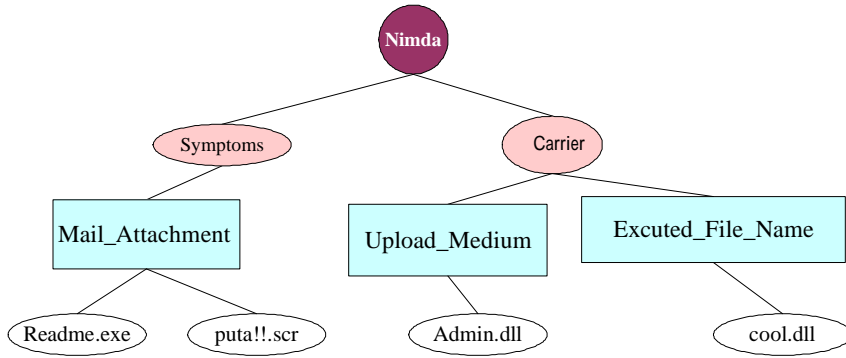
signal equals one and when it appears unimportant the signal equals zero. Therefore, the attribute ordering table will be reconstructed according to the attribute signals collected with time.

**Step 4:** Merge procedure.

Two relations “*has*” and “*is*” are used for constructing worm ontology during the merge procedure in this paper. The relation of “*has*” includes attribute ordering value, for example, when the attribute ordering value equals 3 then the relation should be “HAS:3”. Therefore, from **Step 3**, the ordering value would be retrieved from the reconstructed AOT by AST. Hence, the ontology can be easily transformed into AOT with updated value in *Dynamic EMCUD* whenever the variants are discovered.

**5.5.2 Example of Nimda Worm Detection**

Nimda, an incredibly sophisticated worm that made headlines worldwide, is taken as an example. Nimda is the first worm to modify existing web sites to start offering infected files for download by using Unicode exploit to infect IIS web server. It is the first worm to use normal end user machines to scan the vulnerable web sites. This technique enables Nimda to easily infect intranet web sites located behind firewalls. Assume a simple Nimda concept tree is created in Figure 5.4 after series of Nimda cases diagnosis, and it can be transformed into a worm AT like Table 5.1. The following attributes are considered: the name of the e-mail attachment used by worms, the medium used by worms to upload, and the name of the file used by worms to start execution on servers. After constructing the worm AT, we construct the initial AOT shown in Table 5.2.



**Figure 5.4 Example of Initial Nimda Concept Tree**

**Table 5.1 An Example of Original Nimda AT**

Onject	Nimda
Attribute	
Mail_Attachment	Readme.exe
Upload_Medium	Admin.dll
Executed_File_Name	Riched20.dll

**Table 5.2 An Example of Original Nimda AOT**

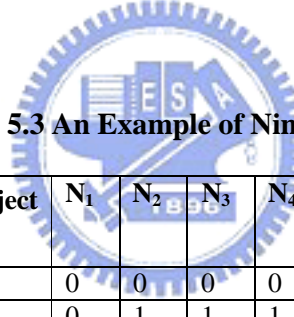
Object	Nimda
Attribute	
Mail_Attachment	2
Upload_Medium	3
Executed_File_Name	4

With both AT and AOT, the *EMCUD* can be processed to generate eight embedded rules and some of them have low CF value such as rule  $R_1$ : “*IF Not Mail\_Attachment = Readme.exe and Upload\_Medium = Admin.dll and Executed\_File\_Name = Riched20.dll Then Nimda*” with CF value = 0.67. Therefore, suppose that in the inference process, the rule  $R_1$  above is learned by neo-learning module almost all the time during a period, and suppose in the last two time points the embedded rule  $R_2$ : “*IF Not Mail\_Attachment = Readme.exe and Not Upload\_Medium*

= *Admin.dll and Not Executed\_File\_Name = Riched20.dll Then Nimda*” with CF value = 0.4 is fired, the AST in Table 5.3 to record the evolutionary trend can be obtained.

Suppose that Nimda is the latest worm occurred in the world, its ordering value of each attribute can not be easily determined because its variants may soon be broken out. The expert may define an AST with several time points, and then assign 0 in the first attribute,  $N_1$ , at first time point in Table 5.3. The attribute event  $N_2$  at the second time point is set to zero. For simplified discussion, we use gracefully accumulating function to adjust the AOT value of each attribute to each object according to the AST.

**Table 5.3 An Example of Nimda AST**



Object	$N_1$	$N_2$	$N_3$	$N_4$	$N_5$	$N_6$	$N_7$
<b>Attribute</b>							
<b>Mail_Attachment</b>	0	0	0	0	0	0	0
<b>Upload_Medium</b>	0	1	1	1	0	0	0
<b>Executed_File_Name</b>	1	0	0	0	1	0	0

In Table 5.3, the Mail\_Attachment attribute is calculated by Function 5.2, and the attribute is assigned a new ordering value = 1 since it is very possible to be changed again, subsequently, ordering value = 3 are assigned for both attributes Upload\_Medium and Executed\_File\_Name according to the AST. Therefore, the CF value of the rule  $R_I$  is leveled up from 0.67 to 0.74. Moreover, several new attribute-values are learned by neo-learning module with Mail\_Attachment = puta!!..scr in  $R_I$ , a new worm variant Nimda.B shown in Table 5.4 can be integrated into Table 5.5, and also an AOT is updated as shown in Table 5.6. Moreover, the Nimda ontology after discovering Nimda.B is updated as Figure 5.5.

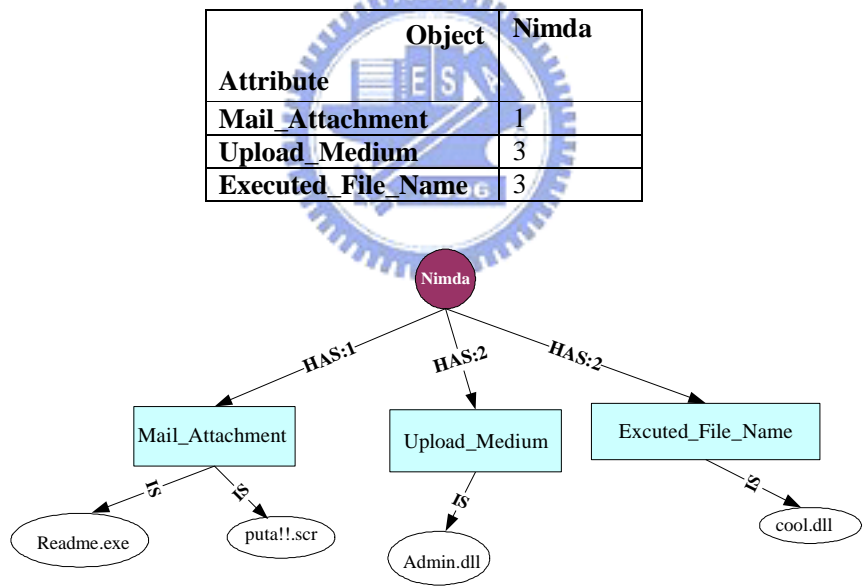
**Table 5.4 An Example of Updated Nimda AT After Discovering Nimda.B**

Object	Nimda.A	Nimda.B
Attribute		
Mail_Attachment	Readme.exe	puta!!.scr
Upload_Medium	Admin.dll	Admin.dll
Executed_File_Name	Riched20.dll	Riched20.dll

**Table 5.5 An Example of Integrated Nimda AT**

Object	Nimda
Attribute	
Mail_Attachment	{ Readme.exe; puta!!.scr }
Upload_Medium	Admin.dll
Executed_File_Name	Riched20.dll

**Table 5.6 An Example of Updated Nimda AOT After Discovering Nimda.B**



**Figure 5.5 The Updated Nimda Ontology after Discovering Nimda.B**

Therefore, with the accumulated inference logs from distributed sensors, the *TEA* can also update the knowledge frequently. Assume *VODKA* learns another new attribute values including Mail\_Attachment = sample.exe, Upload\_Medium = cool.dll, and Executed\_File\_Name = httpodbc.dll in  $R_2$  while the rule  $R_2$  has always been fired

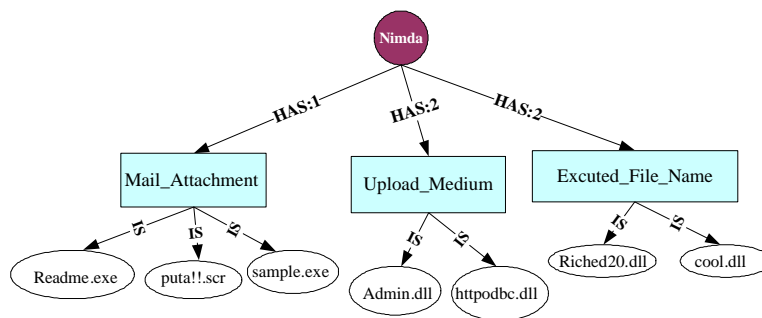
in each time point in a short period, then a new variant Nimda.E is found. Finally, based upon the updated tables shown in Table 5.7 and Table 5.8, the built system will give a whole picture of worms to guide the users who are not familiar in the domain for preventing or removing the malicious worms. Finally, the updated tables are shown in Tables 5.7 and 5.8, and the detailed of ontology of Nimda could be also updated as Figure 5.6.

**Table 5.7 An Example of Integrated Nimda AT After Discovering Nimda.E**

Object	Nimda
<b>Attribute</b>	
<b>Mail_Attachment</b>	{Readme.exe; puta!!.scr; sample.exe}
<b>Upload_Medium</b>	{Admin.dll; cool.dll}
<b>Executed_File_Name</b>	{Riched20.dll; httpodbc.dll }

**Table 5.8 An Example of Updated Nimda AOT After Discovering Nimda.E**

Object	Nimda
<b>Attribute</b>	
<b>Mail_Attachment</b>	1
<b>Upload_Medium</b>	2
<b>Executed_File_Name</b>	2



**Figure 5.6 The Updated Nimda Ontology after Discovering Nimda.E**

Owing to the different background and dynamic knowledge which can change with the times, the domain knowledge constructed at a time may become degraded in the near future. In this chapter, we propose a new knowledge acquisition method, called *TEA*, which traces information with times by interacting with human experts and supported by the learning strategy of *VODKA* to efficiently update the time-related domain knowledge according to the current environment. Therefore, we enrich the knowledge base and ease the effort of constructing the domain knowledge which is changing with the times and environment. Three cases will be used in collaborative framework to assist experts in adjusting the CF values of the discovered knowledge of the new evolved objects from the collection of inference logs. A worm detection system is illustrated to ease the experts' efforts from analyzing and learning and to help retrieving meaningful information for making proper decisions since the knowledge bases become more adaptive for a changing environment by using *TEA*.



## Chapter 6

### Application in Worms and DDoS Detection

A *Worm Immune Service Expert system (WISE)* with *Dynamic EMCUD* and a worm classification embedded rule base is implemented to discover the new variant worms generated by the attacking traffic generator in the experimental environment to evaluate the performance of our proposed method. A DDoS intrusion tolerance system is also implemented.

#### 6.1 The Background of Worms and DDoS Attack

##### 6.1.1 The Introduction of Computer Worms

In recent years, computer worms are grown dramatically to influence the wide computer networks due to the property of easily modifying the source code of original computer worms to create new variant for escaping the detection of related systems, e.g., Symantec Norton [72], Network Viruswall [74], etc. Generally speaking, computer worm usually self-propagates through the following four stages: Target selection, Exploitation, Infection, and Propagation [80]. In Target Selection Stage, 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 Stage, in which a worm compromises the target by exploiting a particular vulnerability and published exploits. If success, the worm goes to Infection Stage, in which the worm sets up on the newly infected machine. Finally, in Propagation Stage,

the worm starts to spread by choosing new targets. And another victim will enter the next four Stages cycle.

### **6.1.2 The Introduction of DDoS Intrusions**

As mentioned above, many different DDoS attacking tools and defending methods [17] to help mitigate the malicious traffic developed result in the rapid growth of complicated characteristics of DDoS intrusion tolerance in recent years. The introduction of DDoS is given in Appendix A.

As we know, there are two different types of attack technique in DDoS attacks: bandwidth consumption and resource consumption. The bandwidth consumption means that the attacking traffic launched by the compromised hosts, which are controlled by attackers, is aggregated to a single huge flood and overwhelms the victim. The resource consumption means that attackers make use of the leak of the network protocol or the system security such as the techniques of SYN flood, land and Teardrop, resulting in the starvation of system resources [16].

As the DDoS attack tools have become more complicated in recent years, the maintenance of the characteristics of DDoS attacks is becoming more difficult despite the previously known common characteristics of each category of discovered DDoS attacks. Therefore, we will propose a knowledge base to store the characteristics of DDoS attacks, which may be obtained by analyzing the traffic behaviors of the DDoS attacking tools, for DDoS intrusion tolerance. Besides, two criteria considering the difference between two types of DDoS attacks will be proposed to evaluate the degree of intrusion tolerance.

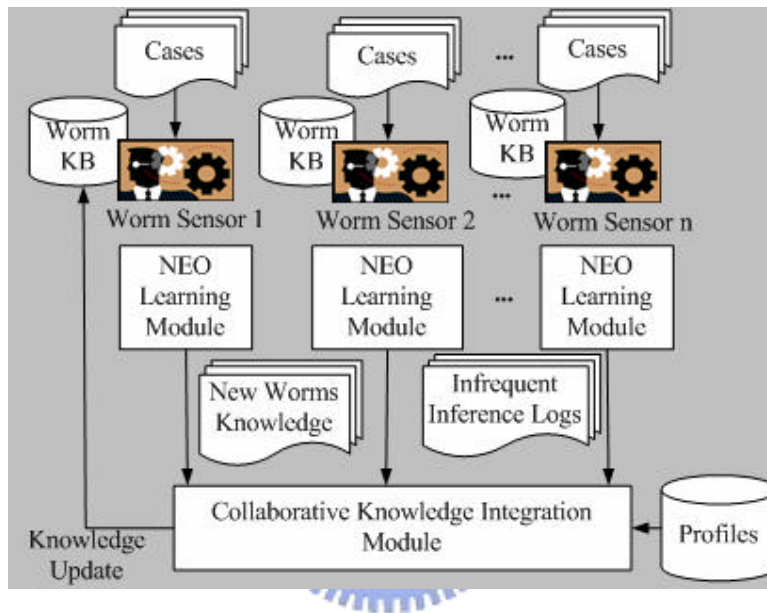


Intrusion tolerance is the ability of a system to continue providing (possibly degraded but) adequate services after a penetration [70]. As mentioned above, it is very hard to detect and prevent the DDoS attacks. Therefore, the intrusion tolerance of DDoS attacks is an important issue to mitigate the damage during DDoS attacks for providing the critical services continuously on Internet. Although a variety of methods, which are given in Appendix A, have been proposed to mitigate the damage during DDoS attacks for providing the critical services continuously, it is still very difficult to keep up with the rapid growth of DDoS expertise in their studies. To solve this problem, a DDoS ontology is proposed to provide a common vocabulary among domain experts and an integrated knowledge acquisition framework is then proposed to assist in quickly accumulating their expertise. We also use the behaviors of access control list to evaluate the performance of the DDoS models.

## 6.2 The Framework Worm Immune Service Expert System

As we know, many antivirus products have been proposed to discover worms, virus or Trojan horse in a computer system. Although these antivirus softwares are developed to protect our system well, it is hard to automatically discover the variant worms without updating their signature database because the signature the worm signatures may change over times. To overcome the weakness, the worm detection prototype system, namely *WISE*, is proposed to enhance the commercial antivirus products instead of replacing them. *WISE* is a knowledge-based system. Unlike pattern matching system, it does not need to write the program again, and therefore is suitable for worm, which is usually variant quickly that updates knowledge base frequently. By only updating the knowledge base, *WISE* can modify the defense me

chanism for the variants of worm; as a result, the system can be easily maintained. Besides, *WISE* contains embedded meanings of knowledge, so it can easily capture some variant worms that in order to avoid signature-based detection system to modify characteristic less. Since the growth of the knowledge of worms is very fast, we propose a collaborative architecture for the adaptive worm detecting problem.

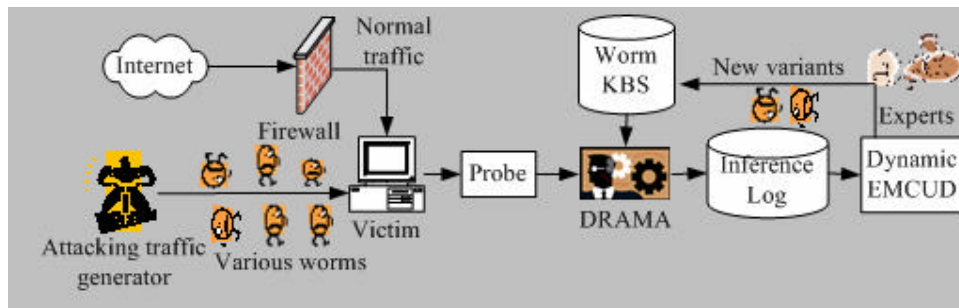


**Figure 6.1 The Collaborative Framework for Worm Detection**

Figure 6.1 shows the collaborative framework for worm detection. In the architecture, each worm sensor provides a web interface to collect or discover all the symptoms of worm cases by user and scanning tools. The NEO-learning module helps each worm sensor constructing AST to reconstruct AOT increment and update main acquisition table using AT increment (monitoring the frequent inference logs of weak embedded rules of worms with the times), where each sensor has its own Worm KB. For example, when worm infects a victim system, the user can scan the host computer by some general antivirus software or can call for help from the Internet. The system

collects all the information and infers the information based upon the worm knowledge with embedded meaning constructed by *EMCUD*. Consequently, the result of inferring will be passed to the users to teach the way of recovering the system. Moreover, the statuses which satisfy certain embedded rules will be considered to learn the new knowledge of new variant worms by neo-learning module. By collecting the new worms knowledge and infrequent inference logs and consulting the Profile, the collaborative framework can integrate the new worm knowledge.

In our *WISE* system, the knowledge of computer worms can be divided into several KCs, including the service provided by host may be infected by certain worms and then produced some symptoms in host or network. Some worm lifecycle, Profile model, and dynamic behaviors knowledge classes are also created. The Log Collecting Stage will be encoded by four meta-rules in *DRAMA*; the Knowledge Learning Stage and *Dynamic EMCUD* are implemented using the JSP to make a communication channel using the *API* provided by *DRAMA*. The related attributes of various computer worms can be collected by some probe tools and used to evaluate the ability of *Dynamic EMCUD*, which deployed in the prototype system.



**Figure 6.2 The Experimental Environment for Detecting Computer Worms**

In order to evaluate the *WISE*, an experimental environment shown in Figure 6.2 for detecting various computer worms is built. In this environment, the victim is received both the normal traffic and the attacking traffic (various worm behaviors). All received traffic can be treated as normal or attacking behavior, which can be transformed as attribute-value pairs. The network traffic collected from Internet is assumed as normal traffic since most attacking behaviors with significant signatures will be filtered by firewall. The attacking traffic generator is designed to randomly generate various worms attacking traffic to infect the victim. Besides the attacking traffic, some signatures, e.g., the system status, host vulnerability information, and large e-mailing behavior, of the victim infected by worms can be also collected. The probe, such as Nessus [73], is also used to automatically collect these worm related attributes (symptoms). Then, these attributes is used to trigger the corresponding classification rules in worm KB. If variant worms occurred frequently in a period, some candidate worm variants may be discovered by *Dynamic EMCUD*. Finally, the corresponding embedded rules of variant worms confirmed by experts will be generated to update the worm *KB*.

### 6.3 DDoS Intrusion Tolerance

As we know, the traditional methods for detecting and filtering DDoS attacks [27] are monitoring the status of network and system, specifying the alert thresholds, defining detection rules, and setting filter policies by domain experts. Based upon interviewing with domain experts, the DDoS ontology proposed to models the behaviors of system and users, the methodologies of defense, and the strategies of

evaluation are described as follows.

### 6.3.1 Ontology of DDoS

Before understanding the more complicated DDoS knowledge, an ontology, which is needed for sharing knowledge with a common terminology among numerous experts of the DDoS domain, could be divided into three parts: Profile model, Defense model, and Evaluation strategy as shown in Figure 6.3. The Profile model is proposed to describe the behaviors of system and users according to the state and user state diagrams. The Defense model consisting of Detection and Filter methodologies is then used to resist the DDoS attacking. Finally, the Evaluation strategy including system and network evaluations is proposed to evaluate the performance of our proposed DDoS model. Hence, the ontology includes Profile model, Detection methodology, Filter methodology and Evaluation strategy knowledge classes (KCs), each may include several sub-KCs and may be obtained by interviewing with domain experts, e.g., Attack predicting and Attack detecting are sub-KCs of the Detection KC.

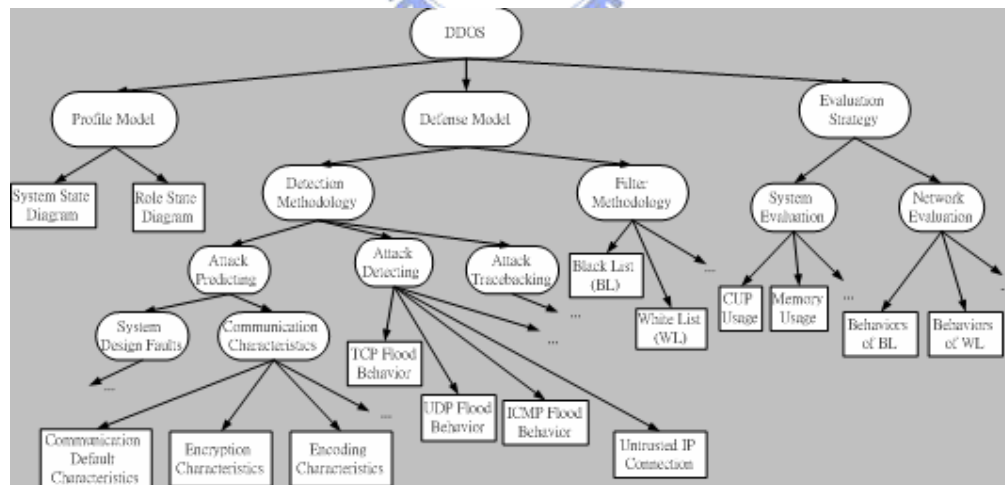
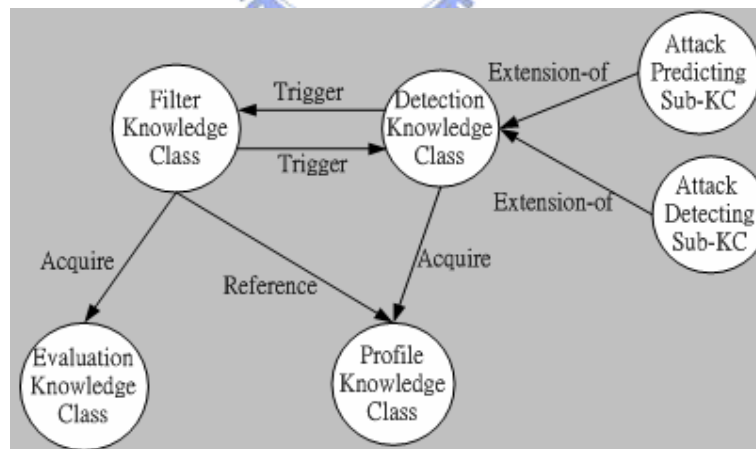


Figure 6.3 The Ontology of DDoS

### 6.3.2 The Relationships Between Knowledge Classes

Four basic relations between KCs have been defined in *Drama/NORM* [46]:

Acquire, Trigger, Reference, and Extension-of relation. These relations are helpful in describing the relationships among KCs. Trigger relation triggers another KC with current facts as knowledge transfer. In other words, the remnant knowledge in original KC should not be necessarily considered. Acquire represents the acquirement relation. After Acquire process, the original inference process will continue and only facts predefined in the acquired KC will be carried back in Acquire relationship. Reference is used to represent the associations between different KCs. Through the Reference relation, the knowledge contained in referred KC is regarded as the base knowledge and will be taken into consideration together with the knowledge defined in the KC. On the other hand, Reference can be thought as an unconditional Acquire relation between KCs. Unlike the Reference relation, the Extension-of relation makes a new KC to include all the knowledge contents of an existent KC. The activities of Extension-of relation include extension and modification. Therefore, it must support the overriding mechanism, including the overriding of facts and rules.



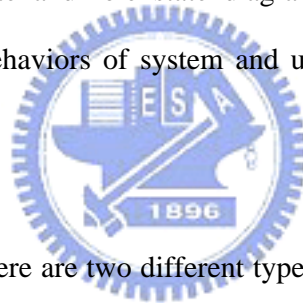
**Figure 6.4 Relationships Between of Knowledge Classes**

As shown in Figure 6.4, the Filter KC referencing the Profile KC can be treated as a filter to filter out the malicious traffic and can be triggered by the outside traffic

events. Also, it can set the new filter policy and acquire the Evaluation KC to evaluate the system performance. The Detection KC triggered by the Filter KC could be treated as a detector to detect the occurrence of DDoS attacks and could trigger the Filter KC according to the specific detection events for dynamically setting the new filtering policy to filter the malicious packets. Also, the Detection KC can acquire the Profile KC to set the suitable attack detecting or attack predicting sub-KCs, which are included by the Extension-of relation.

### 6.3.3 Profile Model and Evaluation Strategy

According to the expert's experiences of defending DDoS attacks, the Profile model including system state and role state diagrams shown in Appendix B are proposed to represent the behaviors of system and users through interviewing with domain experts.



As mentioned above, there are two different types of DDoS attacking technique: bandwidth consumption and resource consumption. Since the only way can stop a DDoS attack once it starts is to identify the addresses of all agents (zombies) sending DDoS packets and to shut off traffic from them, the behaviors of black list and white list are considered to monitor the potential latency of the network and the CPU usage and memory usage are used to monitor the degrading rate of system performance when suffering the DDoS attack and the *Tolerance* of the initial ACL is assumed to be maximal in this chapter.

During resource consumption DDoS attacks, the system capacity of the victim is always decreasing (i.e., the system resource usage  $r$  is increasing), causing *Tolerance*

low. On the other hand, the bandwidth consumption DDoS attacks may increase the members of the black list and decrease the members of the white list due to filtering the malicious traffic; hence the *Tolerance* will become small if the filter policy could not be performed well. Thus, the formula 6.1 combining network tolerance ( $Tolerance_{network}$ ) and system tolerance ( $Tolerance_{system}$ ) is given to represent the degree of DDoS intrusion tolerance, where  $\mathbf{a}$  and  $\mathbf{b}$  are used to indicate the weights of the tolerance. If we focus on protecting network performance,  $\mathbf{a}$  is set to be larger than  $\mathbf{b}$ . Otherwise, a large  $\mathbf{b}$  is recommended to protect the system performance.

$$Tolerance = \mathbf{a} \times Tolerance_{network} + \mathbf{b} \times Tolerance_{system} \quad (6.1)$$

Since the members in the white list ( $WL$ ) may represent they could be served and the members in the black list ( $BL$ ) may represent they could be blocked by the victim, the ratio of the  $WL$  and the  $BL$  is used to evaluate the network bandwidth utilization ( $Bw$ ). A large  $BL$  implies the filtering policies are set to be more restricted and the tolerance may become small; otherwise, a large  $WL$  may represent the system is with more tolerance since more users can access the services of the victim. The number of users who have been moved from white list to black list ( $W2B$ ) is further regarded as a penalty weight for network tolerance. The formula of the  $Tolerance_{system}$  and the  $Tolerance_{network}$  are shown as 6.2. and 6.3 respectively.

$$Tolerance_{system} = 1 - \mathbf{r} \quad (6.2)$$

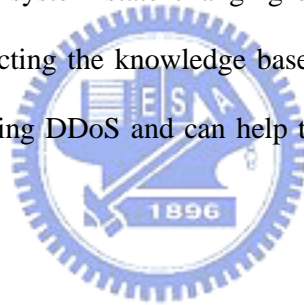
$$Tolerance_{network} = \frac{WL}{BL + W2B} (1 - Bw) \quad (6.3)$$

The system state will be set as NORMAL when the *Tolerance* is larger than the predefined thresholds. Otherwise, the system state will be in SURVIVAL state and the new filter policy based upon Evaluation knowledge will be generated to move the



state to NORMAL.

As mentioned above, each KC may include several sub-KCs due to the hierarchy of the knowledge in DDoS intrusion tolerance. In order to obtain the knowledge of each KC, an integrated knowledge acquisition (KA) framework including interviewing with domain experts, training the predicting and detecting features, and learning the filter policies for adaptively filtering malicious traffic is proposed. All knowledge of DDoS intrusion tolerance can be represented as a natural rule format, IF *Conditions* Then *Conclusions*. The bodies of *Conditions* are the facts generated from network traffic flow, detecting results, and filtering policy. The *Conclusions* may include the alarming events, system state changing events, and user state changing events. Furthermore, constructing the knowledge base can facilitate the maintenance of the knowledge for defending DDoS and can help the administrators manage their networks.



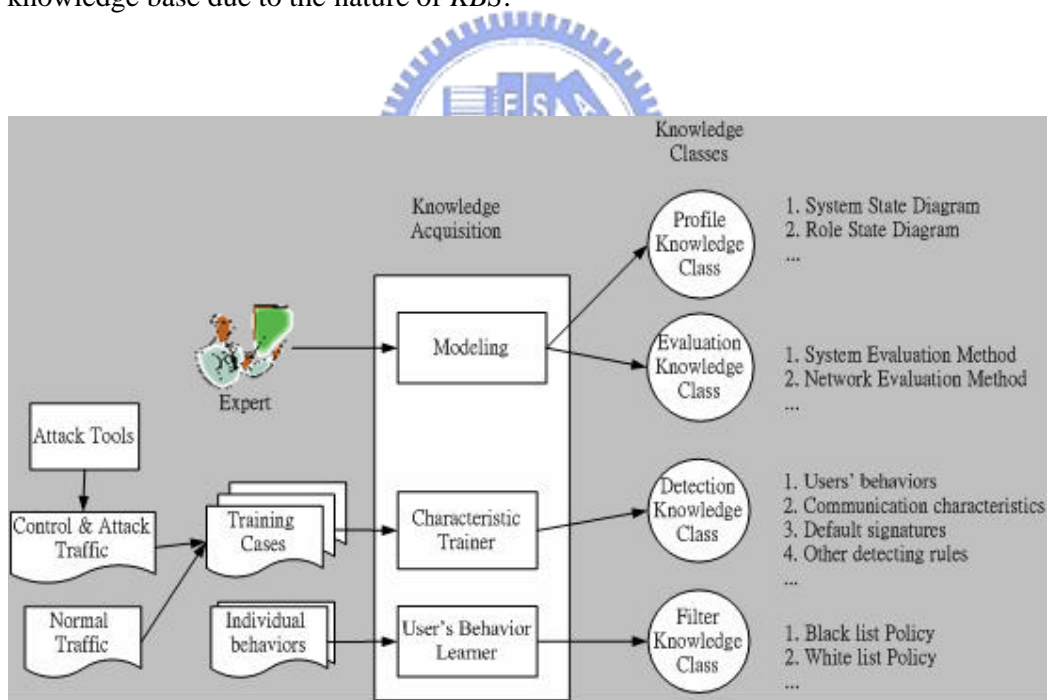
#### **6.3.4 Knowledge Base Construction**

Due to the difficulty of acquiring and collecting the various DDoS characteristics from domain experts, an integrated KA framework including three related KA methods is used for reducing the effort of accumulating the expertise and speeding up the knowledge collection of DDoS characteristics.

#### **The Integrated Knowledge Acquisition (KA) Framework**

The problems that we are faced with during the KA process are usually very hard. In general, KA involves: (1) elicitation (gathering) of data from the expert, (2) interpretation of the data to infer the underlying knowledge or reasoning procedure

and (3) guided by this interpretation, creation of a model of the expert's domain knowledge and performance. Although quite many different kinds of KA approaches have been proposed in many research studies [61][31][83][54][75][79] including interviewing with experts, Repertory Grids, and machine learning, few studies have focused on integrating various kinds KA approaches for an application due to the lack of a common vocabulary. Based upon the DDoS ontology we mentioned above, an integrated KA framework is proposed in this paper. Through the KA framework shown in Figure 7.5, four kinds of KCs including Profile KC, Evaluation KC, Detection KC, and Filter KC could be easily obtained by various KA processes. Other new discovered or defined KCs can also be easily added or modified in our knowledge base due to the nature of *KBS*.



**Figure 6.5 The Framework of KA Process**

As shown in Figure 6.5, all of these KCs of expertise can be obtained in the integrated KA framework, which includes modeling the DDoS environment through

interviewing with domain experts, selecting useful features by analyzing the attacking tools in the Characteristic Trainer, and adaptively learning filter policies in the User's Behavior Learner. The behaviors of users, communication signatures, and other useful features are the characteristics of the Detection KC, so called detecting rules, which can be obtained by the Characteristic Trainer. On the other hand, the User's Behavior Learner is responsible for generating the various filtering policies in Filter KC. All other KCs including Profile KC and Evaluation KC can be directly obtained by interviewing with experts.

### **(1) The Knowledge Obtained by Interviewing With Domain Experts**

As interviewing is one of the traditional approaches to acquire the expertise from domain experts by knowledge engineers, many approaches have been proposed to acquire expertise from experts through interviewing. As mentioned above, the Profile KC and the Evaluation KC could be modeled by domain experts using interviewing approach. Besides, the default knowledge including default communication ports and the white list and black list policy in the Detection KC and the Filter KC could be also acquired by interviewing with domain experts. Because the characteristics of DDoS attacking tools and the filtering knowledge are dramatically increasing, the Characteristic Trainer and the User's Behavior Learner are proposed to obtain the useful knowledge for DDoS intrusion tolerance.

### **(2) Training The Detection KC by Characteristic Trainer**

To obtain the previously undiscovered DDoS characteristics/behaviors, a training process, namely Characteristic Trainer, is proposed to learn the useful, new features and store them into the Detection KC for predicting and detecting DDoS attacks.

The new features of DDoS attacks can be selected by comparing the normal behaviors during the NORMAL system state with the attacking behaviors launched by DDoS attacking tools in the systematic training process of Characteristic Trainer. Thus, to distinguish attacking behaviors from normal behaviors, each kind of characteristics represented from the DDoS behaviors could be easily identified using a Repertory Grids approach, which is a table with four attributes including the feature, the feature threshold  $d$ , the feature operation  $q$ , and the corresponding actions. The  $d$  is a parameter adaptively determined in a short period by different features. For example, if one feature value is larger than  $d$ , it needs to be considered as abnormal behaviors. Therefore, the small  $d$  will increase more false alarms. On the contrary, larger  $d$  will treat more attacking behaviors as normal. After a DDoS attacking feature is detected, the corresponding action e.g., trigger the Filter KC, alarm the attacking traffic coming, or specify the attacking type, must be taken. The characteristic training algorithm is shown as Algorithm 6.1.

In addition to the above previously known features, new characteristics/features may be observed by using the Repertory Grids approach after analyzing fingerprints of DDoS attacks such as spoofing, flooding-based and communication techniques and more attacks could then be detected and predicted. More training DDoS characteristics are shown in Appendix B.

### Algorithm 6.1 The Characteristic Training Algorithm

**Input:** Training Cases  $TC$ , Actions  $A$ , and feature set  $F$  with  $n$  features  
**Output:** The Detection KC

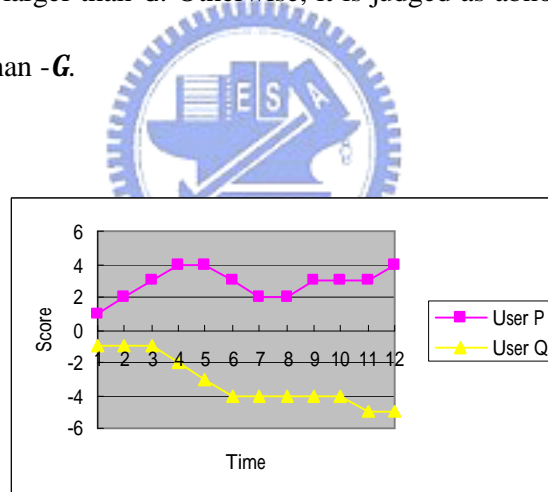
**Step1:** Select a skeleton feature set  $F_k = \{f_1, f_2, \dots, f_k\} \subseteq F$  by interviewing with domain experts.  
**Step2:** Set  $d_i$ , for each feature  $f_i$  in  $F$  in each  $TC$ .  
**Step3:** Choose the proper actions  $A_i$  for the feature  $f_i$  selected in **Step2**.  
**Step4:** Generate the detection rule as “**IF**  $f_i$   $q_i$   $d_i$  **THEN**  $A_i$ ”.  
**Step5:** Repeat **Steps 2~4** until all detection rules have been generated.

### (3) Learning The Filter Knowledge Class by User’s Behavior Learner

In traditional network management system such as firewall, intrusion detection system (IDS), network management system (NMS), etc, ACL is widely used not only to filter suspected connection from untrusted sources but also to admit the access from trusted sources. Black list and white list strategies used in ACL are included in the Filter KC. The former is used to interdict the access right, but the latter is used to permit the access right. Moreover, the various filtering policies can be set according to the configurations of current system and network environments.

In order to dynamically construct suitable ACL to mitigate the damage of DDoS attack, a learning process is also proposed to generate appropriate filtering policies for various network environments. The black list is used to drop the malicious attacking traffic and the white list is used to allow the trusted IP to access the critical service of the victim server. The principles of the User’s Behavior Learner for ACL are twofold in this paper. One is to keep the legal users, whose behaviors are determined as normal, in the ACL. The other is to remove the possible suspected users or the users, who do not request critical service for a long time, from the ACL.

Since the normal user may not change his/her behavior rapidly, a user behavior scoring function is designed to calculate the score of each user by his/her own behavior to determine his/her status of historical behavior. The basic idea of the scoring function is to incrementally adjust the weight. If a current network status is normal, the score becomes larger; otherwise, it becomes smaller. The initial score value of each user is given 0 and the behavior of user changes from  $A$  to  $N$  or from  $N$  to  $A$ , the score is no change. To evaluate the historical network status of the user  $q$ ,  $G$  is defined by expertise to determine the user's historical behavior network status. The historical behavior of the user ranging from  $[MAX, MIN]$  is determined as normal ( $M(q)$ ) if the score is larger than  $G$ . Otherwise, it is judged as abnormal ( $M(q)'$ ) when the value is smaller than  $-G$ .



**Figure 6.6 An Example of Users' Behavior**

In Figure 6.6, an example of  $P = \langle N, N, N, N, A, A, A, N, N, A, N, N \rangle$  and  $Q = \langle A, N, A, A, A, A, N, A, N, A, A, N \rangle$  is given to explain our proposed scoring function. The final score of  $P$  and  $Q$  are 4 and -5, respectively. If  $G = 3$ , the behavior of  $P$  is  $M$  and the behavior of  $Q$  is  $M'$  in this example.

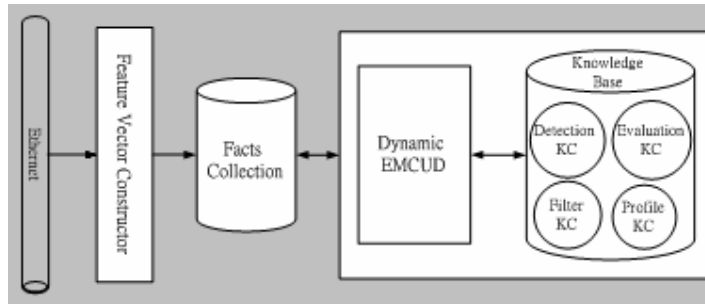
To more accurately categorize the user behavior, a penalty weight ( $PW$ ) could be attached when the characteristic of DDoS attack is discovered from individual user. The range of  $PW$  could be also defined according to the degree of dangerous behavior.

### **6.3.5 The Verification of Selected Features/Characteristics of DDoS**

To evaluate the selected features/characteristics of DDoS attacks, the selected characteristics of DDoS which are similar to the normal behaviors will be eliminated due to the reduction of the false detection rate in the DDoS intrusion tolerance system. And then the Drama-based DDoS intrusion tolerance system will be implemented to evaluate the performance of detection power by the selected characteristics.

#### **(1) NORM-based DDoS Intrusion Tolerance System Using KCs**

In order to evaluate the efficiency of the KCs, the DDoS intrusion tolerance system using KCs is implemented by an inference engine Drama for detecting and predicting the occurrence of DDoS attacks. As shown in Figure 6.7 the four KCs could be easily used to infer the other system components through the inference engine, and each component can be easily replaced according to different configurations of the network environment. The network traffic can be characterized as feature vectors [45] by Feature Vector Constructor. The Facts Collection is used to store all facts including the feature vectors, detecting results, system status, user states, and system evaluation results.



**Figure 6.7 The DDoS Intrusion Tolerance System Using Dynamic EMCUD**

When the anomaly network traffic is detected in the Detection KC, the event of attacking traffic coming is triggered and the Profile KC is acquired to change the state of system. And the alarm event is thus triggered to set the suitable ACL in Filter KC for dropping the huge traffic from the attackers and allow the legitimate traffic from trusted users. Since then, the event of updating filter rules would be triggered to generate a new filter policy for dropping the malicious traffic. It implies the Evaluation KC will be acquired and used to compute the tolerance of the system for updating the filtering policy of the Filter KC. Finally, the Profile KC would be triggered to indicate the proper system state. However, the complex attacking behaviors sometimes make the filtering policy fail. When it failing, the event of generating new filter policy would be triggered again until system state in Profile KC is stable. Otherwise, experts are asked to solve the problem of DDoS attacks.

## **6.4 Knowledge Base Maintenance**

Due to the growth of rule base usage, the scale of rule base is increasing, and hence the performance degradation becomes an important issue about constructing the rule base arise. The performance can be dramatically decreased and hence more resources may be required by the inference engine of the knowledge base when the

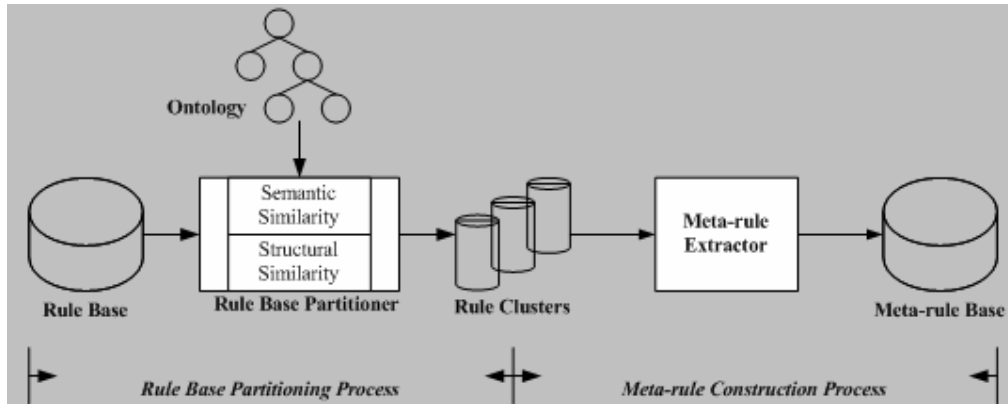


number of rules in the rule base increases. In this section, a *Rule base Partitioning for Meta-knowledge Extraction System (RP-MES)* combining both rule base partitioning and meta-rule construction mechanisms is proposed to solve these issues. As for rule base partitioning [62], *RP-MES* considers not only the structural relatedness between rules but also the semantic relatedness of rules by calculating the semantic relationship between rules in the rule base. In order to maintain the increasing knowledge base, the meta-knowledge is useful to help select suitable rules in inference process in the *KBS*.

#### **6.4.1 Rule Base Partitioning and Meta-rule Extraction System**

Meta-rules provide some related information about each rule cluster. Meta-rules can be used to select appropriate rule clusters which increase the performance of the usage of rule base in inference process. With meta-rules, the structure of the rule base can be easily understood. However, meta-rules can not be easily obtained. In previous applications about meta-rules, the set of meta-rules are usually provided by domain experts; acquiring meta-rules can be time consuming or the expertise may be not available. Therefore, a systematic mechanism is desirable to generate meta-rules.

The Automatic Meta-rule Constructor shown in Figure 6.8 consists of Rule Base Partitioning Process and Meta-rule Construction Process. Rule Base Partitioning Process considers syntactic and semantic structures of given rule base, and then partitions rule base into several rule clusters. In Meta-rule Construction Process, meta-rules will be extracted from the rule clusters obtained in previous phase.



**Figure 6.8 Automatic Meta-rule Construction**

#### 6.4.2 Rule Base Partitioning Process

Rule Base Partitioner is used to group rules into rule clusters from a plain rule base without any structure. At the beginning, each rule of the original rule base is allocated into a single rule cluster. Cluster Similarity Calculator calculates cluster similarity of all pairs of two distinct rule clusters and builds a Cluster Similarity Matrix (CSM). And the rule clusters will be merged according to the information in Cluster Similarity Matrix. The merging process works iteratively until all similar rule clusters are merged. And the rule clusters generated will be the result of this process.

However, the similarity calculation can seriously affect the result of this process, and the merge process of rule cluster is also an important task. In the following paragraphs of this section, similarity calculation will be introduced first. After that, rule base partitioning algorithm will be detailedly described in the forthcoming section.

## ■ Rule Similarity

Rule similarity is a key factor to the clustering result. Several kinds of rule similarity definitions considering structural relatedness only [36][43], or with semantic relatedness (hybrid approach) [42][77] are defined in previous work. In this paper, we incorporate hybrid approach to deal with rule similarity calculation. In the following, we give an example to present the rule similarity calculation.

Before discussing the rule similarity calculation, some notations are given in Definition 1 to be used in following discussions.

**Definition 1.** Expressions, conditions, actions, rules, and rule base.

$A = \{attribute_1, attribute_2, \dots, attribute_N\}$  is the set of  $N$  attributes in the rule base.

$O = \{=, ?, >, <, >=, <=\}$  is the set of all operators used in the expressions.

$V_{attribute_m}$  is the set of possible values of  $attribute_m$ .

$e_m = (attribute_m operator_m value_m)$  is an expression, where  $attribute_m \in A$ ,  $operator_m \in O$ , and  $value_m \in V_{attribute_m}$ .

$r_i$  is a rule of two-tuple ( $CONDITIONS_i, ACTIONS_i$ ) which can be represented as “*IF CONDITIONS<sub>i</sub> THEN ACTIONS<sub>i</sub>*”, where  $CONDITIONS_i$  is a set of expressions of rule  $r_i$ , and expressions in the set are connected with conjunction operator (*AND*) and  $ACTIONS_i$  is a set of expressions of rule  $r_i$ , and expressions in the set are connected with conjunction operator (*AND*), where the operator of the expressions must be “=”.

RB is the set of rules in the rule base. We illustrate the Example in Appendix C.

As we have mentioned before, rule similarity calculation containing structural relatedness and semantic relatedness is described as follows.

### (i) Structural relatedness

The structural relatedness considers the reference of attributes between rules, that is, evaluating the same attributes or asserting new values to the same attributes. When considering the reference of attributes, only the name of attribute is considered instead of attribute value. In the definition of structural relatedness, two rules are related if there exists one attribute used by both rules (either on left- or right-hand sides); otherwise they are independent. The structural relatedness between two rules is thus measured by the number of attributes that are mentioned in both rules. Definition 2 defines four situations of rule dependency, including in-out, share-in, share-out, and not-shared, and four corresponding functions, *inout()*, *sharein()*, *shareout()*, and *notshared()*.



**Definition 2.** *inout()*, *sharein()*, *shareout()*, and *notshared()* Functions.

Given two rules  $r_i = (\text{CONDITIONS}_i, \text{ACTIONS}_i)$ , and  $r_j = (\text{CONDITIONS}_j, \text{ACTIONS}_j)$ , their definitions are defined as below:

*inout*( $r_i, r_j$ ) : the set of attributes that are used in  $\text{CONDITIONS}_i$  and  $\text{ACTIONS}_j$ , or  $\text{ACTIONS}_i$  and  $\text{CONDITIONS}_j$ .

*sharein*( $r_i, r_j$ ) : the set of attribute names that are common to both the  $\text{CONDITIONS}_i$  and  $\text{CONDITIONS}_j$ .

*shareout*( $r_i, r_j$ ) : the set of attribute names that are common to both the  $\text{ACTIONS}_i$  and  $\text{ACTIONS}_j$ .

*notshared*( $r_i, r_j$ ) : the set of all attributes used in  $r_i$  or  $r_j$  but not in *inout*( $r_i, r_j$ ), *sharein*( $r_i, r_j$ ), and *shareout*( $r_i, r_j$ ).

The counts of attributes of the four sets, generated by above four functions, are used to calculate the structural relatedness. And the weight of each count is given as a variable in our rule similarity calculation. Hence, the structural relatedness  $L(r_i, r_j)$  between two rules,  $r_i, r_j$ , can be formulated as (6.4).

$$L(r_i, r_j) = |inout(r_i, r_j)| \cdot w_{inout} + |sharein(r_i, r_j)| \cdot w_{sharein} + |shareout(r_i, r_j)| \cdot w_{shareout} - |notshared(r_i, r_j)| \cdot w_{notshared} \quad (6.4)$$

Where  $0 \leq w_{inout}, w_{sharein}, w_{shareout}, w_{notshared} \leq 1$  and  $w_{inout} + w_{sharein} + w_{shareout} + w_{notshared} = 1$ . For example, the  $w_{inout}, w_{sharein}, w_{shareout}$  and  $w_{notshared}$  could be set to 0.4, 0.2, 0.3, and 0.1, respectively.

## (ii) Semantic Relatedness

In some cases, rules are very similar in syntactic structure, but they may be used to deal with different problems. Considering only structural relatedness between rules cannot effectively distinguish them. As for the rules,  $r_3, r_4$ , and  $r_5$ , listed in Example 1 in Appendix C, structural relatedness between every pair of rules is the same. Even though the rules with only structural relatedness are used to detect different network attacks, no additional information can help separate those rules. Therefore, semantic relatedness is defined and used to complement structural relatedness for calculating rule similarity.

When considering structural relatedness between rules, only the names of attributes are taken into consideration instead of the values or the operators of attributes. In order to capture the semantic meaning between rules, attribute values and operators of the expressions are also considered. The expressions of rules can be

divided into two categories, categorical and numerical, according to the data type of values. For a given expression, if its value is categorical data, it belongs to categorical expression, e.g., (*protocol = TCP*); otherwise, it belongs to numerical expression, e.g., (*destination\_port > 1023*). The semantic relatedness calculations of these two types of expressions are different in our method.

Ontologies of knowledge-based system are often used for content explication or as common dictionary. The semantic relatedness between categorical expressions can be measured by the conceptual similarity between their values. That is, for two categorical values,  $x$  and  $y$ , the semantic similarity of two categorical values can be measured by conceptual similarity function  $s(x, y)$ , which depends on both the distance between them in the ontology and their generality. The conceptual similarity function between  $x$  and  $y$  is defined as (6.5).

$$s(x, y) = \begin{cases} \frac{c}{d(x, y) + \log_2(1 + D(x) + D(y))}, & \text{if } x, y \in \text{ontology} \\ \mathbf{d}(x, y) = \begin{cases} 1 & x=y \\ 0 & x \neq y \end{cases}, & \text{if } x, y \notin \text{ontology} \end{cases} \quad (6.5)$$

The  $d(x, y)$  is the number of “hops” between  $x$  and  $y$ ,  $D(x)$  is the number of all its descendants, and  $c$  is the boundary constant. If  $x$  and  $y$  are not located on the ontology, the Kronecker delta function  $\mathbf{d}(x, y)$  is used to determine their similarity.

Besides, different operators, e.g. “=” and “?”, may influence the evaluations of semantic relatedness between two expressions. Hence, two cases of operator combinations for two categorical expressions must be considered when calculating the semantic relatedness. Therefore, the semantic relatedness is formulated as (6.6).

$$\mathbf{a}(e_m, e_n) = \begin{cases} s(\text{value}_m, \text{value}_n) & , \text{if } \text{operator}_m = \text{operator}_n \\ 1 - s(\text{value}_m, \text{value}_n) & , \text{otherwise} \end{cases} \quad (6.6)$$

On the other hand, both expressions must be transformed into mathematical intervals before evaluating semantic relatedness between two numerical expressions. For instance, expression  $c_1 = (\text{port} > 1023)$  is transformed to  $(1023, \text{max}]$ , where max is the maximum value of  $V_{\text{port}}$ , which is the value range of “port”. The semantic relatedness is measured by  $\beta(e_m, e_n)$  based on the overlapping of two mathematical intervals, which is defined as (6.7).

$$\mathbf{b}(e_m, e_n) = |i_m \cap i_n| / |i_m \cup i_n| \quad (6.7)$$

The  $i_m \in V_{\text{attribute}_m}$  and  $i_n \in V_{\text{attribute}_n}$ .



Therefore, the semantic relatedness between each two expressions can be formally defined as (6.8).

$$S(e_m, e_n) = \begin{cases} \mathbf{a}(e_m, e_n), & \text{both } e_m \text{ and } e_n \text{ are categorical expressions} \\ \mathbf{b}(e_m, e_n), & \text{both } e_m \text{ and } e_n \text{ are numerical expressions} \\ 0, & \text{if } \text{attribute}_m \neq \text{attribute}_n \end{cases} \quad (6.8)$$

Based upon the semantic and structural relatedness defined for expressions of rules as (6.4) and (6.8), the definition of the similarity of two rules,  $r_i$  and  $r_j$ ,  $R(r_i, r_j)$  is thus given in Definition 3.

**Definition 3.** Rule similarity between rules.

Given two rules,  $r_i = (CONDITIONS_i, ACTIONS_i)$ ,  $r_j = (CONDITIONS_j, ACTIONS_j)$ ,

the rule similarity between  $r_i$  and  $r_j$  is defined as formula (6.9).

$$\begin{aligned}
R(r_i, r_j) = & \sum_{e_m \in CONDITION_i, e_n \in ACTION_j, attribute_m, attribute_n \in inout(r_i, r_j)} S(e_m, e_n) \cdot w_{inout} \\
+ & \sum_{e_m \in ACTION_i, e_n \in CONDITION_j, attribute_m, attribute_n \in inout(r_i, r_j)} S(e_m, e_n) \cdot w_{inout} \\
+ & \sum_{e_m \in CONDITION_i, e_n \in CONDITION_j, attribute_m, attribute_n \in sharin(r_i, r_j)} S(e_m, e_n) \cdot w_{sharein} \\
+ & \sum_{e_m \in ACTION_i, e_n \in ACTION_j, attribute_m, attribute_n \in ishareout(r_i, r_j)} S(e_m, e_n) \cdot w_{shareout} \\
- & |notshared(r_i, r_j)| \cdot w_{notshared}
\end{aligned} \tag{6.9}$$

### ■ Cluster Similarity

As mentioned before, Rule Base Partitioner iteratively merges the most similar rule clusters to construct the resulting rule clusters. Besides, in order to avoid too small or too large rule clusters generated, the quantity of rules in rule cluster is also considered when calculating the clusters. Therefore, similarity between two rule clusters,  $g_s$  and  $g_t$ , is defined as (6.10).

$$CS(g_s, g_t) = \sqrt{\sum_{r_i \in g_s, r_j \in g_t} R(r_i, r_j)} + \frac{2}{|g_s| + |g_t|} \tag{6.10}$$

Given the set of rule clusters, the similarity between each pair of rule clusters can be calculated in advance and stored in Cluster Similarity Matrix (CSM), an m-by-m upper triangular matrix, where m is the number of rule clusters. Each entry is the cluster similarity between rule clusters,  $g_s$  and  $g_t$ , that is,  $CS(g_s, g_t)$ .



## ■ Rule Base Partitioning Algorithm

Once the similarity for rule clusters can be calculated, the Rule Base Partitioning Algorithm is used to partition a rule base into rule clusters. The algorithm derives a high-level structure for the rule base based on the information of rule similarity. The stopping criterion is to stop when the cluster similarities between all pairs of rule clusters are no longer larger than a user defined similarity threshold ( $st$ ). The rule base partitioning algorithm is presented as Algorithm 6.2.

### Algorithm 6.2 Rule Base Partitioning Algorithm

**Input:** A set of rules, similarity threshold  $st$

**Output:** A set of rule clusters

**Step1:** Group each rule as a single rule cluster.

**Step2:** Generate  $CSM$  based upon the cluster result.

**Step3:** Choose the entry  $n_{ij}$  with the largest value (most similar) from the  $CSM$ .

**Step4:** Terminate and output the rule clusters, if  $n_{ij}$  is less than or equal to  $st$ .

**Step5:** Combine  $g_i$  and  $g_j$  into a cluster by merging the rules inside the clusters.

**Step6:** Repeat **Step 2** to **Step 5**.

## (2) Meta-rule Construction Process

The meta-rule construction is used to extract meta-rules from the partitioned rule clusters by Meta-rule Extractor which consists of Meta-Apriori Algorithm and Confidence Calculator. The Meta-Apriori algorithm is modified from Apriori algorithm [2] to generate the meta-rules, and Confidence Calculator calculates the confidence value of each meta-rule. The meta-rule generated by Meta-rule Extractor is then stored in the Meta-rule base for further usage.

## ■ Meta-Apriori Algorithm

The Meta-Apriori algorithm tries to discover the most frequent combinations of expressions to describe the rule cluster. The basic idea is that those most frequent combinations of expressions are used in many rules of the rule clusters, and once the combination is met, those rules may be related to the result. The transactions and itemsets defined in Meta-Apriori algorithm are rule conditions and expressions, which is given in Definition 4.

### Definition 4. Transaction and itemset.

Given a rule cluster  $g_i = \{r_{i1}, r_{i2}, \dots, r_{iN}\}$ , where  $N$  is the number of rules in  $g_i$ , the transaction and itemset are defined below:

$t_{ij} = \text{CONDITIONS}_{ij}$ , where  $\text{CONDITIONS}_{ij} = \bigwedge_{j \in [1 \dots N]} r_{ij}$ ,  $j \in [1 \dots N]$ , is a set of expressions to be used as one transaction, e.g.,  $t_{11} = \{(protocol = TCP), (protected\_network\_direction = A), (source\_port > 8080), (string = NetBus)\}$ .

$d$  is the itemset of a set of expressions.

In Meta-Apriori algorithm, the support count of the itemset is defined as the number of transactions that the itemset subsumes. That is, the set of expressions of the itemset subsume those of the transactions. For two expressions,  $e_m$  and  $e_n$ ,  $e_m$  subsumes  $e_n$  if  $sub(e_m, e_n) = 1$ , where  $sub()$  is called expression subsume function shown as (6.11).

$$sub(e_m, e_n) = \begin{cases} 1 & , \exists v \in V_{attribute_n}, v \in V_{attribute_m} \\ 0 & , \text{otherwise} \end{cases} \quad (6.11)$$

Moreover, an itemset subsumes the transaction if each expression of itemset subsumes at least one expression of transaction. The itemset subsumption is defined in Definition 5.

**Definition 5.** Itemset subsumption function.

Given an itemset  $d$  and a transaction  $t$ , the itemset subsumption function is defined as formula (6.12).

$$subsum(d,t) = \begin{cases} 1, & \forall e_m \in t \mid sub(e_m, e_n) = 1 \\ 0, & \text{otherwise} \end{cases} \quad (6.12)$$

After discussing the notations and subsumption issues, the complete Meta-Apriori algorithm is given as Algorithm 6.3.

### Algorithm 6.3 Meta Apriori Algorithm

**Input:** A set of transactions,  $T$ ; minimum support threshold,  $min\_sup$ .  
**Output:** A set of frequent itemset,  $D$ .

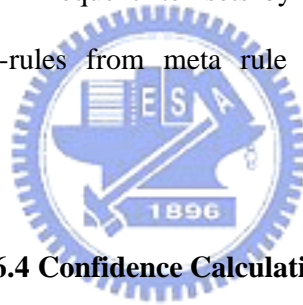
**Step1:** Generate the set of frequent 1-itemsets,  $D_1$ , by scanning  $T$ .  
**Step2:** Set initial value of  $k$  to 2.  
**Step3:** Generate candidate  $k$ -itemsets  $C_{ik}$  from  $D_{i(k-1)}$ .  
**Step4:** For each  $k$ -itemset  $d_k \in C_{ik}$ , compute the support count, that is,  $d_k.support = \sum_{t=1}^N subsum(d_k, t_i)$ .  
**Step5:** Remove those  $k$ -itemsets that their support counts are less than  $min\_sup * N$  from  $C_{ik}$ . **The remaining itemsets are stored in  $D_{ik}$ .**  
**Step6:** If  $D_{ik} \neq \{f\}$ , increase  $k$  and goto **Step 2**.

Moreover, an itemset subsumes the transaction if each expression of itemset subsumes at least one expression of transaction. The Example 6 in Appendix C shows the process of Meta-Apriori algorithm.

## ■ Meta-rule Generation

The meta-rule generation is based on the concept of constraint-based association mining [53]. The meaning of the meta-rule,  $mr_j = (CONDITIONS_j, (RULE\_CLUSTER = g_i), conf_j)$ , is that if all expressions of  $CONDITIONS_j$  are satisfied, the rule cluster  $g_i$  will be selected with confidence value,  $conf_j$ .

The frequent sets of expressions generated from one rule cluster may be the same with those generated from the other rule clusters. Therefore, once the frequent itemsets of all rule clusters are generated by Meta Apriori algorithm shown in Algorithm 6.4, the Confidence Calculator is used to calculate the confidence value of each meta-rule generated from frequent itemsets by accumulating the total support count of all selected meta-rules from meta rule base based upon Confidence Calculation Algorithm.



### Algorithm 6.4 Confidence Calculation Algorithm

**Input:** A set of meta-rules,  $MRB$ , without confidence values.

**Output:** A set of meta-rules,  $MRB'$ , with confidence values.

**Step1:** Set  $MRB'$  is an empty set.

**Step2:** For each meta-rule  $mr_j$  from  $MRB$  which have the same set of expressions in the  $CONDITION_j$ .

**Step3:** Accumulate the total support count of all selected meta-rules.

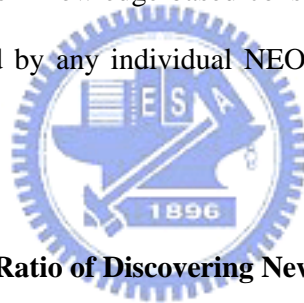
**Step4:** For each cluster  $g_i$ , set the confidence value of  $mr_j$  in  $g_i$  via dividing its support count by total support count.

**Step5:** Remove those selected meta-rules from  $MRB$  to  $MRB'$ .

## 6.5 Experiments

We implemented a *Dynamic EMCUD* web based system, and used computer worm as an experimental domain, where three computers equipped with the *Dynamic EMCUD* system constructed their own KBs to evaluate the performance of

discovering variants. As shown in Figure 6.2, we implemented a simple computer worm sample generator to generate the test samples of worms. The classification rules of 15 kinds of worm families including original worms and some variant worms (polymorphic worms) are extracted by domain experts using *EMCUD*, where the initial AOT value is given directly by experts and these worm classification rules are stored into knowledge base. To evaluate the effectiveness of *Dynamic EMCUD*, we generated 20 kinds of test samples including the behaviors of 15 original worm families and 5 new worm families to randomly attack the victims. The experimental result in Table 6.1 shows that the collaborative framework can successfully discover the variants by NEO-learning module. Since some critical weak embedded rules may be ignored in the beginning of knowledge based construction, some specific variants which can not be discovered by any individual NEO-sensor can be detected by the rules.



**Table 6.1 The Ratio of Discovering New Evolved Worm**

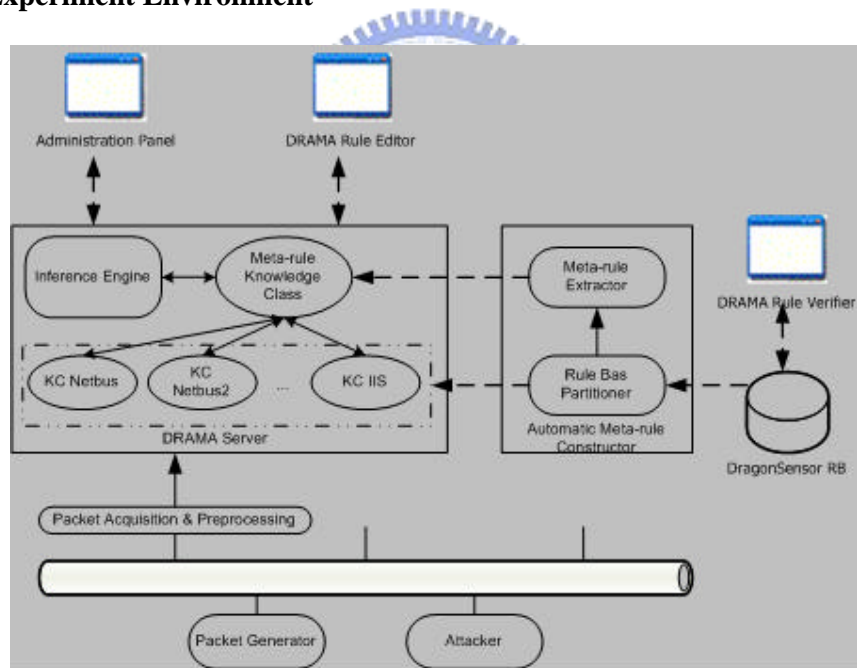
	Original Worms	Variant Worms	New Families	Inference Cost
<b>Dynamic EMCUD</b>	100%	85%	80%	193
<b>Collaborative</b>	100%	92%	80%	200

Both *Dynamic EMCUD* and Collaborative framework can detect the 100% of original worms since the classification rules are stored in worm KB. For detecting the occurrence of variant worms, the *Dynamic EMCUD* can learn the 85% of variant worms. If the *Dynamic EMCUD* is extended to the collaborative framework, 92% of variant worms can be discovered due to the highly complementary configurations between collaborative NEO-sensors. However, both *Dynamic EMCUD* and collaborative can detect the 80% of new worm families in our experiments. Our

collaborative framework can learn the 92% of the variants, it needs only 7 extra inference costs to reach the goal in our experiments. However, the significant difference of new family can not be discovered easily. Thus, deploying the more complementary configurations between collaborative sensors could be efficient in discovering the knowledge of new worms.

Moreover, an Intrusion Detection System (IDS) prototype based upon *RP-MES* is proposed, and the partitioning result as well as performance analysis of the prototype system is also introduced.

### 6.5.1 Experiment Environment

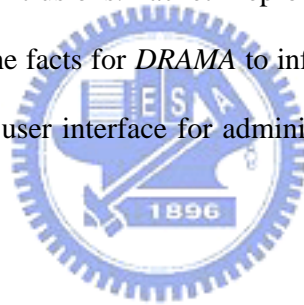


**Figure 6.9 An IDS Prototype System Based *RP-MES***

Figure 6.9 illustrates the system architecture of the IDS prototype system including the *DRAMA* System [46] and Automatic Meta-rule Constructor. *DRAMA* is used to store, represent, process the knowledge of the IDS. Moreover, the *NORM*

knowledge model used in *DRAMA* provides the ability of inferring meta-rules and rule selection by the ACQUIRE knowledge relation, which is a dynamic knowledge relation to include rule cluster only when meta-rule is matched. *DRAMA* Rule Verifier is used to verify rules within the knowledge classes.

Automatic Meta-rule Constructor is also implemented in JAVA. The knowledge can be the rule base of any other IDS, e.g., Snort [68], Dragon Sensor [24], etc. Automatic Meta-rule Constructor can partition the rule base into rule clusters, which are represented as knowledge classes in *DRAMA*, and generate meta-rules from those knowledge classes. Those generated knowledge classes can be processed by *DRAMA* Server for detecting network intrusions. Packet Preprocessor collects the packets from network and translates into the facts for *DRAMA* to infer. Administration Panel of the prototype system provides a user interface for administrator to monitor the situation of network environment.



### **6.5.2 Experimental Results**

The rule base of Dragon Sensor consists of 646 rules, which is used as our experimental knowledge source. In the first experiment, various numbers of rules are partitioned by Automatic Meta-rule Constructor according to different similarity threshold (st) settings, which are used as the criteria to stop clustering process. The partitioning result is shown in Table 6.2. It can be observed that similarity threshold = 1.2 produces more reasonable number of rule clusters since the average size of rule clusters is satisfied with Miller's magic number [49], which says that human beings have a meaningful chunking size up to  $7 \pm 2$ .

**Table 6.2 The Cluster Number with Different Similarity Threshold Settings and Number of Rules**

Rules \ st	1.0	1.1	1.2	1.3
200	3	5	28	48
300	3	5	32	82
400	3	7	52	112
500	3	5	57	139
600	4	7	56	172
646	3	6	59	188

The experiment is conducted to analyze the accuracy according to different similarity threshold (st) settings, the criteria to stop clustering process. For each partitioning result with different cluster similarity setting, Automatic Meta-rule Constructor also generates corresponding meta-rules and stores those meta-rules in a knowledge class. Those knowledge classes are loaded into the IDS prototype system for accuracy experiment. The  $accuracy = |F_p| / |F_{DS}|$  is obtained by comparing the result of original rule base and partitioned rule base. The  $F_p$  is the set of rules fired in partitioned rule base, and  $F_{DS}$  is the set of rules fired in original rule base. The result is shown in Table 6.3.

**Table 6.3 Accuracy Comparisons**

Rules \ st	1.0	1.1	1.2	1.3
200	98%	99%	100%	100%
300	98.33%	100%	100%	100%
400	96.25%	96.75%	98.75%	99.25%
500	94.20%	94.20%	97%	97.60%
600	95.33%	95.67%	97.33%	98.00%
646	95.51%	95.82%	97.37%	98.14%

We also use the traditional rule base partitioning approach, which only considers structural relatedness of rule, to partition the same rule base used in previous

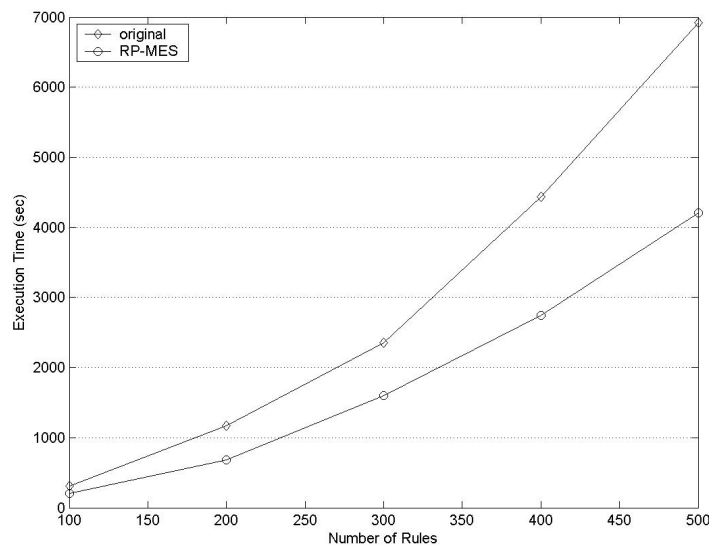


experiments. Table 6.4 shows the number of rule clusters partitioned for both approaches to meet the same accuracy (the accuracy for *RP-MES* approach with  $st = 1.2$ ).

**Table 6.4 Comparison of Number of Clusters**

Rules	200	300	400	500	600	646
RP-MES	28	32	52	57	56	59
TRADITIONAL	51	85	113	139	170	188

The final experiment is to evaluate the performance between the rule base not partitioned and the rule base partitioned by *RP-MES*. The result of the experiment is shown in Figure 6.11.



**Figure 6.11 The Performance Comparison**

When the number of rules is small, the performance difference is not obvious. But if the number of rules becomes large, the execution time of original rule base increases greatly. However, the execution time of *RP-MES* increases more smoothly than that of original rule base.

# Chapter 7

## Application in Alert Classification Model

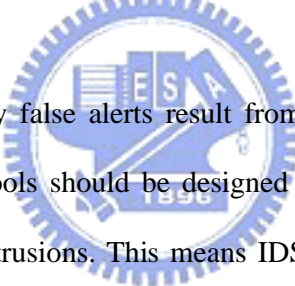
### Construction

#### 7.1 Introduction

In order to detect and prevent anomaly network behaviors, many Intrusion Detection Systems (IDS) or Firewalls have been developed to focus on well-known intrusion patterns through packet-based information, connection-base information, or some statistical network information [29][32][47][52][55][56]. Although these kinds of approaches can be useful to defend the obvious activity patterns of intrusions, many intrusions are still hard to be detected by IDSs to notice human experts because of numerous noises and insufficient information among different intrusions. In other words, existing IDS tools just tag suspicious network behaviors into IDS alerts, but can not avoid generating false alerts. For domain experts, it is time-consuming to classify IDS alerts into true or false alert sequences precisely due to numerous noisy IDS alerts; for junior administrators, it is difficult to generate aggregated IDS alert sequences according to the similarities of intrusion patterns due to the lack of domain knowledge.

Although several methods [4][18][60][66][78] such as generic algorithm, neural networks, and data mining approaches, have been used to discover either unknown or

useful patterns for experts, lots of hidden and concealed intrusion patterns may still be escaped because of insufficient and dirty information. None of them discusses on discovering intrusion patterns thoroughly from IDS information data, called IDS alerts. Therefore, we are concerned with how to design a systematic framework to assist administrators discovering intrusion patterns with IDS alerts. Our idea is to construct a decision support system to help experts construct an alert classification model for on-line intrusion detection of IDS alerts. For domain experts, the built alert classification model will reduce experts' efforts on how to precisely and quickly fix the root causes; for junior administrators, alert classification model will help them reuse the embedded domain expertise as references, and the domain knowledge will be no longer a limitation for intrusion detection.



As we know, too many false alerts result from IDS tools in present network environments because IDS tools should be designed as powerful as possible not to miss any detection of real intrusions. This means IDS tools become more and more sensitive to generate false alerts which are noise to discover real intrusion patterns in some network environments. In this chapter, we collect alert transactions in an attack-free environment, which is a virtual intranet with several hosts in laboratories to simulate real Internet behaviors in the training stage. Accordingly, all alerts are treated as false alerts in this virtual intranet, and the frequent patterns to be mined in this case are treated as patterns of normal behaviors, or called patterns of false alert. In other words, we can use these data to construct normal behavior patterns to remove false alerts. Besides, using some specific rootkits to simulate the real intrusions of networks, we can also construct known intrusion patterns in the training stage to classify existing intrusion patterns.

The proposed decision support system consists of three phases in the training stage: Alert Preprocessing Phase, Model Constructing Phase and Rule Refining Phase to assist administrators construct three kinds of rule classes (normal rule class, intrusion rule class and suspicious rule class) to remove false alert patterns and analyze each existing or unknown alert pattern, where each rule class represents a set of classification rules. Because flags of alert patterns may change between two consecutive time intervals, so the differences of specific patterns must be highlighted and refreshed again to experts in each time interval. A least recently used (*LRU*) rule replacement policy is used to replace the rules which are less used recently in each classification rule class to ensure the performance of our system.

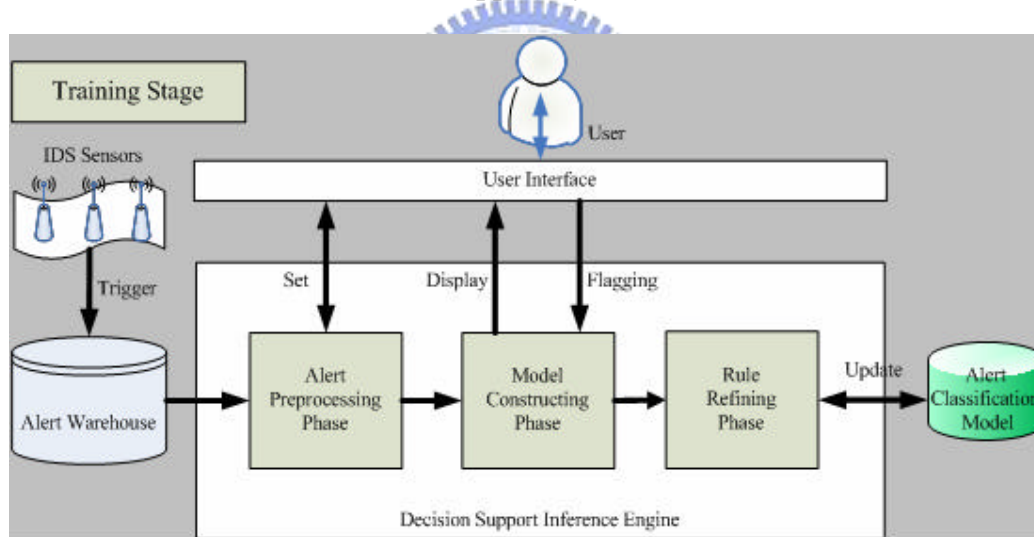
## **7.2 Decision Support System Architecture**

Although many IDS have been proposed to assist administrators in detecting intrusion, false alarms are still huge and result in the difficulty of analysis. Traditional intrusion pattern analysis methods use different data sources with their own data formats according to different methods of IDS alert analysis. Different analysis methods have different characteristics to get the desired intrusion patterns. However, most of these researches are conceptual deficient and mutually independent; some of them provide sufficient data formats, and some others are conspicuous on analysis performance. Each intrusion pattern analysis method has its own limitation, so integrating advantages of different methods seems better than redesigning a new analysis framework to take the advantage of these methods. Hence, we propose a decision support system for constructing alert classification behavior patterns to help experts easily to construct an alert classification model using the huge amount alerts.

The main purposes of IDS alerts collection and analysis are finding more meaningful alert information and discovering the information relation between real alerts to verify system vulnerabilities and to infer attack causes. Some issues are derived from these purposes:

- (1) How to choose appropriate analysis targets and data formats.
- (2) How to filter false alerts efficiently.
- (3) How to discover attack patterns and display appropriate data types for administrators to make policies.

### 7.2.1 The Framework of Decision Support System



**Figure 7.1 The Framework of Decision Support System**

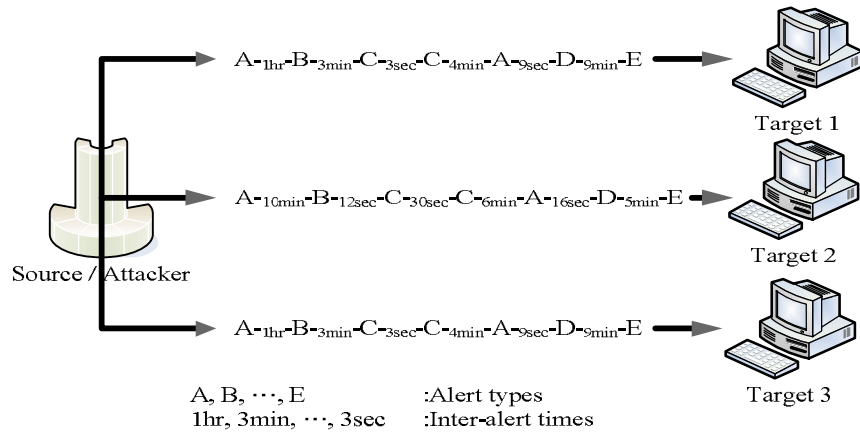
In the training stage, a decision support system for constructing an alert classification model consists of alert preprocessing phase, model constricting phase, and rule refining phase as shown in Figure 7.1. As we know, most attack patterns are a sequence of actions, which can be represented as a sequence of IDS alerts. In alert

preprocessing phase, all alerts triggered by IDS sensors are stored in the alert warehouse and would be transformed into alert sequences for each IDS sensor during a period of time which could be set by experts for batch training. Without alert preprocessing, the time complexity of pattern analyzing will become huge and the accuracy of pattern discovering will low down due to the huge amount of false alerts. In model constructing phase, filtering and analysis methods are proposed to assist experts construct different classification rule classes to remove false alert patterns and analyze each existing or novel alert pattern. The normal alert behavior patterns will be trained firstly in an attack-free environment by sequential miming algorithm and used to reduce the affect of noise on intrusion patterns as more as possible because the normal behavior pattern will occur periodically and frequently. Then, the suspicious behavior patterns will be trained in a simulated attacking environment and the corresponding classification labels could be flagged by experts according to the alert patterns in model constructing phase. These obtained behavior patterns could be used to help administrator identify intrusions in the on-line stage. If new alert behavior patterns are discovered, they will be integrated into the alert classification model; otherwise, the alert model could be refined in rule refining phase.

### **7.2.2 Alert Preprocessing Phase**

Since most part of present attacks are target-specific and stealthy intrusions instead of large-scale violence [72] and alert transactions in alert warehouse are large, it is necessary to transform raw alert transactions into pre-defined alert transaction formats for further analyzing. Some characteristics of attack tools can be discovered by analyzing alert sequences [37]. As shown in Figure 7.2, assume a source host triggers the same sequence of alerts against different target hosts in a noise-free

environment without false alerts. In general, it can be easily seen these scenarios from an attacker try out his or her attack tool against different targets.



**Figure 7.2 An Attack Tool Being Run Against Three Targets**

Alert sequences can be represented as specific characteristics of specific attack tools or attackers, so we select alert sequence as our target data format. We collect IDS alert transactions from many IDS sensors as our training data sources, and set an appropriate time period *Batch\_Time\_Window* for batch preprocessing. The value of *Batch\_Time\_Window* may be an hour, a day, a week, or a month; but we suggest that to set the value of *Batch\_Time\_Window* as one day will be better in our experience. Besides, most intrusions will not continue for a very long time in its own attack lifecycle, so we set a short-term time period *Short\_Time\_Window* as the time period of our alert sequence transactions. We assume that every intrusion will finish its whole lifecycle in this short-term time period. The value of *Short\_Time\_Window* may be 1 minute, several minutes, or even an hour; but we suggest that we set the value of *Short\_Time\_Window* as half of an hour will be better in our experience.

After explaining the definition of *Batch\_Time\_Window* and *Short\_Time\_Window*,

there are still some issues for alert transaction construction. First, for a single alert transaction, it is difficult to classify it into true alert or false alert because some alerts are used to be triggered. Second, it is still not appropriate for alert sequences to be too long because long sequential patterns are not easy to be frequent. Our idea is to design proper policies corresponding to different environments, and some policies of alert sequence partition are proposed to construct alert sequence transactions for model constructing phase with different requirements. These policies are used to partition alert sequences into subsequences in each *Short\_Time\_Window*. According to different purposes, the corresponding policies are designed to obtain desired alert subsequence transactions. Three policies are shown as follows.

#### **CASE 1: Left-To-Right Non-Repeat Policy**

**Step 1:** Scan every alert from the first (left) to the end (right) of sequence.

**Step 2:** Partition sequence into scanned part and unscanned part; IF next alert to be scanned is equal to some alert in the scanned part, Do partition from the first element to the present element and set the unscanned part as a new sequence.

**Step 3:** IF there is still any element in the unscanned part, GOTO **Step 1**.

#### **CASE 2: Right-To-Left Non-Repeat Policy**

**Step 1:** Scan every alert from the end (right) to the first (left) of sequence.

**Step 2:** Partition sequence into scanned part and unscanned part; IF next alert to be scanned is equal to some alert in the scanned part, Do partition from the first scanned element to the present element and set the unscanned part as a new sequence.



**Step 3:** IF there is still any element in the unscanned part, GOTO **Step 1**.

### CASE 3: Equi-Length Policy

**Step 1:** Ask administrators to set a value of the subsequence length.

**Step 2:** Partition each alert sequence into several subsequences with the fixed length.

Here ‘Left-To-Right Non-Repeat Policy’ is used as our partition policy for example. We suppose that there is an alert sequence of sensor H1 in short-term slice  $t1$  as following:

Sensor ID	<i>Short_Time_Window(1)</i>
H1	XABYXCYC

At first, let AS[7] be alert sequence: XABYXCYC; because AS[4] equals AS[0], so this sequence is divided into two alert subsequences: the scanned part XABY and the unscanned part XCYC, and executes partitioning again in the unscanned part XCYC as new sequence, or consider as new AS[4]. In AS[4], we can find that AS[4] equals AS[2] again, so this sequence AS[4] is divided into two alert subsequences: the scanned part XCY and the unscanned part C. After whole original alert sequence being scanned, we can get three new alert sub-sequence transactions such as XABY, XCY and C as follows.

Sensor ID	<i>Short_Time_Windows(1)</i>		
H1	XABY	XCY	C

### 7.2.3 Model Constructing Phase

There are some researches discussing about how to filter false alerts efficiently, and different data characteristics and different filtering heuristics brings quite different

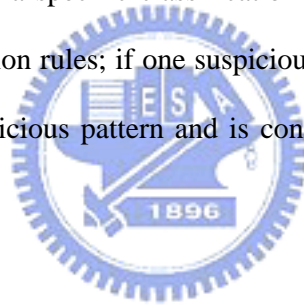
filtering results. Generally speaking, most of these researches use specific analysis methods or compile expert experiences to construct filter models, and use this filter model to discard those highly-possible false alerts to get clearer data.

Besides, to find specific patterns in these kinds of numerous sources is like to discover meaningful patterns in distributed databases or data warehouses for decision support. Many researches have been proposed to analyze behavior models in databases, and different alert pattern analysis methods with different data source formats cause different outcomes. In our thought, none of these analysis algorithms is powerful enough for correctly detecting intrusions.

There are two different purposes for alert pattern discovering obviously: false alert filtering and useful alert pattern discovering. It is common to filter false alerts before alert pattern discovering because these noisy alerts will affect the accuracy of alert pattern discovering. In other words, our approach tries to discover patterns of false alerts first, and then continue to discover useful patterns of suspicious or known alerts.

In the training stage, we design some rule class construction methods to build specific behavior rule classes. We construct three types of behavior classification rule classes with domain experts: normal behavior rule class, intrusion behavior rule class, and suspicious behavior rule class. Each type of rule classes is constructed by individual methods, which use different data sources as their data inputs.

For example, if we want to construct normal behavior rule class which is the collection of false alert patterns, we may collect IDS alerts in an attack-free network environment and use specific normal behavior rule class construction method to discover false alert sequences. Besides, if we want to construct intrusion rule class and suspicious rule class, we may collect IDS alerts in a simulated network environment using some existing intrusion tools to generate simulated intrusions, to discover interim suspicious behavior rule class by specific rule class construction method. We will provide all interim classification rules of intrusion/suspicious behavior rule classes to experts, and the system will interact with them to flag these suspicious alert sequences with specific tags. If one suspicious pattern is a known attack, experts will flag it as a specific classification name and seem it as a kind of intrusion behavior classification rules; if one suspicious pattern is never verified, then it is really an unknown suspicious pattern and is considered as a kind of suspicious behavior classification rules.



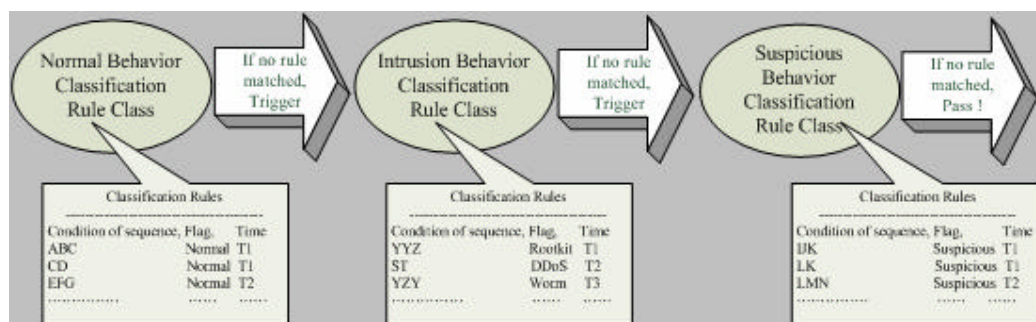
The procedure of behavior classification rule class construction in the training stage is shown as following:

- (1) Construct Normal Behavior Classification Rule Class.
- (2) Construct Suspicious/Intrusion Classification Rule Class.
- (3) Classify Classification Rules of Suspicious/Intrusion Rule Class into Suspicious Behavior Rule class and Intrusion Behavior Rule Class.

These behavior classification rule classes are used to monitor IDS alert behaviors in the on-line stage. IDS sensors trigger alerts at any moment, and it is not easy for experts to classify these consecutive alert sequences into normal behaviors or

suspicious behaviors without our decision support system. With these three types of rule classes, it is possible for real-time monitoring IDS alert sequences to discover most intrusion behaviors. The decision support system lightens the load of experts in analyzing IDS alerts and assist them focusing on how to explain the intrusion principles and how to fix the root causes. The rule class construction algorithms are detailedly described in Appendix D.

The activation flow of each rule class is shown as follows: in the on-line stage, alert sequences are triggered by IDS tools and will be forwarded to alert warehouse. Before being stored into alert warehouse, each alert sequence will be matched by classification rule classes of classification model first. After interviewing with domain experts, the normal behavior patterns will be firstly used to filter out most part of false alerts. Then, the intrusion behavior patterns will be used to detect the true intrusion alerts. Finally, the remaining alerts should be classified into known suspicious behavior patterns or unknown suspicious alerts.



**Figure 7.3 Meta-rules of Classification Rule Classes for On-line Monitoring**

The activation flow of each rule class is shown in Figure 7.3. If there is any alert subsequence matching the classification rules in normal behavior classification rule

class, it will be filtered out immediately; if there is no alert subsequence matching, classification rules of intrusion behavior classification rule class will be triggered to verify these on-line alert sequences. In the similar way, if there is any alert subsequence matching the classification rules of intrusion behavior classification rule class, system will highlight the alert subsequence to notice experts; if not, classification rules of suspicious behavior classification rule class will be triggered to verify these alert sequences again. If there is still no classification rule matched by alert subsequences, these alert sequences will be stored in alert warehouse as new batch data sources of the next time window.

#### **7.2.4 Rule Refining Phase**

These obtained behavior rule classes could be used to help administrator identify intrusions in the on-line stage. If new alert behavior classification rules are discovered, they will be integrated into the alert classification rule classes; otherwise, the alert rule classes could be refined in the rule refining phase.

First, for the efficiency of rule inference, the maximum number of classification rule in each classification rule class needs to be limited. If new classification rules are incrementally added into rule classes, the inference performance of on-line monitoring will be decreased due to the huge of rules. The initial number of maximum rule number of each classification rule class in our experiment is set to 200.

If the quotas of classification rule class are full with classification rules, it is necessary to find replacement strategies for rule replacement. It is possible for each classification rule class to use different rule replacement policies to implement rule

replacement according to individual data characteristics. In the field of operating system, the *LRU* policy is often used as a page-replacement algorithm and is considered to be quite good. In this paper, a *least recently used (LRU)* replacement policy is used in each classification rule class. *LRU* replacement associates with each rule the time of that rule's last use. When a rule must be replaced, *LRU* chooses that rule that has not been matched for the longest period of time. This strategy is the optimal rule-replacement policy looking backward in time, rather than forward.

For the rule contradiction checking issue, two classification rules with the same alert sequence may be flagged in different classification labels (e.g., normal or intrusion) in two consecutive batch interactions. For example, in one time interval, alert sequence ABC are flagged as “normal”, but ABC are flagged as “Suspicious” in the next time interval. In this case, system will interact with experts to make sure the flag of this alert sequence instead of simple and fixed replacement.

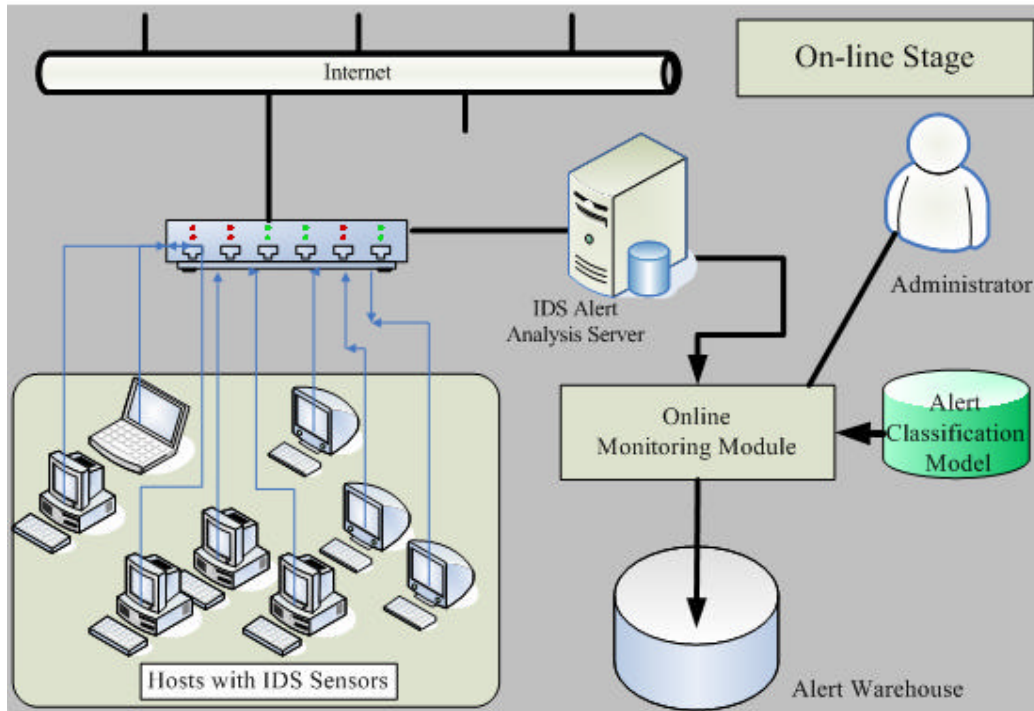
### **7.3 Experiments**

In order to evaluate the efficiency of the obtained behavior pattern, we set up an experimental environment equipped with one server and eight hosts to simulate real network.

#### **7.3.1 The Design of The Experimental Environment**

The knowledge-based architecture of collaborative discovering of suspicious network behaviors is implemented as shown in Figure 7.7. All the related tools described above are one server, which plays the role of IDS Alert Analysis Server, including IDS center for alert warehouse, web-based analysis console and alert analysis console. Besides, eight hosts all play the role of IDS sensors to trigger alerts

as our data sources. The system and network profiles of these sensors are shown in Figure 7.4.



**Figure 7.4 Decision Support System Prototype in Experiments**

We have conceptualized alerts according to Snort rule set, which is a network-based IDS whose alerts are triggered by a collection of signature-based rules. Each Snort rule is composed of a Snort identification number, a message that is included in the alert when the rule is triggered, an attack signature, and references to sources of information about the attack. Each alert is provided with an identifier, time and data, sensor identifier, triggered signature, IP and TCP headers and payload. These alerts will be stored in the relational database as our alert warehouse. Alerts in one period of time, e.g., 24 hours, are collected by IDS center as data source in this experiment. For easy reading, we replace the original alert signature names with

different capital letters.

The training data set collection in our experiment is shown as follows. The total number of IDS alerts is 164 collected in a day and the Short\_Time\_Window is set to 1 hour. The total number of alert sequence transactions is 295.

<b>Total number of IDS alerts</b>	<b>Batch_Time_Window</b>	<b>Short_Time_Window</b>	<b>Total number of alert sequence transactions</b>
1640	24 hours	1 hour	295

### 7.3.2 Experimental Results

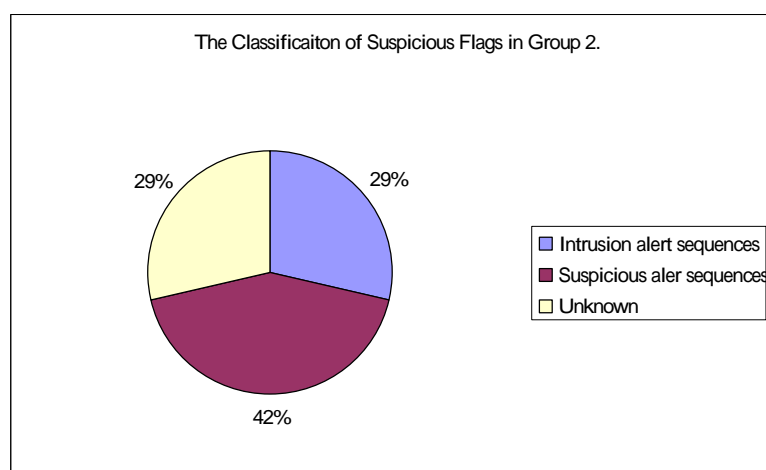
To verify the feasibility of our decision support system, we use the data set to execute our alert classification model. We construct an alert classification model in the training stage, and use this alert classification model for on-line monitoring. After filtering false alerts with normal behavior classification rule class, the result of alert reduction is shown in Figure 7.5. The classification rules of normal behavior classification rule class which are used in this experiment would change their time flags for rule refining.



**Figure 7.5 Alert Reduction Rate of Normal Behavior Classification Model**



The filtered alert sequences are used to discover suspicious/intrusion alert sequences. We used pre-defined suspicious alert classification rule class and intrusion alert classification rule class to discover useful alert patterns for experts. Suspicious alert classification rule class and intrusion alert classification rule class are used to verify the on-line alert sequences. The result of suspicious/intrusion alert sequence verification is shown in Figure 7.6. 29% of candidate IDS alert sequences are known alert sequences, 42% of candidate IDS alert sequences are suspicious alert sequences, and 29% of candidate IDS alert sequences are unknown alert sequences. The alert sequences triggered as suspicious alert sequences or intrusion alert sequences would be highlighted to notice experts for on-line detection, and these used alert classification rules would change their time flags for rule refining. The part of unknown alert sequences would be forward to alert warehouse as the data sources of next batch model construction in the training stage.



**Figure 7.6 Observations of Percentages of Different Suspicious Flags**

In this chapter, we proposed a decision support system for constructing an alert classification model, which consists of three phases: alert preprocessing phase, model constructing phase and rule refining phase. In alert preprocessing phase, raw IDS alerts are collected into alert warehouse and will be transformed into alert transactions. Three kinds of alert classification rule classes including normal behavior classification rule class, intrusion behavior classification rule class and suspicious behavior classification rule class, are constructed in model constructing phase to filter normal alert patterns and then discover each known or novel alert pattern based upon the remaining alert transactions. Each classification rule class consists of fixed number of classification rules. An AprioriAll-like sequential pattern mining algorithm is proposed to construct classification rules of normal behavior classification rule class. A set of intrusion tools to collect data sources of suspicious/intrusion behavior classification rule class construction in the simulated attacking network environment. Our experiment demonstrates that the accuracy of our decision support system is well. This decision support system will construct a classification model for on-line monitoring. The alert classification model is useful for experts to discover suspicious or intrusion patterns quickly and precisely, and lightens the load of on-line alert analysis for experts obviously. The current implementation of our research constructs rule classes in the alert classification model. In the near future, more types of classification rule classes will be created for enhancing the performance of detection.

## Chapter 8

### Conclusion and Future Work

In this dissertation, we proposed new knowledge acquisition methodologies, *Dynamic EMCUD* which is an iteratively knowledge acquisition method to monitor the inference behaviors of weak embedded rules, including variant knowledge acquisition (*VODKA*), and evolutionary knowledge acquisition (*TEA*) for dynamic knowledge by collecting these sufficient contexts to notify experts the occurrence of evolved objects. *VODKA* is proposed to iteratively discover the variants knowledge through observing the frequent inference behaviors of those weak embedded rules with marginally acceptable CF to assist domain experts to single out ambiguous objects. Three recommendations, including no change, changing the data type of an attribute, or adding a new attribute, are proposed to help them easily discover the new variants according to the learned large itemsets.

Moreover, *TEA* is also proposed to learn the evolutionary knowledge which is evolved with the changing environment by adjusting the AOT value over times based upon the trend of behaviors. Two methods are applied to trace the evolutionary knowledge and calculate the suitable CF value of each discovered rules: entropy based calculating and gracefully accumulating, to make the knowledge more adaptive for the current environment. Based upon the acquisition table increment of variant knowledge generated by *VODKA* and AOT increment of evolutionary knowledge

generated by *TEA*, we proposed *Grid Merging* algorithm to integrate the acquisition table increment into the main acquisition table for adapting the weak embedded rules to archive the knowledge evolution.

Moreover, a collaborative knowledge acquisition framework is also proposed to integrate the new knowledge generated from local *KBSs* and help experts easily discover the new evolved knowledge which are unseen in every local *KBSs* based upon the *Dynamic EMCUD* and the designed context which is designed to describe the static profile and dynamic behaviors of individual and environment. Consequently, six collaborative heuristics are proposed to help experts be aware of the occurrence of dynamic knowledge with the changing environment as time goes on.

Two applications including in worms and DDoS detection, and alert classification model construction are used to evaluate the performance of *Dynamic EMCUD*. Since the knowledge base can be evolved as time goes on, the evolutionary knowledge base can become huge and hard to maintain. We proposed *RP-MES* to solve the issues by designing a new approach combining both rule base partitioning and meta-rule construction mechanisms. As for rule base partitioning, *RP-MES* not only takes care of the structural relatedness between rules, but also considers the semantic relatedness of rules in the rule base. Based upon the clustering result, we can easily extract the meta rule of each rule cluster using meta-Apriori algorithm for improve the usage of knowledge base system. A *Worm Immune Service Expert* system (*WISE*) based upon *DRAMA* has been implemented and deployed in an experimental environment to evaluate the performance of our collaborative knowledge acquisition methodologies. The results show that new variants can be discovered after the

occurrence of a series of worm instances.

In the near future, we will improve *WISE* system with a collaborative knowledge acquisition ability to evaluate our idea by developing compatible pre-specified ontology of worms related domain knowledge. More robust knowledge acquisition methods for acquiring dynamic knowledge will be designed. More applications such as intrusion detection, ubiquitous learning will be also considered to evaluate the performance of collaborative knowledge acquisition methodologies.



## Reference

- [1] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large database," in *Proceedings of the ACM SIGMOD Conference*, 1993.
- [2] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proceedings of International Conference on Very Large Data Bases (VLDB'94)*, pp. 487-499, 1994.
- [3] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Proceedings of International Conference on Data Engineering (ICDE)*, pp. 3-14, 1996.
- [4] A. Alharby and H. Imai, "IDS false alarm reduction using continuous and discontinuous patterns," in *Proceedings of ACNS 2005*, pp.192-205, 2005.
- [5] J. Barlow and W. Thrower, "TFN2K – An analysis by Jason Barlow and Woody Thrower." <http://www2.axent.com/swat/swat.htm>, 2001.
- [6] Basic Analysis and Security Engine (BASE), <http://secureideas.sourceforge.net/>, 2005.
- [7] K. Boegl. *Design and implementation of a web-based knowledge acquisition toolkit for medical expert consultation systems*. Doctorial thesis, Technical University of Vienna, Austria, 1997.
- [8] J. H. Boose, "Personal construct theory and the transfer of human expertise," in *Proceedings of AAAI Conference*, California, 1984.
- [9] J. H. Boose, "A knowledge acquisition program for expert systems based on personal construct psychology," *International Journal of Man-Machine Studies*, Vol. 23, 1985.
- [10] J. H. Boose and J. M. Bradshaw, "NeoETS: Capturing expert system knowledge in hierarchical rating grids," *IEEE Expert System in Government Symposium*, 1986.
- [11] J. H. Boose and J. M. Bradshaw, "Expertise transfer and complex problems: using AQUINAS as a knowledge-acquisition workbench for knowledge-based systems," *International Journal of Man-Machine Studies*, Vol. 26, No. 1, pp. 3-28, 1987.
- [12] T. Bridis, "Powerful attack cripples majority of key Internet computers," *Yahoo! News*, Oct. 22, 2002.

- [13] J. B. D. Cabreraa, L. Lewis, X. Qin, W. Lee, R. K. Prasanth, B. Ravichandran, and R. K. Mehra, "Proactive detection of distributed denial of service attacks using MIB traffic variables - A feasibility study," in *Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management*, pp. 609-622, 2001.
- [14] O. Cairo, "KAMET: A comprehensive methodology for knowledge acquisition from multiple knowledge sources," *Expert Systems with Applications*, Vol. 14, No. 1, pp. 1-16, 1998.
- [15] J. J. Castro-Schez, N. R. Jennings, X. D. Luo, and N. R. Shadbolt, "Acquiring domain knowledge for negotiating agents: a case of study," *International Journal of Human-Computer Studies*, Vol.64, pp. 3-31, 2004.
- [16] CERT Coordination Center. "DDoS attacks," <http://www.cert.org>, 2003.
- [17] K. C. Chang, "Defending against flooding-based distributed denial of service attacks: A tutorial," *IEEE Communications Magazine*, Vol. 40, Iss. 10, 2002.
- [18] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang, "NiagaraCQ: A scalable continuous query system for internet databases," in *Proceedings of ACM SIGMOD 2000*, pp.379-390, 2000.
- [19] CORETECH Inc., *DRAMA Expert System*, [http://www.coretech.com.tw/c\\_DRAMA.htm](http://www.coretech.com.tw/c_DRAMA.htm), 2006.
- [20] P. J. Criscuolo, "Distributed denial of service-Trin00, Tribe Flood Network, Tribe Flood Network 2000, and Stacheldraht," *Technical Report CIAC-2319, Department of Energy - Computer Incident Advisory Capability*, 2000.
- [21] P. Crowther and J. Hartnett, "Using repertory grids for knowledge acquisition for spatial expert system," in *Proceedings of Australia and New Zealand Conference on Intelligent Information Systems*, Adelaide, SA, Australia, Nov. 18-20, pp.14-17. 1996.
- [22] D. Dittrich, (2000) "DDoS: Is there really a threat?" in *Proceedings of USENIX Security Symposium*, Aug. 16, 2000.
- [23] A. K. Dixit and R. S. Pindyck, *Investment under uncertainty*, Princeton University Press, 1994.
- [24] Dragon Sensor, Enterasys Networks, Inc. <http://www.enterasys.com>, 2004.
- [25] E. A. Feigenbaum, *Knowledge Engineer: The applied Side of Artificial Intelligence*, Stanford, California: Stanford University, 1980.
- [26] B. R. Gaines, "An overview of knowledge-acquisition and transfer,"

- International Journal of Man-Machine Studies*, Vol. 26, pp. 453-472, 1987.
- [27] L. Garber, "Denial-of-Service attacks rip the Internet," *IEEE Computer*, Vol. 33, Iss. 4, pp.12-17, 2000.
- [28] A. Ginsberg, S. M. Weiss, and P. Politakis, "Automatic knowledge base refinement for classification systems," *Artificial Intelligence*, Vol. 35, No. 2, pp. 197-226,1988.
- [29] R. P. Goldman, W. Heimerdinger, S. A. Harp, C. W. Geib, V. Thomas, and R. L. Carter, "Information modeling for intrusion report aggregation," in *Proceedings of DARPA Information Survivability Conference and Exposition II*, pp. 115-137, 2001.
- [30] S. T. Gruber, *The acquisition of strategic knowledge*, Academic Press Inc, 1988.
- [31] S. Hirasawa and W. W. Chu, "Knowledge acquisition from documents with both fixed and free formats," in *Proceedings of IEEE Internatoinal Conference on Systems, Man and Cybernetics 2003*, Vol. 5 , pp. 4694 -4699, 2003.
- [32] W. Y. Hsin, S. S. Tseng, S. C. Lin, "A study of alert-based collaborative defense," in *Proceedings of The 8th International Symposium on Parallel Architectures, Algorithms & Networks (ISPAN 2005)*, Las Vegas, Nevada, U.S.A., pp. 148-153, 2005.
- [33] G. J. Hwang, "Knowledge acquisition for fuzzy expert systems," *International Journal of Intelligent Systems*, Vol. 10, 541-560, 1995.
- [34] 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, 431-451, 1990.
- [35] G. J. Hwang and Tseng, "On building a medical diagnostic system of acute exanthema," *Journal of Chinese Institute of Engineers*, Vol. 14, No. 2, 185-195, 1991.
- [36] R. J. K. Jacob and J. N. Froscher, "A software engineering methodology for rule-based systems," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 2, 1990, pp. 173-189, 1990.
- [37] K. Julisch and M. Dacier, "Mining intrusion detection alarms for actionable knowledge, in *Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining*, pp. 366-375, 2002.
- [38] B. Kang, *Multiple classification ripple down rules*. Ph.D Dissertation, University of New South Wales, 1996.



- [39] G. A. Kelly, *The psychology of personal constructs*, Norton, New York, 1955.
- [40] D. M. Kienzle and M. C. Elder, "Recent worms: A survey and trends," in *Proceedings of WORM'03*, Washington DC, U.S.A, 2003.
- [41] G. Kolousek, "*The system architecture of an integrated medical consultation system and its implementation based on fuzzy technology*," Doctoral Dissertation, Technical University of Vienna, Austria, 1997.
- [42] T. T. Kuo, S. S. Tseng, and Y. T. Lin, "Ontology-based knowledge fusion framework using graph partitioning," in *Proceeding of IEA/AIE'2003*, pp. 11-20, 2003.
- [43] O. Lee and P. Gray, "Knowledge base clustering for KBS maintenance," *Journal of Software Maintenance: Research and Practice*, Vol. 10, pp. 395-414, 1998.
- [44] H. Leitich, H. P. Kiener, G. Kolarz, C. Schuh, W. Graninger, and K. P. Adlassnig, "A prospective evaluation of the medical consultation system CADIAG-II/RHEUMA in a rheumatological outpatient clinic," *Methods Inform Med*, Vol. 40, pp. 213-220, 2001.
- [45] S. C. Lin, S. S. Tseng, and Y. T. Lin, "A new mechanism of mining network behavior," *Lecture Notes in Artificial Intelligence 2336*, pp. 218-223, 2002.
- [46] Y. T. Lin, S. S. Tseng, and C. F. Tsai, "Design and implementation of new object-oriented rule base management system," *Expert Systems with Applications*, Vol. 25, pp. 369-385, 2003.
- [47] S. R. Madden, M. A. Shah, and J. M., "Hellerstein continuously adaptive continuous queries over streams," in *Proceedings of ACM SIGMOD 2002*, pp. 49-60, 2002.
- [48] K. L. Mcgraw and K. Harbison-Briggs, *Knowledge Acquisition: Principles and Guidelines*, Prentice-Hill International Editions, pp. 1-27, 1989.
- [49] G. Miller, "The magical number seven, plus or minus two: some limits on our capacity for processing information," *The Psychological Review*, Vol. 63, pp. 81-97, 1956.
- [50] J. Mirkovic, J. Martin, and P. Reiher, "A taxonomy of DDoS attacks and DDoS defense mechanisms," TR. 020018, Computer Science Department, University of California, Los Angeles, 2002.
- [51] D. Moore, C. Shannon, G. M. Voelker, and S. Savage, "Internet quarantine: requirements for containing self-propagating code," in *Proceedings of INFOCOM 2003*, Mar. 30 - Apr. 3, San Francisco, U.S.A., 2003.

- [52] B. Morin and H. Debar, "Correlation of intrusion symptoms: an application of chronicles," in *Proceedings of the 6th symposium on Recent Advances in Intrusion Detection (RAID 2003)*, pp. 92-112, 2003.
- [53] R. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang, "Exploratory mining and pruning optimizations of constrained associations rules," in *Proceedings of International Conference on Management of Data (SIGMOD'98)*, pp. 13-24, 1998.
- [54] S. Nikiforou and E. Fink, "Knowledge acquisition for clinical-trial selection," in *Proceedings of IEEE Int'l Conf. on Systems, Man and Cybernetics 2002*, Vol. 1, pp. 66 -71, 2002.
- [55] P. Ning, Y. Cui, and D. S. Reeves, "Constructing attack scenarios through correlation of intrusion alerts," in *Proceedings of 9th ACM Conference on Computer and Communications Security*, pp.245-154, 2002.
- [56] P. Ning, D. Xu, C. G. Healey, and R. A. St. Amant, "Building attack scenarios through integration of complementary alert correlation methods," in *Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS'04)*, 2004.
- [57] D. Pan, Q. L. Zheng, A. Zeng, and J. S. Hu, "A novel self-optimizing approach for knowledge acquisition," *IEEE Transactions on Systems, Man, and Cybernetics*, Part A, Vol. 32, No. 4, 505-514, 2002.
- [58] K. Park and H. Lee, "On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law Internets," in *Proceedings of the 2001 conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pp.15-26. 2001.
- [59] P. Politakis and S. M. Weiss, "Using empirical analysis to refine expert system knowledge bases," *Artificial Intelligence*, Vol. 22, pp. 673-680, 1984.
- [60] P. A. Porras, M. W. Fong, and A. Valdes, "A mission-impact-based approach to INFOSEC alarm correlation," in *Proceedings Recent Advances in Intrusion Detection*, pp.95-114, 2002.
- [61] A. Rafea and H. Hassen, "Automatic knowledge acquisition tool for irrigation and fertilization expert systems," *Expert Systems with Applications*, Vol. 24, Iss. 1, Jan. 2003, pp. 49-57, 2003.
- [62] T. Raz and N. Botten, "The knowledge base partitioning problem: Mathematical formulation and heuristic clustering," *Data and Knowledge Engineering*, Vol. 8,

- pp. 329-337, 1998.
- [63] M. Sabhnani and G. Serpen, G, "Application of machine learning algorithms to KDD intrusion detection dataset within misuse detection," in *Proceedings of the International Conference on Machine Learning; Models, Technologies and Applications (MLMTA'03)*, pp. 209-215, 2003.
- [64] M. L. G. Shaw and B. R. Gaines, "KITTEN: Knowledge initiation and transfer tools for experts and novices," *International Journal of Man-Machine Studies*, Vol. 27, pp. 251-280, 1987.
- [65] M. L. G. Shaw and B. R. Gaines, "Web Grid: Knowledge modeling and inference through the world wide web," in *Proceedings of tenth knowledge acquisition workshop*, pp. 65-1-65-14, 1996.
- [66] M. S. Shin, E. H. Kim, and K. H. Ryu, "False alarm classification model for network-based intrusion detection system," in *Proceedings of IDEAL 2004*, pp.259-265, 2004.
- [67] E. H. Shortliffe and B. G. Buchanan, "A model of inexact reasoning in medicine," *Math. Bioscience*, Vol. 23, pp. 351-379, 1975.
- [68] Snort, *Intrusion Detection/Prevention System*, <http://www.snort.org>, 2006.
- [69] R. Srikant and R. Aggrawal, "Mining sequential patterns: generalizations and performance improvements," in *Proceedings of the Fifth International Conference on Extending Database Technology*, 1996.
- [70] V. Stavridou, B. Dutertre, R. A. Riemenschneider, and H. Sa, "Intrusion tolerant software architectures," in *Proceedings. of DARPA Information Survivability Conference and Exposition (DISCEX II'01)*, Vol. 2, Anaheim, California, U.S.A., 2001.
- [71] J. M. Su, S. S. Tseng, W. Wang, J. F. Weng, D. J. T. Yang, and W. N. Tsai, "Learning portfolio analysis and mining for SCORM compliant environment," *Journal of Educational Technology & Society* (ETS), Vol. 9, Iss. 1, pp. 262-275, 2006.
- [72] Symantec Corp., "Symantec Internet Security Threat Report: Trends for July 05-Decamber 05," <http://www.symantec.com/index.htm>, 2005.
- [73] Tenable Network SecurityTM, 2005. Nessus Open Source Vulnerability Scanner Project. <http://www.nessus.org/>.
- [74] Trend Micro Co., Trend Mirco Network Viruswall, <http://www.trendmicro.com/tw/products/network/overview.htm>, 2005.

- [75] J. J. P. Tsai and A. Liu, "Knowledge-based software architectures: acquisition, specification, and verification," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 11, Iss. 1, pp. 187 -201, 1999.
- [76] S. S. Tseng and S. C. Lin "VODKA: Variant objects discovering knowledge acquisition," submitted to *International Journal of Human Computer Studies*, 2006. (under revision)
- [77] S. Tsumoto and S. Hirano, "Visualization of rule's similarity using multidimensional scaling," in *Proceedings of the Third IEEE International Conference on Data Mining (ICDM'03)*, 2003.
- [78] A. Valdes and K. Skinner, "Probabilistic alert correlation," in *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, pp.54-68, 2001.
- [79] G. Webb and J. Wells, "An experimental evaluation of integrating machine learning with knowledge acquisition," *Machine Learning*, Vol. 35, No. 1, pp. 5-23, 1999.
- [80] N. Weaver, V. Paxson, S. Staniford, R. Cunningham, and U. Maulik, "A taxonomy of computer worms," in *Proceedings of WORM'03*, Washington DC, U.S.A, 2003.
- [81] B. Wielinga, A. Schreiber, and J. Breuker, "KADS: A modeling approach to knowledge engineering," *Journal of Knowledge Acquisition*, Vol. 4, No. 1, pp. 5-53, 1992.
- [82] J. Xu and W. Lee, "Sustaining availability of web services under distributed denial of service attacks." *IEEE Transactions on Computers*, Vol. 52, Iss. 2, pp. 195-208, 2003.
- [83] H. M. Yan and Y. T. Jiang, "Internet-based knowledge acquisition and management method to build large-scale medical expert systems," in *Proceedings of the Second Joint Conference on EMBS/BMES*, Vol. 3, pp. 1885 -1886, 2002.

# Appendix A

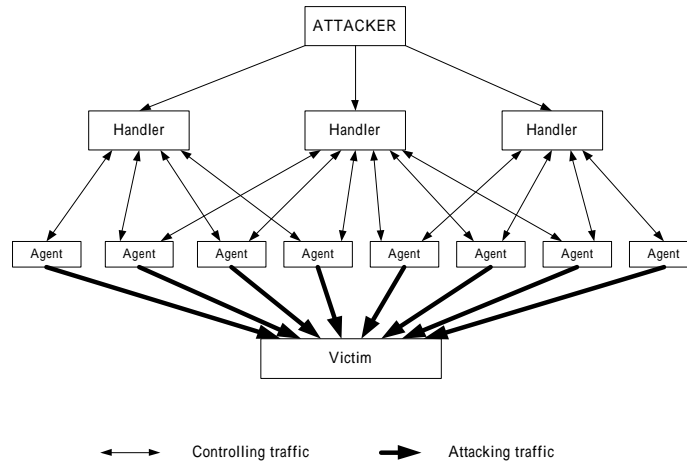
## Introduction of DDoS

### (1) DDoS Basic

Table A.1 The DDoS Attacks Developed from 1998 to 2002

Date	Attack	Attack method						
		UDP Flood	ICMP Flood	Smurf	TCP SYN	TCP ACK	TCP RST	TCP SYN/ACK
1998/5	FAPI	o	o					o
1999/6-1999/7	Trim00	o						
1999/8-1999/9	TFN	o	o	o	o			o
1999/9-1999/10	Stacheldracht	o	o	o	o			
1999/11	TFN 2K	o	o	o	o			
2000/1-2000/2	Shaft	o	o		o			
2000/2	Trinity	o			o	o	o	
2002/1	DRDoS							o
2002/4	Mstream					o		

The DDoS appeared in June of 1998 firstly means that many attackers launch malicious traffic to the same victims together and make the victims too busy to respond all the traffic including legitimate requests. Several different DDoS attacks which were developed [13][12][5][20][22] from 1998 to 2002 can be divided into several categories including UDP flood, ICMP flood, TCP flood, and Smurf as shown in Table A.1, so the common characteristics of each category of DDoS attacks could be easily observed and extracted. For example, the TFN2K, discovered in November 1999, is a kind of DDoS attacking tools. It can launch UDP flood, ICMP flood, Smurf, or TCP SYN flood attacking method to attack victims in one time. All of the observed knowledge can be represented as a natural rule format and stored into a knowledge base.



**Figure A.1 The General Topology of DDoS Attacks**

The general topology of DDoS attack shown in Figure A.1 could be divided into control stage and attack stage. In control stage, a scan is performed in large ranges of network to find the list of vulnerable hosts. Generally speaking, the vulnerable hosts consist of handlers and agents, where the handlers (the first level vulnerable hosts) are controlled by attackers and agents (the second level vulnerable hosts) are also controlled by attackers through handlers. Most of the controlling traffic, the traffic of communication in control stage, is signal direction between attacker and handler but is bi-direction between handlers and agents. The two level topology results in the locations of attackers can be hidden. After the control stage, the list of vulnerable hosts is then used to launch the distributed attacking traffic in attack stage. The attacking traffic including UDP flood, ICMP flood, Smurf, TCP SYN, TCP ACK, TCP RST, and TCP SYN/ACK as shown in Table A.1 could overwhelm the victim.

## (2) DDoS Intrusion Tolerance

Intrusion tolerance is the ability of a system to continue providing (possibly degraded but) adequate services after a penetration (Stavridou, 2001). As mentioned

above, it is very hard to detect and prevent the DDoS attacks. Therefore, the intrusion tolerance of DDoS attacks is an important issue to mitigate the damage during DDoS attacks for providing the critical services continuously on Internet. Park and Lee [58] suggested a method to install packet filters at different autonomous system on the Internet to filter out attacking traffic traveling between them. The method is very effective but not practical to defend DDoS attacks because of requiring the cooperation of thousands of autonomous systems on Internet.

Chang [CHA01] also introduced the concept of Internet firewall and four detecting and filtering approaches consisting of ubiquitous ingress packet filtering, router-based packet filtering, local attack detecting, and distributed attack detecting for defending flooding based DDoS attacks. He also indicated that more effective detect-and-filter approaches, such as distributed attack detecting, should be developed for DDoS intrusion tolerance. However, all of them lack a systematic approach to integrate the knowledge of DDoS intrusion tolerance.

Xu and Lee [82] proposed a DDoS intrusion tolerance system to sustain the availability of web service under DDoS attacks. The main idea is to isolate and protect legitimate traffic from huge volumes of DDoS traffic when an attack occurs. Unfortunately, it only aims on protecting web service instead of protecting the network.

# Appendix B

## The Example of Knowledge Class in DDoS Intrusion Tolerance

### (1) System State Diagram

In the traditional systems, NORMAL and DEAD states are usually enough to describe the system status, because the time is too short to respond the DDoS attacks for the administrator before the victim system moving to DEAD state. Therefore, the behaviors of the most traditional information systems will be NORMAL-DEAD pattern when they are attacked. By applying the various filtering policies for DDoS attacks according to the users' behaviors, the survival time of the system can then be extended and the SURVIVAL state is further added to represent such situation. With the heuristic of setting more restricted filtering policies, the system will tend to move SURVIVAL state quickly backed to NORMAL state. On the other hand, the behaviors of the DDoS intrusion tolerance system will be expected to be NORMAL-SURVIVAL-NORMAL pattern during DDoS attacks.

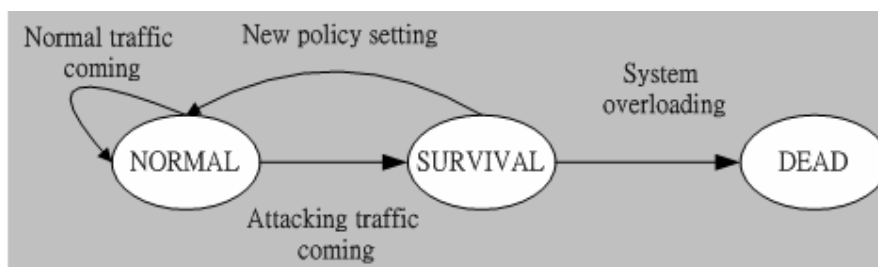


Figure B.1 System State Diagram

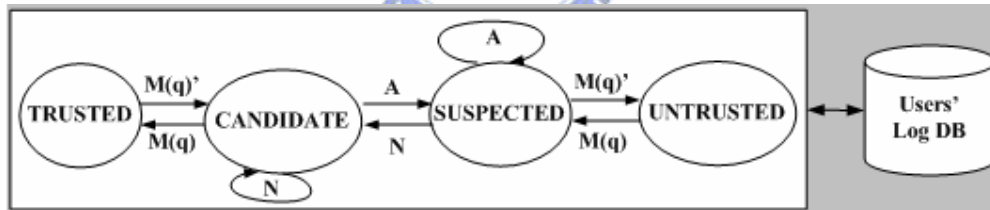
To describe the relationships between states of the system conceptually, the system state diagram including NORMAL, SURVIVAL, and DEAD states are proposed as shown in Figure B.1. A system capability and the bandwidth utilization



could be computed by the CPU load, the memory usage and the behaviors of access control list (ACL) to determine the transition of the system state diagram according to the new policy setting event, the attacking traffic coming event, and the system overloading event.

## (2) Role State Diagram

We assume the behaviors of normal users may not be changed dramatically in a short period of time. To represent the behaviors of each user, a role state diagram based upon an efficient ACL including white list and black list policy is proposed. We assume the DDoS attackers may frequently request service during the abnormal network status instead of normal ones. To monitor users' behaviors, the historical behaviors of the users based upon the current network status in a short time slice are proposed to distinguish the abnormal users from normal users efficiently, where each historical behavior is represented as a sequence of current network status.



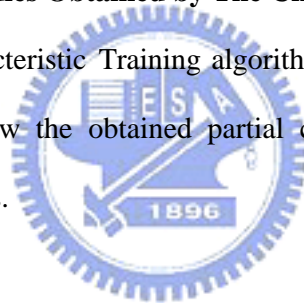
**Figure B.2 Role State Diagram**

According to the white list and black list shown in Figure B.2, all users could be categorized into TRUSTED, UNTRUSTED and CANDIDATE states. The  $N$  and  $A$  indicate that the current network status is normal and abnormal respectively, and the user in CANDIDATE or SUSPECTED state is moved to CANDIDATE state when current network status is  $N$ ; otherwise, he/she is moved to SUSPECTED state. Then, the  $M(q)$  and  $M(q)'$  indicate that the historical behavior of the user  $q$ , which is

acquired from Users' Log DB, is normal and abnormal respectively. The user in CANDIDATE state will be moved into TRUSTED state if he/she has accumulated sufficient normal behaviors; it means that the historical behavior of the user  $q$  should be normal,  $M(q)$ . On the other hand, the user in SUSPECTED state will be switched to UNTRUSTED state if he/she has accumulated sufficient abnormal behaviors,  $M(q)'$ . The users in TRUSTED and UNTRUSTED states will be moved to CANDIDATE and SUSPECTED states respectively if their historical behaviors match the corresponding constraints. With the role state diagram, the filtering policies could be adaptively generated based upon dynamically changing network environment.

### (3) Examples of Detailed Rules Obtained by The Characteristic Trainer

According to the Characteristic Training algorithm shown in Algorithm 6.1, the following six examples show the obtained partial characteristics/features and the corresponding detection rules.



#### ■ **Example B.1:** *The ratios of untrusted IP addresses and ports*

Since the attacking traffic of DDoS is launched from the compromised hosts which disperse on Internet, most of the attacking traffic comes from the untrusted IP addresses. Also, some DDoS attacking tools may generate the destination port of attack packets randomly. Therefore, we select the ratios of traffic from untrusted IP addresses and ports as two important characteristics in Step 1 to detect the occurrence of DDoS attacking traffic, set dangerous ratio  $d = 50$  by comparing the list of ports and ACL defined by ISPs in Step 2, and the action “Trigger Filter KC” for these two detecting features is chosen in Step 3. Finally, the rule “*IF (the ratio of untrusted IP addresses > 50) OR (the ratio to untrusted ports > 50) Then (Spoofing DDoS*

*attacking) AND (Trigger Filter KC)*” is generated to discover DDoS attacks.

■ **Example B.2:** *Number of flows from the same source increase rapidly*

Generally speaking, the number of flows from one source normally does not increase dramatically if the attacks without spoofing did not occur. Hence we select the number of flows from one specific source as a characteristic of suspected user to detect the DDoS attacks in Step 1. Next, the rapidly increasing rate  $d = 50$  in Step 2. In Step 4, the rule “*IF (Number of flows from the same source > 50) Then (Same source DDoS attacking) AND (Trigger Filter KC)*” is generated to discover DDoS attacks.

■ **Example B.3:** *Number of flows in the state of “SYN\_RECEIVED”*

Resource consumption attacks often use the SYN flood technique (Criscuolo, 2000), such as TFN, TFN2K, stacheldrucht and so on, to overwhelm the victim, where client sends a fake packet to Server, and Server accepts this SYN packet, responds the SYN/ACK packet to the fake address, and waits for the response of SYN/ACK packet. But the response will never arrive due to the faked source address. Therefore, the number of flows in the state of SYN\_RECEIVED is first selected to detect the DDoS attacks in Step 1 and the  $d$  is set to 1000 in Step 2 in this example. Finally, the rule such as “*IF (#SYN\_RECIEIVED > 1000) Then (SYN flooding DDoS attacking) AND (Trigger Filter KC)*” is generated to discover DDoS attacks.

■ **Example B.4:** *Percentages of UDP and ICMP packets*

The percentages of UDP and ICMP packets usually representing the error control messages during communications are always small in normal traffic and they will

become large if the bandwidth consumption DDoS attack, the UDP or ICMP flood, could be launched to overwhelm victims. Since the percentages of UDP and ICMP packets are various for different autonomous systems, the environment-dependent characteristics on monitoring the huge traffic of UDP and ICMP packets are required and selected to detect the DDoS attacks in Step 1. The  $d$  is then set to be 30 in Step 2. Finally, the corresponding detection rules such as “*IF (P(UDP) > 30) OR (P(ICMP) > 30) Then (Flooding-based DDoS attacking) AND (Trigger Filter KC)*” are generated to discover DDoS attacks.

■ **Example B.5:** *Oversize of UDP packets and ICMP packets*

As mentioned above, the UDP and ICMP packets are usually the error control messages during communication, the encryption of payload in packet for DDoS tools is usually used for communication between attackers, handlers, and agents in controlling traffic, since the controlling traffic stores the source addresses of attackers, handlers and victims and attacks do not want to be revealed. Besides, the accounts, the passwords, and the commands to control handlers and agents from attackers are also necessarily encrypted. Since encrypting the information may enlarge the size of UDP and ICMP packets, the ratio of oversize UDP and ICMP packets might be selected as good features in Step 1 and set threshold value = 100. Finally, the corresponding prediction rules such as “*IF (#Oversize > 100) Then (Controlling traffic attacking) AND (Trigger DDoS Prediction KC)*” are generated to predict the DDoS attacks.

■ **Example B.6:** *Percentage of BASE64 encoding packets*

Once the communication packets are encrypted, the resulting special characters

should be encoded to avoid the occurrence of errors in communication during DDoS attacks. BASE64 encoding is very popular in solving this problem, but the payload of packet contains only alphanumeric character and 0x41('A') always appears at the end of the BASE64 encoding packet in the TFN2K attack. Both may be selected as useful features to predict the occurrence of DDoS attacks in Step 1. Finally, the predicting rule “*IF (AlphaBeta = Yes) AND (Tailing = ‘A’) Then (TFK2K Communication traffic attacking) AND (Trigger Filter KC)*” is generated.

The following table shows the summarization of Example B.1 to Example B.6.

Feature Name ( <i>f</i> )	Operator ( <i>q</i> )	Threshold ( <i>d</i> )	Actions ( <i>A</i> )
<i>ratio of untrusted IP</i>	>	50	<i>(Spoofing DDoS attacking) AND (Trigger Filter KC)</i>
<i>ratio of untrusted ports</i>	>	50	<i>(Spoofing DDoS attacking) AND (Trigger Filter KC)</i>
<i>ratio of same IP</i>	>	50	<i>(Same IP DDoS attacking) AND (Trigger Filter KC)</i>
<i>#SYN_RECIEIVED</i>	>	1000	<i>(SYN flooding attacking) AND (Trigger Filter KC)</i>
<i>UDP percentage</i>	>	30	<i>(Flooding-based attacking) AND (Trigger Filter KC)</i>
<i>#Oversize packets</i>	>	100	<i>(Controlling traffic) AND (Trigger Prediction KC)</i>
<i>AlphaBeta</i>	=	Yes	<i>(Communication traffic) AND (Trigger Filter KC)</i>
<i>Packet Tailing</i>	=	‘A’	<i>(Communication traffic) AND (Trigger Filter KC)</i>

#### (4) An Example of Knowledge Classes

The following shows a simple example of KC to defend the TFN2K attack, where the Profile KC includes system state transition rules and user state transition rules, the Detection KC includes attack detecting and attack predicting sub-KCs, the Detection KC includes rules focusing on detecting TFN2K attack, the Filter KC lists the heuristics of policy learning proposed in this paper, and the Evaluation KC illustrates the evaluating rules including system performance evaluating rules and the network performance evaluating rules.

##### ■ A Partial Profile KC

<b>System state transition rules</b>	
IF Ss = NORMAL AND Traffic = Normal Then Ss = NORMAL	//Ss is system state
IF Ss = NORMAL AND Traffic = Attack Then Ss = SURVIVAL	
IF Ss = SURVIVAL AND Policy_Set = Enable Then Ss = NORMAL	
IF Ss = SURVIVAL AND System = Overload Then Ss = DEAD	
<b>User state transition rules</b>	
IF Us = TRUSTED AND Uhb = M Then Us = CANDIDATE	//Us is user state
IF Us = CANDIDATE AND Uhb = M Then Us = TRUSTED	//Uhb is historical user behavior
IF Us = CANDIDATE AND Ncb = N Then Us = CANDIDATE	//Ncb is current user behavior

```

IF Us = CANDIDATE AND Ncb = A Then Us = SUSPECTED
IF Us = SUSPECTED AND Ncb = N Then Us = CANDIDATE
IF Us = SUSPECTED AND Ncb = A Then Us = SUSPECTED
IF Us = SUSPECTED AND Uhb = M' Then Us = UNTRUSTED
IF Us = UNTRUSTED AND Uhb = M Then Us = SUSPECTED

```

### ■ A Partial Detection KC

```

Attack detecting sub-KC
IF P(Protocol) >  $\delta_{\text{Protocol}}$  Then Flooding-based DDoS attacking AND Trigger Filter KC
// P(Protocol) is the percentage of protocol
IF IR(U) > 50% Then Same source DDoS attacking AND Trigger Filter KC
// IR(U) is the increase rate of the user
IF (Untrusted > 50%) AND (#Request > 200) Then Spoofing DDoS attacking AND Trigger Filter KC
// Untrusted is the rate of users in UNTRUSTED
IF #SYN_RECEIVED >  $\delta_{\text{SYN}}$  Then SYN-flooding DDoS attacking AND Trigger Filter KC
Attack predicting sub-KC
IF AlphaBeta = Yes AND Tailing = 'A' Then TFK2K Communication traffic attacking AND Trigger Filter KC
//BASE64 encoding
IF #OverSize > 10 Then Controlling traffic attacking AND Trigger DDoS Prediction KC //Oversize packet

```

### ■ A Partial Filter KC

```

Acquire Evaluation KC
IF  $N_{cb} = N$  AND Us = CANDIDATE Then  $S(U) = S(U) + 1$  //S(U) is the historical behavior score of user
IF  $N_{cb} = A$  AND Us = SUSPECTED Then  $S(U) = S(U) - 1$ 
IF  $N_{cb} = N$  AND  $N_{cl}(U) = N$  AND Us = TRUSTED Then  $S(U) = S(U) + 1$  //Ncl is the last network state
IF  $N_{cb} = A$  AND  $N_{cl}(U) = A$  AND Us = TRUSTED Then  $S(U) = S(U) - 1$ 
IF  $N_{cb} = N$  AND  $N_{cl}(U) = N$  AND Us = UNTRUSTED Then  $S(U) = S(U) + 1$ 
IF  $N_{cb} = A$  AND  $N_{cl}(U) = A$  AND Us = UNTRUSTED Then  $S(U) = S(U) - 1$ 
IF  $S(U) \geq G$  Then  $U_{hb} = M$ 
IF Us = TRUSTED Then Set U in WL
IF  $S(U) \leq -G$  Then  $U_{hb} = M'$ 
IF Us = UNTRUSTED Then Put U in BL
IF U in BL Then (Block U) AND (Policy_Set = Enable)
IF Policy_Set = Enable Then Acquire Evaluation KC
IF U in WL Then Trigger Detection KC

```

### ■ A Partial Evaluation KC

```

IF CPU = X AND MEM = Y Then  $r = \text{AVG}(X, Y)$ 
IF  $r \geq 90$  Then System = Overload
IF  $r \leq 45$  Then Traffic = Normal AND Policy_Set = Disable
IF ( $r > 45$ ) AND ( $r < 90$ ) Then Traffic = Attack
IF  $Bw \geq 50$  Then  $N_{cb} = A$ 
IF  $Bw < 50$  Then  $N_{cb} = N$ 

```

When the TFN2K traffic is coming, the Filter KC will be firstly triggered to filter the users in black list. Next, it will acquire the Evaluation KC to obtain the current network situation and then report abnormal ( $N_{cb} = A$ ). In the meanwhile, the Filter KC will also trigger the Detection KC for detecting the users who pass the Filter KC. If

the attacker launched “TCP-SYN flood” attack, then the rule “*(IF #SYN\_RECEIVED > d<sub>SYN</sub> Then SYN-flooding DDoS attacking AND Trigger Filter KC)*” will be matched; hence the Filter KC is triggered to adapt the ACL. If one user’s behavior is abnormal comparing to his/her historical behavior, he/she is gradually moved to black list and the Policy\_Set is enabled. After new policy is set, the Filter KC will trigger Evaluation KC to determine the performance of the policy. If the system capacity become high, then system state will be transformed into NORMAL; otherwise, the policy will be reset if necessary.



# Appendix C

## The Examples for Rule Base Partitioning

**Example C.1:** The following five rules for detecting different network anomalies form a rule base RB; that is,  $RB = \{r_1, r_2, r_3, r_4, r_5\}$ .

$r_1$  : IF {(protocol = TCP), (protected\_network\_ direction = A), (source\_port > 8080), (string = NetBus)} THEN {(name = NETBUS)};

$r_2$  : IF {(protocol = TCP), (protected\_network\_ direction = A), (source\_port > 1023, (string = NetBus2)} THEN {(name = NETBUS2)};

$r_3$  : IF {(protocol = UDP), (protected\_network\_ direction = A), (destination\_ port > 1023), (string = /ce/63/d1/d2/16/e7/13/cf/3c/a5/a5/86)} THEN {(name = DIR)};

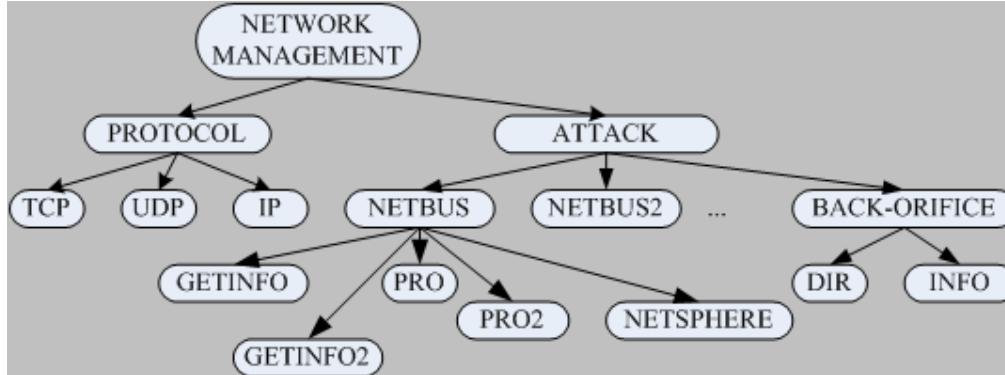
$r_4$  : IF {(protocol = UDP), (protected\_network\_ direction = A), (destination\_ port > 1023), (string = /ce/63/d1/d2/16/e7/13/cf/39/a5/a5/86)} THEN {(name = INFO)};

$r_5$  : IF {(protocol = TCP), (protected\_network\_ direction = A), (destination\_ port = 53), (string = /00/00/ff)} THEN {(name = ANY-TCP)}.

**Example C.2:** For the rules  $r_1$  and  $r_2$  listed in **Example C.1**, the set of common attribute names in both conditions is  $\text{sharein}(r_1, r_2) = \{\text{protocol, protected\_network\_direction, source\_port, string}\}$ , and the set of common attribute name in both actions is  $\text{shareout}(r_1, r_2) = \{\text{name}\}$ , while  $\text{inout}(r_1, r_2) = \emptyset$ , and  $\text{notshared}(r_1, r_2) = \emptyset$ .



**Example C.3:** Given the ontology illustrated in Figure C.1, suppose the constant  $c$  is set to 0.9; the semantic relatedness between  $e_1 = (\text{name} = \text{NETBUS})$  and  $e_2 = (\text{name} = \text{NETBUS2})$  is 0.25.



**Figure C.1** Part of The Network Ontology

**Example C.4:** Given two numerical expressions  $e_1 = (\text{port} > 1023)$  and  $e_2 = (\text{port} > 8080)$ , and the corresponding intervals  $i_1 = (1023, 65535]$  and  $i_2 = (8080, 65535]$ , the semantic relatedness between these two expressions is 0.89.

**Example C.5** For two rules,  $r_1$  and  $r_2$ , listed in **Example C.1**, the rule similarity between  $r_1$  and  $r_2$  is  $R(r_1, r_2) = 0 \cdot 0.4 + (1 + 1 + 0.89 + 0) \cdot 0.2 + 0.25 \cdot 0.3 - 0 \cdot 0.1 = 0.653$ .

**Example C.6** Assume that all rules listed in **Example C.1** are partitioned according to the Rule Base Partitioning Algorithm. The similarity threshold  $st$  is set to 1.5.

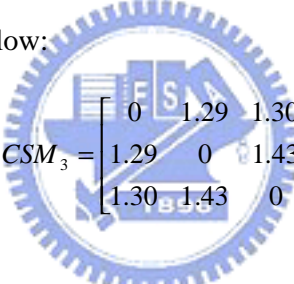
At first, each rule is assigned to a single rule cluster, i.e.,  $g_1 = \{r_1\}$ ,  $g_2 = \{r_2\}$ ,  $g_3 = \{r_3\}$ ,  $g_4 = \{r_4\}$ , and  $g_5 = \{r_5\}$ . The CSM for this configuration is shown as below:

$$CSM_1 = \begin{bmatrix} 0 & 1.81 & 1.37 & 1.37 & 1.45 \\ 1.81 & 0 & 1.42 & 1.42 & 1.45 \\ 1.37 & 1.42 & 0 & 1.86 & 1.54 \\ 1.37 & 1.42 & 1.86 & 0 & 1.54 \\ 1.45 & 1.45 & 1.54 & 1.54 & 0 \end{bmatrix}$$

By reviewing  $CSM_1$ ,  $n_{34}$  has the largest value of the matrix and is larger than  $st$ ;  $g_3$  and  $g_4$  are combined to form a new rule cluster. The remaining rule clusters are  $g_1 = \{r_1\}$ ,  $g_2 = \{r_2\}$ ,  $g_3 = \{r_3, r_4\}$ , and  $g_4 = \{r_5\}$ . The  $CSM_2$  in this iteration is shown as below:

$$CSM_2 = \begin{bmatrix} 0 & 1.81 & 1.20 & 1.45 \\ 1.81 & 0 & 1.27 & 1.45 \\ 1.20 & 1.27 & 0 & 1.43 \\ 1.45 & 1.45 & 1.43 & 0 \end{bmatrix}$$

The largest value within the  $CSM_2$ ,  $n_{12}$ , can be discovered. Two rule clusters  $g_1$  and  $g_2$  are thus combined since  $n_{12}$  is larger than 2. After grouping  $g_1$  and  $g_2$ , the  $CSM_3$  can be generated and shown as below:



$$CSM_3 = \begin{bmatrix} 0 & 1.29 & 1.30 \\ 1.29 & 0 & 1.43 \\ 1.30 & 1.43 & 0 \end{bmatrix}$$

No more grouping is needed because that all entries of  $CSM_3$  are less than similarity threshold  $st$ . The process is terminated and output the result, set of the rule clusters,  $\{\{r_1, r_2\}, \{r_3, r_4\}, \{r_5\}\}$ .

**Example C.7** In the **Example C.6**, rule base RB is partitioned into three rule clusters. The minimum support threshold,  $min\_sup$ , is set to 0.9. For the sake of simplicity, every expression occurred in the RB is encoded in Table C.1.

**Table C.1 Encodings of Expressions in RB**

encoding	Expression
----------	------------

e <sub>1</sub>	(protocol = TCP)
e <sub>2</sub>	(protected_network_direction = A)
e <sub>3</sub>	(source_port > 8080)
e <sub>4</sub>	(source_port > 1023)
e <sub>5</sub>	(string = NetBus)
e <sub>6</sub>	(string = NetBus2)

There are two rules in  $g_1$ ,  $T_1$  consists of two transactions,  $t_{11} = \{e_1, e_2, e_3, e_5\}$  and  $t_{12} = \{e_1, e_2, e_4, e_6\}$ . After several iterations, we can obtain the candidate 3-itemsets,  $C_{13}$ , and the process is thus terminated since  $|D_{13}| = 1$ , no more candidate itemsets of  $C_{14}$  can be generated. Therefore, the final output is  $D_{13} = \{\{e_1, e_2, e_4\}\}$ . According to Table C.1, the frequent combination of expressions is  $\{(protocol = TCP), (protected\_network\_direction = A), (source\_port > 1023)\}$ .



# Appendix D

## Rule Class Construction Algorithms of Model Constructing Phase

### (1) The Concept of Constructing Alert Models

It is very difficult for experts and administrators to monitor on-line IDS alerts to discover useful intrusion patterns. Nowadays, experts are still using their own knowledge and experience to defend intrusions, but the efficiency and effectiveness of intrusion detection are hard to improve. Our idea is to construct a model to classify the collected alert patterns into several classifications with different flags, and experts can discover on-line suspicious or intrusion patterns easily and quickly. The model consists of normal, intrusion and suspicious behavior classification rule classes, where each rule class consists of hundreds to thousands classification rules depending on the computational ability of each detection sensor. As shown in Figure D.1, each rule class is constructed with individual construction method, and use individual data source in specific network environments as their data input.

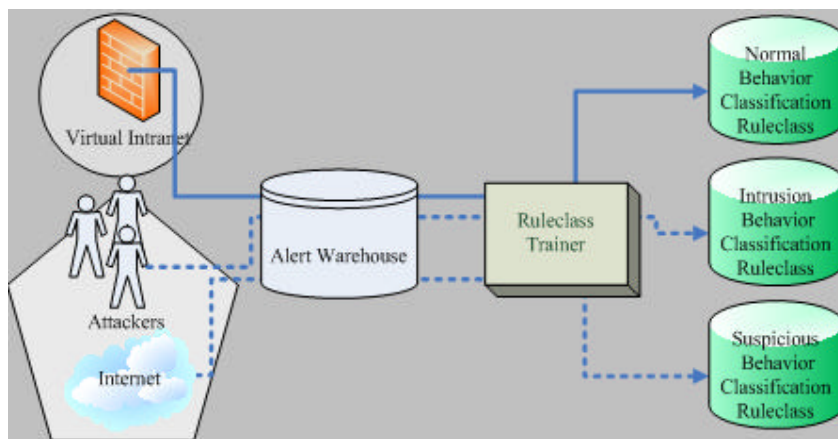


Figure D.1 Three Types of Alert Behavior Classification Rule Classes

Normal behavior classification rule class represents the set of false alert patterns. Frequent behaviors of IDS alerts in an attack-free environment are the most alerts in experience, and these alerts are exactly false alerts. We use an AprioriAll-like sequential pattern mining algorithm to discover frequent sequences of IDS alerts in an attack-free environment in laboratories, and these frequent sequences are used to construct classification rules of normal rule class.

As we know, some intrusions which have fixed patterns can be easily discovered by simple pattern matching approach on-line. However, varied intrusions become too sophisticated to detect, our suspicious/intrusion classification rule construction method is hence focusing on these kinds of intrusions. We design a score based method in this approach to discover varied alert subsequences as suspicious patterns. Some rootkit tools are used to simulate real-world attacks in a period of time, and these IDS alerts are collected as training data of suspicious/intrusion classification rule class construction. Some suspicious alert sequences will be discovered with this method, and system will interact with experts to classify these sequences into intrusion rule class or suspicious rule class; if one sequence is considered as a known intrusion, it is classified by intrusion rule class, and then used to update the classification rule of intrusion rule class; if one discovered alert subsequence is not considered as a known intrusion, it is highly possible to be a novel intrusion and should be classified into suspicious rule class to construct a new suspicious classification rule.

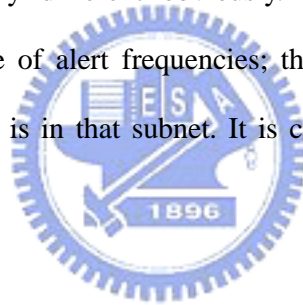
## **(2) Normal Behavior Rule Class Construction**

To achieve an objective of high detection rate without missing any intrusion, the

design of IDS signature-based rules is asked to be as powerful as possible, but that makes IDS become more sensitive. Filtering of dirty alerts has two advantages including reducing the accuracy of alert analysis and the complex of data execution at the same time. Generally speaking, most of these researches use special analysis methods or compile expert experiences to construct filter models, and use this filter model to discard those highly-possible false alerts to get clear data. Before discussing the design of procedure, we must consider the characteristics of alerts and attacks.

(i) Alert Frequency

According to different importance and bandwidth of hosts, their numbers of triggered alerts are very different obviously. Besides, the scale of subnets aggregate the difference of alert frequencies; the bigger scale a subnet is, the more total alert number is in that subnet. It is common to collect thousands of alerts in a busy subnet.



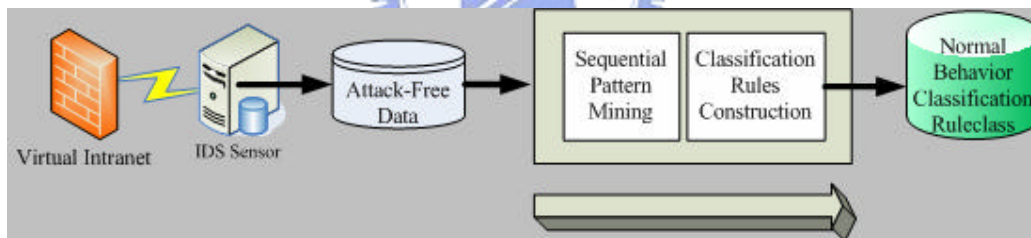
(ii) False Alert Frequency

There are full of false alerts in alert warehouse. According to the results of most researches, it is indicated that false alert rates of different IDSs are lain in between 60% to 90%. The most important concept of all, some researches indicate that some of these false alerts occur with similar patterns in the same network, such as specific alert sequences or frequent source IP addresses, and those normal behaviors are triggered as alerts but they are not intrusions in fact. In other words, the idea of these researches is that if we can discover frequent behavior patterns of alerts, these frequent patterns are most likely to be false alerts.

(iii) Attack Characteristic

There are also some characteristics in attacks, so some filtering methods using specific intrusion's characteristics to efficiently detect the corresponding intrusion. For an example, some specific Rootkits will usually give rise to fix alert sequences, so it is more appropriate for these intrusions to use Sequential Pattern Mining to filter out false alerts. For another example, worm is a kind of variable intrusions, so using Generic Algorithm to filter false alerts seems better than others.

Our idea is to construct classification rules of normal behavior classification rule class, and then we can execute filtering by comparing all on-line alert sequences with the normal behavior classification rule class.



**Figure D.2 The Procedure of Normal Behavior Classification Rule Class Construction**

A Normal Behavior Classification Rule Class Construction (NBCRC) algorithm shown in Figure D.2 is proposed to discover attack sequences of normal network behaviors. This algorithm consists of two steps of sequential pattern mining and classification rule construction. At first, we assume frequent behavior, over an extended period of time, is likely to be normal. In other words, a modified *AprioriAll*

sequential pattern mining method (Valdes and Skinner, 2001) is used to discover frequent sequences of alerts in single sensor, and these frequent sequences are seen as false alert patterns and collected as a rule class. At last, all frequent sequences are flagged with ‘normal’ and transformed to classification rules of normal behavior classification rule class. The normal behavior classification rule class is used to reduce false alert sequences in on-line stage. The specific algorithm proposed by us is shown in Algorithm D.1. To fit in with requirements of flexibility and robustness for administrators in such a decision support system, system interacts with administrators to decide the value of minimum support in AprioriAll algorithm dynamically. That makes it possible for administrators to make proper decisions according to different situations.

**Algorithm D.1 The Normal Behavior Classification Rule Class Construction**

**Algorithm**

**Input:** Alert sequences of individual IDS sensors.

**Output:** Normal rule class with classification rules.

**Step 1:** For each sensor, Ask administrators to decide the appropriate values of minimum support in AprioriAll.

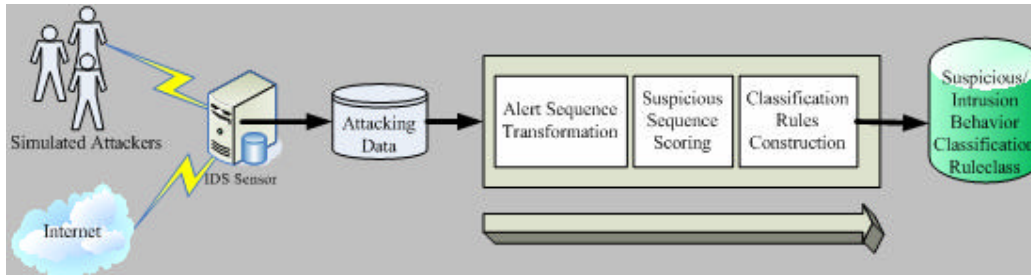
**Step 2:** Generate frequent sequences of one sensor by AprioriAll with specific value of minimum support.

**Step 3:** Flag each frequent alert sequences with ‘normal’ and use these flagged alert sequences to construct classification rules of normal behavior patterns.

**(3) Intrusion/Suspicious Behavior Rule Class Construction**

As shown in Figure D.3, the procedure of suspicious/intrusion behavior classification rule class construction, including alert sequence transformation, suspicious scoring and classification rule construction, is proposed to construct suspicious/intrusion behavior classification rule class, where each rule is represented as a suspicious/intrusion alert sequence.





**Figure D.3 The Procedure of Suspicious/Intrusion Classification Rule Class**

### **Construction**

Since the false alerts have been filtered out in this phase, all alert sequence transactions into 2-candidate alert subsequences (2-candidate means the length of this sequence is 2) to inspect each possible alert sequence strictly. Besides, we propose the specific suspicious scoring method to model possible behaviors of intrusions where two variables are designed to record locations and frequency of each 2-candidate alert subsequences respectively between several consecutive time short-term windows. According to the characteristics of intrusions, higher the scoring value is, more suspicious the alert pattern is. If a subsequence is continuously repeated and discovered in the same host set, it will be treated as a suspicious behavior of intrusion.

For each 2-candidate alert subsequence,  $Host(t_i)$ , a set of hosts, represents the locations which discovered this subsequence in the time period  $t_i$  and the  $|Host(t_i)|$ , the number of hosts, represents the frequency which discovered this subsequence in  $t_i$ . Assume there are  $n$  hosts in the simulated environment. The scoring policy could be divided into three cases according to  $Host(t_i)$  and  $|Host(t_i)|$ . In case 2 and case 3,  $\oplus$  is an exclusive-or operator.

## Scoring Policies

**CASE 1:**  $Host(t_i) = Host(t_{i-1})$ . // Continuous attacking

Set **Score**=0 AND **Repeat**++.

**CASE 2:**  $Host(t_i) \neq Host(t_{i-1})$  &  $|Host(t_i)| = |Host(t_{i-1})|$ . //Light variation

Set **Score** =  $Score + \frac{1}{2} [ |Host(t_i) \oplus Host(t_{i-1})| * (1/n) ]$

AND **Repeat**=0.

**CASE 3:**  $Host(t_i) \neq Host(t_{i-1})$  &  $|Host(t_i)| \neq |Host(t_{i-1})|$ . // Heavy variation

Set **Score** =  $Score + |Host(t_i) \oplus Host(t_{i-1})| * [ ||Host(t_i)| - |Host(t_{i-1})|| ]$

$*(1/n)$  AND **Repeat**=0.

Finally, specific thresholds are set to flag some special situations as suspicious attack patterns, and then administrators are noticed to trace the causes of suspicious patterns and fix intruded hosts. Four flagging rules are proposed as follows to determine 2-candidate alert sequences with suspicious patterns if there is any 2-candidate alert sequence conforming to one of these rules. Therefore, all alert sequences of suspicious behaviors are constructed into classification rules of suspicious/intrusion behavior classification rule class.


## Flagging rules

<p><b>RULE 1: IF</b>           <b>Score &gt; Threshold(score),</b>           <b>THEN</b> flag as “Suspicious”.</p> <p><b>RULE 2: ELSE IF</b>   <b>(Repeat &gt; Threshold(repeat) &amp;  Host(i) !=0 ),</b>           <b>THEN</b> flag as “Suspicious”.</p> <p><b>RULE 3: ELSE IF</b>     <b> Host(i)  == n,</b>           <b>THEN</b> flag as “Suspicious”.</p> <p><b>RULE 4: ELSE</b>        flag as “Unkonwn”.</p>
--

After the procedure of suspicious/intrusion behavior classification rule class construction, we can construct lots of suspicious alert sequences, and then experts are

asked to flag these suspicious alert sequences with tags of specific intrusions

System provides each suspicious alert sequence to experts, and experts flag these alert sequences by their own experiences and domain knowledge to construct classification rules of suspicious behavior classification rule class or intrusion behavior classification rule class. If one alert sequence is considered as a known intrusion pattern, it is necessary for experts to flag this alert sequence with specific intrusion signature tag which represents a name of known attack; if one alert sequence is not considered as a known intrusion by experts, it is perhaps a kind of novel intrusion patterns, and this alert sequence is still flagged as ‘suspicious’ to be constructed as a new classification rule of suspicious behavior classification rule class.



According to scoring policies and flagging rules, the algorithm of the suspicious/intrusion behavior classification rule class construction is shown in Algorithm 2. After calculating the  $Host(t_i)$  and  $|Host(t_i)|$  at time  $t_i$  in **Step 2**, the scoring policy is applied to calculate the values of variation score and repeat score of the 2-subsequence. Then, the Scoring Policies and Flagging Rules are used to help experts construct the classification rules in **Step 4** and **Step 5**. In order to discover complete intrusions patterns, we will string up 2-candidates alert subsequences into all k-subsequences in **Step 6** if two subsequences have the end to end relation. For example, if a set of 2-candidate subsequences is {AB, BC}, then the extended subsequences {ABC} will be generated. After all extended alert subsequences are discovered, experts are asked to flag the classification label in each alert subsequence in **Step 7**.

## Algorithm D.2 The Suspicious/Intrusion Behavior Classification Rule Class

### Construction Algorithm

<b>Input:</b> Candidate alert sequences, <i>Threshold(score)</i> and <i>Threshold(repeat)</i> .
<b>Output:</b> Alert behavior classification rules of suspicious and intrusion behavior.
<b>Step 1:</b> For each sensor, Transform the candidate alert sequences into 2-candidate subsequences.
<b>Step 2:</b> For each 2-subsequence, Calculate the $Host(t_i)$ and the $ Host(t_i) $ values.
<b>Step 3:</b> Store results of all 2-candidate subsequence transactions.
<b>Step 4:</b> Apply Scoring Policies to calculate the values of Score and Repeat.
<b>Step 5:</b> Flag classification label using the Flagging Rules.
<b>Step 6:</b> Aggregate the extended alert subsequences.
<b>Step 7:</b> Interact with experts to flag all suspicious alert sequences to classify these alert sequences into suspicious alert classification rules and intrusion alert classification rules.



# Appendix E

## The Overview of The Related Tools

Snort [68] is a signature-based intrusion detection system and open source software. It represents a cost-effective and robust NIDS solution that fits the needs of many organizations. Snort is very flexible in the ways it can be deployed. Many security industry watchdogs use the Snort signatures as part of their security announcements (such as CERT). Intrusions are ravaging the Internet since they are constantly evolving to new variants by multiple updates weekly. The Snort mailing lists are fantastic resource for people who are trying to write their own signatures to develop the applications of central monitoring and alerting consoles.

BASE [6] is the Basic Analysis and Security Engine. It is based on the code from the Analysis Console for Intrusion Databases (ACID) project. This application provides a web front-end to query and analyze intrusions from the alerts coming generated by Snort.

To post processing of alert transactions requires commercial databases, e.g., the MS-SQL 2000 server, which is used to automatically extract, transform and load data from heterogeneous sources. The MS-SQL 2000 Server Analysis Services includes OLAP, data mining and data warehouse tools, which makes better decisions, performs rapidly, and executes analysis on large and complex data sets using multi-dimensional storage.

The DRAMA [19] is applied for building up the decision support inference engine. DRAMA is a rule-based, client-server tool/environment for knowledge-based system development. It can assist knowledge engineers in building up an rule-based decision support system. Using the client-server architecture of DRAMA, the rule base is maintained on a server and clients could access DRAMA server for inference services.



# Appendix F

## The Case Study of e-Learning Using VODKA

In e-learning, the learning behaviors of students and their learning achievements are usually different even if they study the same learning content (the knowledge of teachers). Therefore, teachers want to apply appropriate teaching strategy to provide personalized learning content and learning sequence for students to improve their learning performance. In this case, the objects to be classified is defined as learning behaviors of students, where each behavior consists of profiles, learning sequence, and quiz grade of the student. The students can be firstly clustered into several groups according to the similarity of the learning behaviors, and teachers can provide appropriate learning content for each group in advance. However, students might change their learning sequence due to different learning situation, learning equipments (desktop, PDA, etc.), course content (text, video, etc.), learning time (day or night). This causes the evolution of learning behaviors and results in various learning achievements. As we know, the quiz for students is useful to evaluate their learning achievement. For example, teachers should provide easier learning content or learning sequence for the students with lower learning achievement.

Hence, *VODKA* provides a good idea to assist teachers in observing the occurrence of variant learning behaviors through a sequence of online quiz to evaluate the learning performance of each student with different learning sequence, and then to notify them for generating suitable learning sequence for further applying. Here, each learning sequence deviated from one of predefined learning

sequences will be treated as a variant learning behavior. In e-learning, it is important for student to gain a good grade after learning some materials with a specific learning sequence. Hence, the grade of quiz is treated as a CF for collecting these good variant learning sequences. If many students gained grades larger than a threshold with similar or same learning sequence (high frequency), some good variant learning sequences will be discovered to notify teachers to determine these new learning sequences. Therefore, the log is collected as the pair of  $\langle LS_i, CF_i \rangle$ , where  $LS_i$  is the leaning sequence of the student  $i$ ; and the  $CF_i$  is the grade of this student. Example 4.2 illustrates the concept of e-learning using *VODKA*.

#### **Example F.1 The Concept of e-learning Using *VODKA***

To simplify our discussion, we assume *VODKA* collects several good learning behaviors of students and their grades of quiz are larger than a threshold. For the learning sequence log shown in Table F.1,  $LS_1 = \langle B, C, A, D, E, F, G, H, I, J \rangle$  denotes that Student 1 studies the learning content B first and then studies the learning contents C, A, D, E, F, G, H, I, J sequentially.

**Table F.1 The Learning Sequence of Students**

Student ID	Learning Sequence
1	$\langle B, C, A, D, E, F, G, H, I, J \rangle$
2	$\langle A, B, H, D, E, F, C, G, I, J \rangle$
3	$\langle A, D, F, G, H, B, C, I, J \rangle$
4	$\langle A, B, D, E, C, F, G, H \rangle$
5	$\langle A, C, J, F, B, H, D, E, I, G \rangle$
6	$\langle B, H, F, D, E, A, G, C, I \rangle$
7	$\langle A, J, E, H, B, C, I, D, G \rangle$
8	$\langle B, C, G, E, A, H, D, I, J, F \rangle$
9	$\langle C, E, G, F, J, B, H, A, D \rangle$
10	$\langle B, C, A, J, D, E, G, H, F, I \rangle$

In this example, the sequential pattern mining algorithm instead of the original Apriori algorithm is applied [3][69]. Therefore, we use the Modified GSP algorithm to



discover the maximal frequent learning patterns as shown in Table F.2. The details can be found in [71]. For example, in  $L_4$ , we have discovered that one candidate of new learning sequence of good students is to learn B course content first and then to study the learning contents D, E, G sequentially. Hence, these candidates of various learning sequence will be suggested by *VODKA* for teachers to generate new variant learning sequence.

**Table F.2 The Maximal Frequent Learning Patterns of Good Students**

Large Itemset	Maximal Frequent Learning Patterns									
$L_2$	A→F	A→H	A→J	B→H	C→D	C→F	C→H	E→F	F→G	G→H
$L_3$	A→D→G	B→C→G								
$L_4$	B→D→E→G									

In this case study, we illustrate that *VODKA* can collect all interesting learning behaviors (learning sequences) of students whose testing grade from online quiz system is good; hence, then the maximal frequent learning sequence, a part of whole learning sequence, will be used to recommend teachers to adapt course material for variant learning behaviors if necessary.